Optimum Circuits for Bit-Dimension Permutations

Mario Garrido, Jesus Grajal and Oscar Gustafsson

The self-archived postprint version of this journal article is available at Linköping University Institutional Repository (DiVA):

<http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-158360>

N.B.: When citing this work, cite the original publication.

Garrido, M., Grajal, J., Gustafsson, O., (2019), Optimum Circuits for Bit-Dimension Permutations, *IEEE Transactions on Very Large Scale Integration (vlsi) Systems*, 27(5), 1148-1160.

<https://doi.org/10.1109/TVLSI.2019.2892322>

Original publication available at:

<https://doi.org/10.1109/TVLSI.2019.2892322>

Copyright: Institute of Electrical and Electronics Engineers (IEEE)

<http://www.ieee.org/index.html>

©2019 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Optimum Circuits for Bit-Dimension Permutations

Mario Garrido, *Member, IEEE*, Jesús Grajal, *Member, IEEE*, and Oscar Gustafsson, *Senior Member, IEEE*

Abstract—In this paper we presents a systematic approach to design hardware circuits for bit-dimension permutations. The proposed approach is based on decomposing any bit-dimension permutation into elementary bit-exchanges. Such decomposition is proven to achieve the theoretical minimum number of delays required for the permutation. This offers optimum solutions for multiple well-known problems in the literature that make use of bit-dimension permutations. This includes the design of permutation circuits for the fast Fourier transform (FFT), bit reversal, matrix transposition, stride permutations and Viterbi decoders.

Index Terms—Bit-dimension permutation, bit reversal, data management, FFT, matrix transposition, pipelined architecture, streaming data, Viterbi decoder.

I. INTRODUCTION

Bit-dimension permutations [1] are permutations on $N = 2^n$ data defined by a permutation of n bits that represent the index of the data in binary. Bit-dimension permutations are a wide category that includes, among other permutations, the perfect shuffle [2], matrix transposition [3], [4], stride permutations [5]–[8] and bit reversal [9]–[15], as shown in Fig. 1. Regarding their use, bit-dimension permutations are used in important signal processing algorithms such as the fast Fourier transform (FFT) [15]–[21] and Viterbi decoders [22].

One way to design digital circuits that carry out bit-dimension permutations is to use life-time analysis and register allocation [23], [24]. This approach determines the content of the registers used for the permutation at each time instant, leading to an efficient use of the registers.

More recent works use memories [3], [13], [25]–[31] or delays (buffers or registers) [3], [6]–[10] to carry out the permutations. The approaches based on memories consist of a memory bank in parallel, and multiplexers at the input and output of the memories. The multiplexers decide to/from which memory data is written/read. The approaches based on delays include delays and multiplexers in series and in parallel.

Nowadays there exist optimum solutions for bit-dimension permutations in terms of the number of delays [23], [24]. However, the interconnection between registers is complex and requires a large number of wires and multiplexers. There exist

Fig. 1. Classification of permutations.

also optimum solutions for specific bit-dimension permutations. For bit reversal, optimum circuits have been proposed for serial [9] and parallel data [10]. Likewise, there are circuits with the minimum number of delays for matrix transposition and other stride permutation [8]. However, there is no general solution in the literature that provides the minimum number of delays as well as a reduced number of multiplexers for any bit-dimension permutation.

In this work we present a systematic approach to design hardware circuits for bit-dimension permutations, different to the commonly used Kronecker products [8], [25]. The proposed approach leads to circuits with optimum number of delays for any bit-dimension permutation. This has several implications. First, the proposed approach widens the scope with respect to previous papers that only focus on specific permutations such as bit reversal, matrix transpositions, etc. Second, it provides optimum solutions in terms of delays for a wide range of permutations. Furthermore, it reduces the number of multiplexers with respect to previous approaches based on delays. As a result, the proposed approach is a systematic and optimized solution for a large group of permutations.

The paper is organized as follows. Section II briefly reviews the concept of bit-dimension permutations. Section III explains how to model a continuous data flow, which is the basis of the proposed approach. Section IV presents the circuits for elementary bit-exchange, which are the basic circuits that we use to carry out bit-dimension permutations. Section V describes how to calculate the cost of a bit-dimension permutation. Section VI presents the theoretical minimum latency and number of delays for a bit-dimension permutation. Section VII shows how to derive optimum circuits for bit-dimension permutations. Section VIII compares the proposed approach to previous approaches in the literature. Section IX summarizes the main conclusions and Appendix A shows an example on how to use the proposed approach.

II. BIT-DIMENSION PERMUTATIONS

Let us consider a set of $N = 2^n$ data, $n \in \mathbb{N}$, in an n -dimensional space $x_{n-1}x_{n-2} \dots x_0$, where $x_i \in \{0, 1\}$. In this

M. Garrido and O. Gustafsson are with the Dept. of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden, e-mails: mario.garrido.galvez@liu.se, oscar.gustafsson@liu.se

J. Grajal is with the Dept. of Signal, Systems and Radiocommunications, Universidad Politécnica de Madrid, 28040 Madrid, Spain, e-mail: jesus.grajal@upm.es

This work was supported by the Swedish ELLIIT Program, by the FPU Fellowship AP2005-0544 of the Spanish Ministry of Education, by the Spanish National Research and Development Program under Project TEC2014-53815-R and by the Madrid Regional Government under Project S2013/ICE-3000 (SPADERADAR-CM).

Copyright (c) 2019 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending an email to pubs-permissions@ieee.org

context, a bit-dimension permutation, σ , defines a reordering of the data according to a permutation of the coordinates in the space [1]. This allows for defining the permutation on a set of n elements instead of defining it for 2^n values, which is most times mathematically inaccessible [32].

In general, a bit-dimension permutation is a permutation

$$\sigma(u_{n-1}u_{n-2} \dots u_0) = u_{\sigma(n-1)}u_{\sigma(n-2)} \dots u_{\sigma(0)}, \quad (1)$$

which transforms a point in the space $u_{n-1}u_{n-2} \dots u_0$ into a new point $u_{\sigma(n-1)}u_{\sigma(n-2)} \dots u_{\sigma(0)}$ whose coordinates are a permutation of the coordinates of the original point. Before the permutation, $x_i = u_i$ and, after the permutation, $x_i = u_{\sigma(i)}$. Thus, $\sigma(i)$ defines the element $u_{\sigma(i)}$ that is moved to x_i . Likewise, the inverse permutation $\sigma^{-1}(i)$ indicates that the element in x_i is moved to $x_{\sigma^{-1}(i)}$. Finally, it is fulfilled that $\sigma \circ \sigma^{-1} = \sigma^{-1} \circ \sigma = \text{Id}$.

A. Types of bit-dimension permutations

A bit-dimension permutation that only involves two dimensions¹, x_j and x_k , and exchanges their coordinates is called *elementary bit-exchange* (EBE) [32]. This elementary bit-exchange can be represented as $\sigma : x_j \leftrightarrow x_k$ [9]. Throughout the paper, we also use the notation $(j \ k)$ to represent this elementary bit-exchange.

A *perfect shuffle* [2] is a circular permutation of one bit to the left:

$$\sigma_{PS}(u_{n-1}u_{n-2} \dots u_1u_0) = u_{n-2} \dots u_1u_0u_{n-1}. \quad (2)$$

Likewise, a *perfect unshuffle* is a circular permutation of one bit to the right:

$$\sigma_{PU}(u_{n-1}u_{n-2} \dots u_1u_0) = u_0u_{n-1}u_{n-2} \dots u_1. \quad (3)$$

A stride-by- 2^s permutation [5], [7] is a circular permutation of s bits to the left, and it can be expressed as a composition of s perfect shuffles:

$$\sigma_{S2^s} = \underbrace{\sigma_{PS} \circ \dots \circ \sigma_{PS}}_{s \text{ times}} = (\sigma_{PS})^s. \quad (4)$$

III. MODELLING A HARDWARE DATA FLOW

In this section, we propose a new model to describe a hardware data flow. The model considers a continuous data flow of $N = 2^n$ data in an n -dimensional space $x_{n-1}x_{n-2} \dots x_0$.

A. Serial and parallel dimensions

In a hardware circuit, data flows in series and/or in parallel. Data flowing in series are provided to the same terminal at different clock cycles. Data flowing in parallel are provided at the same time to different terminals.

Fig. 2 shows the definition of serial and parallel dimensions in a data flow. As a convention, data flows from left to right, x_0 to x_{p-1} are parallel dimensions and x_p to x_{n-1} are serial ones. This means that there are p parallel dimensions and $n-p$ serial dimensions. This also means that the data flow is modelled as a rectangle of 2^p data in parallel times 2^{n-p} in series.

¹In this paper, the word 'dimension' refers to a direction in which we can move in the space.

Fig. 2. Definition of serial and parallel dimensions in a data flow.

Fig. 3. Data flow with one parallel and two serial dimensions. Position is shown in parenthesis and is related to the time (t) and the terminal (T). The data indexes are shown by the boxed numbers without parenthesis. This data flow is defined by $\mathcal{P} \equiv b_0b_1|b_2$.

B. Position, time and terminal

In the data flow, we can define the position occupied by each datum. As the number of data is $N = 2^n$, the positions are numbered from 0 to $2^n - 1$ according to

$$\mathcal{P} = \sum_{i=0}^{n-1} x_i 2^i. \quad (5)$$

In other words, we can say that $\mathcal{P} \equiv x_{n-1}x_{n-2} \dots x_0$, where $(\equiv)$ relates the decimal and binary representations of a number.

In the data flow we can define the time of arrival and the terminal of the datum in any position $\mathcal{P}$. The time of arrival is calculated as

$$t(\mathcal{P}) = \sum_{i=p}^{n-1} x_i 2^{i-p}, \quad (6)$$

and the input terminal is

$$T(\mathcal{P}) = \sum_{i=0}^{p-1} x_i 2^i. \quad (7)$$

Note that $t(\mathcal{P})$ is the time of arrival relative to the arrival of the first sample at a given point of the circuit. Therefore, $t(\mathcal{P}) = 0$ means that the sample in position $\mathcal{P}$ is the first one to arrive at that point of the circuit.

Note that the input terminal is only determined by the parallel dimensions, whereas the time of arrival only depends on the serial ones. Besides, there exists a total of 2^p terminals, which are numbered from $T = 0$ to $T = 2^p - 1$ according to (7), and all the data arrive in 2^{n-p} clock cycles, from $t = 0$ to $t = 2^{n-p} - 1$ according to (6).

Finally, the vertical bar ($|$) is used throughout this paper to separate the serial and parallel dimensions. According to this, we can represent the position as

$$\mathcal{P} = t | T. \quad (8)$$

Example: Fig. 3 shows a data flow with three dimensions. One of them is parallel and two are serial. The position is indicated in parenthesis and numbered from 0 to 7. The time and

terminal are also indicated in the figure. For instance, position $\mathcal{P} = 5$ corresponds to $x_2x_1x_0 = 101$. As $p = 1$, the time of arrival is $t(\mathcal{P}) = 2 \equiv 10 = x_2x_1 = x_{n-1} \dots x_p$. Likewise, the terminal is $T(\mathcal{P}) = 1 \equiv 1 = x_0 = x_{p-1}, \dots, x_0$.

C. Data flow with indexed data

Signal processing algorithms define mathematical operations on indexed data. In our approach, we define the index of the data as $I \equiv b_{n-1}b_{n-2} \dots b_0$ or, equivalently,

$$I = \sum_{i=0}^{n-1} b_i 2^i. \quad (9)$$

Thus, I represents the decimal value of the index and b_i are the bits of its binary representation.

In a data flow with indexed data, the position is defined as a function of b_i . This allows to assign each indexed data to a position in the data flow.

Example: In the data flow of Fig. 3 the data indexes are shown by the boxed numbers without parenthesis and the positions are in parenthesis. The definition $\mathcal{P} \equiv b_0b_1|b_2$ allows to know the position of each index in the data flow. For instance, $I = 6 \equiv 110 = b_2b_1b_0$ is in position $\mathcal{P} \equiv b_0b_1|b_2 = 01|1 \equiv 3$. This also holds for any other index in the data flow.

D. Permuting a continuous data flow

A shuffling circuit is represented by a function σ . The permutation that a circuit carries out is defined as $\sigma(u)$, and can be applied to a data flow with or without indexed data. If the input order is $\mathcal{P}_0$ and the output order is $\mathcal{P}_1$, then:

$$\sigma(\mathcal{P}_0) = \mathcal{P}_1. \quad (10)$$

Example: $\sigma(u_2u_1|u_0) = u_2u_0|u_1$ defines the permutation of a shuffling circuit. When applying it to the input order $\mathcal{P}_0 \equiv b_0b_1|b_2$, we obtain the output order $\mathcal{P}_1 \equiv b_0b_2|b_1$.

IV. HARDWARE CIRCUITS FOR ELEMENTARY BIT-EXCHANGE

Any bit-dimension permutation can be decomposed into a series of elementary bit-exchanges [32]. This principle is followed in this paper to design circuits for bit-dimension permutation. This section describes the hardware circuits used to calculate elementary bit-exchanges. Later sections explain how to use the elementary bit-exchanges to create circuits for bit-dimension permutations.

A. General considerations

An elementary bit-exchange exchanges the coordinates of two dimensions, x_j and x_k . Without loss of generality, let us assume that $j > k$. If σ is an elementary bit-exchange that calculates $\sigma(\mathcal{P}_0) = \mathcal{P}_1$, then $\mathcal{P}_0$ and $\mathcal{P}_1$ only differ in the coordinates that are exchanged, i.e.,

$$\begin{aligned} \mathcal{P}_0 &\equiv u_{n-1} \dots u_{j+1}u_ju_{j-1} \dots u_{k+1}u_ku_{k-1} \dots u_0 \\ \mathcal{P}_1 &\equiv u_{n-1} \dots u_{j+1}u_ku_{j-1} \dots u_{k+1}u_ju_{k-1} \dots u_0. \end{aligned} \quad (11)$$

As $x_i \in \{0, 1\}$, samples for which $x_j = x_k$ will remain in the same position, as $\mathcal{P}_1 = \mathcal{P}_0$. Conversely, if $x_j \neq x_k$, the input position corresponds to one of these options:

$$\begin{aligned} \mathcal{P}_{0A} &\equiv u_{n-1} \dots u_{j+1}0u_{j-1} \dots u_{k+1}1u_{k-1} \dots u_0 \\ \mathcal{P}_{0B} &\equiv u_{n-1} \dots u_{j+1}1u_{j-1} \dots u_{k+1}0u_{k-1} \dots u_0. \end{aligned} \quad (12)$$

If the initial position is $\mathcal{P}_0 = \mathcal{P}_{0A}$, the elementary bit-exchange moves the sample to $\mathcal{P}_1 = \mathcal{P}_{0B}$. If $\mathcal{P}_0 = \mathcal{P}_{0B}$ the output position is $\mathcal{P}_1 = \mathcal{P}_{0A}$. Therefore, pairs of samples whose position only differ in x_j and x_k are swapped. As a result, an elementary bit-exchange changes the position of half of the N samples, i.e., those for which $x_j \neq x_k$. The rest of the samples are unaffected by the permutation and keep their positions.

Note that this holds independently of the serial or parallel nature of the dimensions. However, depending on this nature we can define three different cases: Either both dimensions are parallel, or both are serial, or one of them is serial and the other one is parallel. Each of these cases leads to different shuffling circuits, as shown next.

B. Parallel-parallel EBE

If x_j and x_k are parallel, then $p > j > k$. This leads to

$$\begin{aligned} \mathcal{P}_0 &\equiv u_{n-1} \dots u_p|u_{p-1} \dots u_j \dots u_k \dots u_0 \\ \mathcal{P}_1 &\equiv u_{n-1} \dots u_p|u_{p-1} \dots u_k \dots u_j \dots u_0, \end{aligned} \quad (13)$$

and the pairs of inputs whose position must be exchanged are

$$\begin{aligned} \mathcal{P}_{0A} &\equiv u_{n-1} \dots u_p|u_{p-1} \dots 0 \dots 1 \dots u_0 \\ \mathcal{P}_{0B} &\equiv u_{n-1} \dots u_p|u_{p-1} \dots 1 \dots 0 \dots u_0. \end{aligned} \quad (14)$$

As no serial dimensions are involved in the permutation, the difference in time between two inputs that must be switched is $\Delta t = t(\mathcal{P}_{0B}) - t(\mathcal{P}_{0A}) = 0$. Therefore, pairs of data whose position must be exchanged are received at the same time at different terminals of the shuffling structure. This means that the permutation can be carried out by rearranging the inputs at each time instant and the circuit does not need any delay element to store samples. Besides, according to (13), input data at terminal $T_0 \equiv u_{p-1} \dots u_j \dots u_k \dots u_0$ are always forwarded to $T_1 \equiv u_{p-1} \dots u_k \dots u_j \dots u_0$. Consequently, a parallel-parallel EBE can be simply carried out by an interconnection between each input terminal, T_0 , and the corresponding output one, T_1 .

C. Serial-serial EBE

If both dimensions x_j and x_k are serial, then $j > k \geq p$. This leads to

$$\begin{aligned} \mathcal{P}_0 &\equiv u_{n-1} \dots u_j \dots u_k \dots u_p|u_{p-1} \dots u_0 \\ \mathcal{P}_1 &\equiv u_{n-1} \dots u_k \dots u_j \dots u_p|u_{p-1} \dots u_0, \end{aligned} \quad (15)$$

and the pairs of inputs whose position must be exchanged are

$$\begin{aligned} \mathcal{P}_{0A} &\equiv u_{n-1} \dots 0 \dots 1 \dots u_p|u_{p-1} \dots u_0 \\ \mathcal{P}_{0B} &\equiv u_{n-1} \dots 1 \dots 0 \dots u_p|u_{p-1} \dots u_0. \end{aligned} \quad (16)$$

This means that pairs of input data that must be interchanged arrive at the same input terminal, because $T(\mathcal{P}_{0A}) = T(\mathcal{P}_{0B})$, and they are separated a constant number of clock cycles

$$\Delta t = t(\mathcal{P}_{0B}) - t(\mathcal{P}_{0A}) = (2^j - 2^k)/2^p. \quad (17)$$

Fig. 4. Basic circuit for a serial-serial EBE.

Fig. 4 shows the circuit to carry out a serial-serial EBE. It consists of a buffer of length

$$L = \Delta t = (2^j - 2^k)/2^p, \quad (18)$$

and two multiplexers controlled by the same control signal, S . The latency of the circuit is equal to the length of the buffer, i.e., $\text{Lat} = L$. This is the minimum number of delays that makes the circuit causal and, therefore, implementable.

The control signal of the multiplexers depends on the serial dimensions that are involved and is obtained as

$$S = \overline{x_j} \text{ OR } x_k. \quad (19)$$

Note that $S = 0$ only if $x_j = 1$ and $x_k = 0$, i.e., when a sample in position $\mathcal{P}_{0B}$ is at the input of the circuit and sample in position $\mathcal{P}_{0A} = \mathcal{P}_{0B} - \Delta t = \mathcal{P}_{0B} - L$ is at the output of the buffer. As $S = 0$ both samples are interchanged. Otherwise, $S = 1$ and data are not permuted.

When there exist parallel dimensions, the circuit is replicated in parallel for each input terminal. In this general case the total number of delays is

$$D(\sigma) = 2^p \cdot L = 2^p \cdot \Delta t = 2^j - 2^k. \quad (20)$$

Permutations of serial data are used for bit reversal [9] and for the serial commutator FFT [20], [21].

D. Serial-parallel EBE

When the dimension x_j is serial and x_k is parallel, then $j \geq p > k$. This leads to

$$\begin{aligned} \mathcal{P}_0 &\equiv u_{n-1} \dots u_j \dots u_p | u_{p-1} \dots u_k \dots u_0 \\ \mathcal{P}_1 &\equiv u_{n-1} \dots u_k \dots u_p | u_{p-1} \dots u_j \dots u_0, \end{aligned} \quad (21)$$

and the pairs of inputs whose position must be exchanged are

$$\begin{aligned} \mathcal{P}_{0A} &\equiv u_{n-1} \dots 0 \dots u_p | u_{p-1} \dots 1 \dots u_0 \\ \mathcal{P}_{0B} &\equiv u_{n-1} \dots 1 \dots u_p | u_{p-1} \dots 0 \dots u_0. \end{aligned} \quad (22)$$

Accordingly, pairs of input samples that must be interchanged arrive at different terminals, because $T(\mathcal{P}_{0A}) \neq T(\mathcal{P}_{0B})$. Likewise, they arrive at different time instants at the circuit, being

$$\Delta t = t(\mathcal{P}_{0B}) - t(\mathcal{P}_{0A}) = 2^{j-p}. \quad (23)$$

In order to do the swapping, the input sample at terminal $T(\mathcal{P}_{0A})$, which arrives first, must wait Δt clock cycles until the other sample arrives. Figure 5 shows the circuit that permutes a parallel dimension with a serial one. It consists of two buffers and two multiplexers, where the length of each buffer is directly determined by

$$L = \Delta t = 2^{j-p}, \quad (24)$$

Fig. 5. Basic circuit for a serial-parallel EBE.

TABLE I
COSTS OF ELEMENTARY BIT-EXCHANGES

Cost	Elementary Bit-Exchange		
	Serial-serial	Parallel-parallel	Serial-parallel
D	$2^j - 2^k$	0	2^j
L	$(2^j - 2^k)/2^p$	0	$2^j/2^p$
M	2^{p+1}	0	2^p

and the control signal is

$$S = x_j. \quad (25)$$

If there is more than one parallel dimension, the circuit in Fig. 5 is replicated in parallel 2^{p-1} times. This leads to a total number of delays

$$D(\sigma) = 2^{p-1} \cdot 2L = 2^j. \quad (26)$$

Examples of serial-parallel permutations can be found in the parallel feedforward FFT architectures [16], [17].

E. Implementation of the delays using memories

Although the circuits have been described in terms of delays, these delays can be implemented in hardware by memories that act as a buffers. Actually, several delays in the permutation circuits can be grouped together to form a bigger memory [33]. This reduces the power consumption and the area of the circuit if the number of delays is large [34].

V. COST OF A BIT-DIMENSION PERMUTATION

The costs of the circuits in Section IV are summarized in Table I. The cost is shown in terms of total number of delays, D , buffer length and latency, L , and multiplexers, M .

For any bit-dimension permutation, the number of delays is calculated from the cost of the EBEs that it consists of, according to

$$D(\sigma) = \sum_{q=1}^Q D(\sigma_q) - \sum_r \min(D(\sigma_r), D(\sigma_{r+1})), \quad (27)$$

where Q is the total number of elementary bit-exchanges and r corresponds to the values for which σ_r and σ_{r+1} are both serial-parallel elementary bit-exchanges that share the parallel dimension. The latency of a bit-dimension permutation is related to the number of delays by the number of parallel samples $P = 2^p$, i.e., $\text{Lat} = D(\sigma)/2^p$. Finally, the number of

Fig. 6. Alternative circuits to calculate the permutation $\sigma(u_2u_1|u_0) = u_1u_0|u_2$. (a) $\sigma = \sigma_1 \circ \sigma_2$. (b) $\sigma = \sigma_3 \circ \sigma_1$. (c) $\sigma = \sigma_2 \circ \sigma_3$

multiplexers is equal to the sum of the number of multiplexers of the individual permutations, i.e.,

$$M(\sigma) = \sum_{q=1}^Q M(\sigma_q). \quad (28)$$

Example: The perfect shuffle $\sigma(u_2u_1|u_0) = u_1u_0|u_2$ can be calculated in three ways depending on how we break down the permutation, i.e., $\sigma = \sigma_1 \circ \sigma_2 = \sigma_3 \circ \sigma_1 = \sigma_2 \circ \sigma_3$, where $\sigma_1 : x_1 \leftrightarrow x_0$, $\sigma_2 : x_2 \leftrightarrow x_1$ and $\sigma_3 : x_2 \leftrightarrow x_0$ are elementary bit-exchanges. The permutations σ_1 and σ_3 are serial-parallel and the permutation σ_2 is serial-serial. According to Table I and considering that $p = 1$, the number of delays and multiplexers of the elementary bit-exchanges is

$$\begin{aligned} D(\sigma_1) &= 2^1 = 2 & M(\sigma_1) &= 2 \\ D(\sigma_2) &= 2^2 - 2^1 = 2 & M(\sigma_2) &= 4 \\ D(\sigma_3) &= 2^2 = 4 & M(\sigma_3) &= 2. \end{aligned} \quad (29)$$

By implementing the elementary bit-exchanges according to Section IV, the three circuits to calculate σ in Fig. 6 are obtained. The number of delays for the three implementations according to equation (27) is

$$\begin{aligned} D(\sigma_1 \circ \sigma_2) &= D(\sigma_1) + D(\sigma_2) = 2 + 2 = 4 \\ D(\sigma_3 \circ \sigma_1) &= D(\sigma_3) + D(\sigma_1) - D(\sigma_1) = 4 + 2 - 2 = 4 \\ D(\sigma_2 \circ \sigma_3) &= D(\sigma_2) + D(\sigma_3) = 2 + 4 = 6. \end{aligned} \quad (30)$$

In the case of $\sigma_3 \circ \sigma_1$, both permutations are serial-parallel, so we subtract the minimum cost among them, which is $D(\sigma_1)$. This fact can be observed in Fig. 6(b), where two delays in the parallel branches between multiplexers can be removed thanks to pipelining, leading to a total of 4 delays. The latency is then $\text{Lat} = 4/2^1 = 2$ clock cycles for $\sigma_1 \circ \sigma_2$ and $\sigma_3 \circ \sigma_1$, and $\text{Lat} = 6/2^1 = 3$ for $\sigma_2 \circ \sigma_3$. Finally, the number of multiplexers is:

$$\begin{aligned} M(\sigma_1 \circ \sigma_2) &= M(\sigma_1) + M(\sigma_2) = 2 + 4 = 6 \\ M(\sigma_3 \circ \sigma_1) &= M(\sigma_3) + M(\sigma_1) = 2 + 2 = 4 \\ M(\sigma_2 \circ \sigma_3) &= M(\sigma_2) + M(\sigma_3) = 4 + 2 = 6 \end{aligned} \quad (31)$$

As a result, the permutation with the lowest cost is $\sigma_3 \circ \sigma_1$.

VI. THEORETICAL LIMITS

A. Minimum latency

The latency of a permutation circuit is the difference between the time when the first input arrives to the circuit and the time when the first output is provided.

The time that a certain input is inside the permutation circuit, t_I , is equal to the time of departure, $t(\mathcal{P}_1)$, minus the time of arrival, $t(\mathcal{P}_0)$, plus the circuit latency. Note that the time of arrival/departure defined in Section III is referred to the arrival/departure of the first input/output. This gives:

$$t_I = t(\mathcal{P}_1) - t(\mathcal{P}_0) + \text{Lat} \geq 0. \quad (32)$$

where the time t_I needs to be greater than or equal to zero to make the circuit causal. This leads to:

$$\text{Lat} \geq t(\mathcal{P}_0) - t(\mathcal{P}_1), \quad \forall \text{ data}, \quad (33)$$

Therefore, the minimum latency is:

$$\text{Lat}_{\min} = \max(t(\mathcal{P}_0) - t(\mathcal{P}_1)). \quad (34)$$

Consequently, the minimum latency is set by the datum that arrives the latest with respect to the time in which it should be provided, which will force the rest of data to wait for it.

B. Minimum number of delays

The outputs of a circuit are provided $\text{Lat}_{\min}$ clock cycles after the inputs are received. The minimum number of delays is then equal to the amount of data stored during this time:

$$D_{\min} = \text{Lat}_{\min} \cdot P = \max(t(\mathcal{P}_0) - t(\mathcal{P}_1)) \cdot P, \quad (35)$$

where $P = 2^p$ is the number of parallel inputs. For serial data ($P = 1$), this lower bound was already derived in [23]. For the general case, we continue the analysis as follows.

By substituting equation (6) in (35) and taking into account that $x_i = u_i$ at the input and $x_i = u_{\sigma(i)}$ at the output according to (1), we obtain

$$D_{\min} = \max \left(\sum_{i=p}^{n-1} u_i 2^i - \sum_{i=p}^{n-1} u_{\sigma(i)} 2^i \right). \quad (36)$$

The permutation σ is a bijection and the relation between i and $\sigma(i)$ is the same as the relation between $\sigma^{-1}(i)$ and i . Therefore, by applying the variable change $i \rightarrow \sigma^{-1}(i)$ to the second summation we obtain

$$D_{\min} = \max \left(\sum_{i=p}^{n-1} u_i 2^i - \sum_{\sigma^{-1}(i)=p}^{n-1} u_i 2^{\sigma^{-1}(i)} \right). \quad (37)$$

As $u_i \in \{0, 1\}$, then each term of the first summation will add a positive number or zero and each term of the second summation will subtract a positive number or zero. When $2^i >$

$2^{\sigma^{-1}(i)}$, the sum of the terms corresponding to the same u_i will be positive or zero and if $2^i < 2^{\sigma^{-1}(i)}$ the sum of the terms corresponding to the same u_i will be negative. Therefore, the maximum will occur when $u_i = 1$ for $i > \sigma^{-1}(i)$ and $u_i = 0$ for $i < \sigma^{-1}(i)$. This leads to

$$D_{\min} = \sum_{\substack{i \geq p \\ i > \sigma^{-1}(i)}} 2^i - \sum_{\substack{\sigma^{-1}(i) \geq p \\ i > \sigma^{-1}(i)}} 2^{\sigma^{-1}(i)}. \quad (38)$$

This equation results in Algorithm 1, which is used to calculate the cost of the optimum permutation. Equation (39) shows how to apply Algorithm 1 to the permutation $\sigma(u_4 u_3 u_2 u_1 | u_0) = u_2 u_1 u_0 u_4 | u_3$. In this case, $p = 1$ and $\sigma^{-1}(u_4 u_3 u_2 u_1 | u_0) = u_1 u_0 u_4 u_3 | u_2$.

u_4	u_3	u_2	u_1	u_0	
16	8	4	2	0	Input weight
− 2	0	16	8	4	Output weight
14	8	−12	−6	−4	Subtraction
<div style="display: flex; justify-content: space-around; width: 100%;"> <div style="text-align: center;"> $\swarrow$ 22 </div> <div style="text-align: center;"> $\swarrow$ Total </div> </div>					

(39)

Finally, the upper bound of equation (38) for a permutation of $N = 2^n$ elements and $P = 2^p$ parallel streams is

$$D_{\text{UB}} = \begin{cases} N - 2\sqrt{N} + P, & P < \sqrt{N} \text{ and } n \text{ even,} \\ N - \sqrt{2N} - \sqrt{\frac{N}{2}} + P, & P < \sqrt{N} \text{ and } n \text{ odd,} \\ N - P, & P \geq \sqrt{N}. \end{cases} \quad (40)$$

Note that the number of delays is always smaller than N , i.e., $D_{\text{UB}} < N$.

Algorithm 1 Theoretical minimum number of delays

- 1: In a first row, write $u_{n-1}, \dots, u_0$.
 - 2: In a second row, write 2^i under each u_i . This is the weight at the input of the permutation. If u_i corresponds to a parallel dimension, the weight will be 0 instead.
 - 3: In a third row, write the weight at the output of the permutation. For each u_i the output weight is $2^{\sigma^{-1}(i)}$, being $\sigma^{-1}(i)$ the number of the bit in which u_i appears at the output of the permutation. If u_i corresponds to a parallel dimension at the output of the permutation, the weight will be 0 instead.
 - 4: In a fourth row, write the result of subtracting the numbers in the third row from the numbers in the second row, column by column. Then, in this row, underline the values that are positive.
 - 5: In a fifth row, write the sum of the positive values that are in the fourth row. This is the minimum number of delays. Alternatively, the absolute value of the sum of the negative numbers also corresponds to the minimum number of delays.
-

C. Dealing with cycles

Permutations can be broken down in cycles. Different cycles in bit-dimension permutation do not share any dimension. As cycles are not mixed, when calculating the minimum number

of delays according to Algorithm 1 each column can only correspond to one cycle. Thus, the minimum number of delays for a cycle is obtained by adding the positive values in the columns that correspond to that cycle. Likewise, the minimum number of delays in a bit-dimension permutation is obtained as the sum of the minimum number of delays of the individual cycles. The example in Appendix A clarifies all of this fact.

VII. OPTIMUM CIRCUITS FOR BIT-DIMENSION PERMUTATION

In this section, we propose a methodology to obtain optimum circuits for bit-dimension permutations. First, the permutation is broken down into cycles. Then, for each cycle, the optimum permutation is obtained, which depends on the serial or parallel nature of the dimensions involved.

A. Decomposing the permutation into cycles

For the optimization purpose, permutation cycles can be treated independently. Each cycle can be one among the following three types:

- The cycle only includes parallel dimensions.
- The cycle only includes serial dimensions.
- The cycle includes serial and parallel dimensions.

The case when the cycle only includes parallel dimensions is straightforward: The optimum circuit simply consists of connecting each input terminal to the corresponding output terminal, as discussed in Section IV-B. For the other two cases, the next sections show how to achieve the optimum circuit.

Note also that the number of EBEs of each cycle is one less than the number of dimensions that the cycle involves. Therefore, if c is the number of cycles in the permutation, the total number of EBEs of a permutation is

$$\#EBEs(\sigma) = n - c. \quad (41)$$

B. Cycles with only serial dimensions

1) *A problem with elevators*: The optimization for cycles with only serial dimensions is analogous to the following problem with elevators. Once the problem of elevators is understood, it is easy to apply it to our optimization problem.

Let us assume that a building has n elevators that can move between floors $F = 0$ and $F = n - 1$. Each elevator can be in any of the n floors. However, there is always one elevator at each floor.

Each elevator has a number. This number corresponds to the floor that the elevator must reach.

Elevators can move. The movement is done in pairs of elevators that change floor. This exchange is done to respect the rule of one elevator per floor. The cost of moving one elevator along several floors depends on the initial and the final floors. As long as the elevator moves towards its final floor the cost will be the same independent of the number of stops in intermediate floors. However, if an elevator gets further than its target floor, there will be an extra cost.

For this problem, we want to calculate the most efficient movements in order to make all the elevators reach their destination floors.

Fig. 7. All the cases in which two elevators moving in different directions can be.

Algorithm 2 Obtaining the optimum permutation for cycles with only serial dimensions

```

while 1 do
  done = 0;
  for j = 0 : n - 2 do
    for k = j + 1 : n - 1 do
      if j ≤ F(k) & F(k) < F(j) & F(j) ≤ k then
        Add (F(j) F(k)) to the sequence of EBEs
        Carry out the EBE (F(j) F(k))
        done = 1;
      if done = 1 then break;
    if done = 1 then break;
  if F = (n - 1 : 0) then break;

```

Solution: As pairs of elevators exchange floors, in order to move them it is necessary that one of them moves down and the other one moves up. It is also necessary that each of them reaches the floor where the other one was.

Fig. 7 shows all the cases in which two elevators moving in different directions can be. The elevator j is in floor $F(j)$ and aims to reach floor j . The elevator k is in floor $F(k)$ and aims to reach floor k . Without loss of generality, we consider that $j > k$. This means that the elevator j moves up and the elevator k moves down. In other words, it is fulfilled that $F(j) < j$ and $F(k) > k$.

Among the cases in Fig. 7, in (a), (b) and (c), j cannot reach $F(k)$ without surpassing its destination. In cases (c), (e) and (m), k can not reach $F(j)$ without surpassing its destination. And in case (o), a swap of the floors would only make the elevators further than their destination. Therefore, the cases that advance to the destinations without incurring in additional costs are (d), (g), (h) and (n). These cases share the properties: $j \geq F(k)$, $F(k) > F(j)$ and $F(j) \geq k$. Therefore, the minimum cost is achieved as long as the movements fulfill these properties.

2) *Optimizing cycles with only serial dimensions:* The optimization of cycles with only serial dimensions is done by translating it to a problem of the elevators. This is possible because for serial-serial permutations the cost of moving up (analogously down) any u_i from x_k to x_j is the same when

it is done directly, i.e., $D = 2^j - 2^k$, and when there are intermediate stops, e.g., by stopping in h , $j > h > k$, the cost is $D = (2^h - 2^k) + (2^j - 2^h) = 2^j - 2^k$ as before.

Once the problem has been translated into a problem with elevators, the next step is to identify allowed movements that respect the properties $j \geq F(k)$, $F(k) > F(j)$ and $F(j) \geq k$, which guarantee minimum cost. Each of these movements leads to a new building and each of these buildings creates a branch of a tree.

Then, the process repeats for each branch of the tree and continues until all the elevators reach their destination.

At the end, each branch of the tree represents an optimum permutation. The sequence of permutations to reach one optimum is obtained by following the tree from the top to the end of any branch.

If, instead of obtaining all optimum solutions, we only need one of them, Algorithm 2 obtains such permutation. This algorithm searches for feasible movements of the elevators and, when it finds one, it collects the EBE, does the swap corresponding to that EBE and continues from that point, i.e., it does not search all the optimum cases, but follows the first that it finds.

Example: Let us consider $\sigma(u_4 u_3 u_2 u_1 u_0) = u_1 u_4 u_0 u_2 u_3$. First, it is translated into the building with elevators at the top of Fig. 8. The floor in which each elevator starts is equal to the numbers on top of the building, which corresponds to the subindex i of u_i at the output of the given permutation.

An allowed movement is to swap the elevators 4 and 0, which are in floors 3 and 1, respectively. Another allowed movement is to swap the elevators 4 and 1, which are in floors 3 and 2, respectively. These two cases lead to the two buildings to the sides of the top one. Then, the process repeats for each of the resulting buildings. until the tree is finished.

At the end, any branch of the tree represents an optimum movement. For instance, by going from the top to the most left branch, we obtain the elementary bit-exchanges (3 1), (1 0), (2 1) and (4 3). It can be checked that this sequence of elementary bit-exchanges carry out the desired permutation and its cost in terms of delays is 17, which corresponds to the theoretical minimum in previous section.

3) *Proof of optimality:* To proof optimality we know that in our solution we only consider the movements (d), (g), (h) and

Fig. 8. Obtaining all the optimum permutations for $\sigma(u_4 u_3 u_2 u_1 u_0) = u_1 u_4 u_0 u_2 u_3$.

(n). All these movements move elevators closer to their final floor and guarantee that the final cost is optimum, since no cost apart from the minimum is introduced. Furthermore, by following any of these movements the final cost is the same, as it is independent on the stops in intermediate floor. What remains to proof is that at any step of the algorithm we can always apply at least one of these movements. Any step of the algorithm consists of one or more cycles. If a given cycle involves only two dimensions, then it will be equal to the case (g). If the cycle involves more than two dimensions, then the upper part of the cycle must look like (d) or (e). They, together with (g), are the only cases that can create the upper part of the cycle. The case (d) is already one of the valid movements. For (e), if $F(j)$ is the lowest floor in the cycle, i.e., $F(j) = F_{\min}$, the elevator j will go from the lowest to the highest floor. This forces that the bottom part of the cycle is closed with (h), which is a valid movement. For (e), if $F(j) > F_{\min}$, there will exist an elevator $h < F(j)$ that comes from $F(h) > F(j)$, which allows for reaching the floors under $F(j)$. Otherwise, the cycle could not be closed. In this case we can apply the movement (n) to the elevators j and h . Therefore, any step of the algorithm has at least a valid movement. This guarantees that the algorithms always reaches the optimum permutations.

C. Cycles with serial and parallel dimensions

1) Optimizing cycles with serial and parallel dimensions:

When a cycle includes serial and parallel dimensions, the circuit is optimized by using one of the parallel dimensions

as a pivot. Thus, all the elementary bit-exchanges are carried out between the pivot dimension and another dimension. This transforms all serial-serial EBEs into serial-parallel, which follows the ideas in Section V and results in less multiplexers and equal or less delays than using serial-serial permutations.

The order of the elementary bit-exchanges that must be carried out is obtained easily. Starting with the pivot dimension, the value u_i is moved to its corresponding place at the output. This not only allocates u_i in its place but also moves another $u_{i'}$ to the pivot dimension. Next, $u_{i'}$ is moved to its corresponding place at the output, and a new $u_{i''}$ is moved to the pivot dimension. The procedure continues in the same way until all the values reach their place. Note that $i' = \sigma^{-1}(i)$ and $i'' = \sigma^{-2}(i)$ according to the definition in Section II.

The previous procedure results in Algorithm 3. An example of application of this algorithm is shown in Appendix A.

2) *Proof of optimality:* All the resulting permutations are serial-parallel or parallel-parallel. By including the costs in Table I in (27), the cost of the resulting permutation is

$$D = \sum_{i \geq p} 2^i - \sum_{\substack{i \geq p \\ i' \geq p}} \min(2^i, 2^{i'}). \quad (42)$$

$$D = \sum_{\substack{i \geq p \\ i \geq i'}} 2^i + \sum_{\substack{i \geq p \\ i < i'}} 2^i - \sum_{i' > i \geq p} 2^i - \sum_{i > i' \geq p} 2^{i'}. \quad (43)$$

As $i' = \sigma^{-1}(i)$, this corresponds to the minimum number of delays in (38).

Algorithm 3 Obtaining the optimum permutation for cycles with at least one parallel dimension

- 1: Decide the pivot dimension x_i among the parallel dimensions of the cycle ($i < p$) and write the subindex i .
- 2: Starting with the pivot dimension, write in order the sequence of dimensions that are involved in the cycle. For instance, $i \rightarrow i' \rightarrow i'' \rightarrow i$ means that u_i moves to $x_{i'}$, $u_{i'}$ moves to $x_{i''}$ and $u_{i''}$ moves to x_i . Note that the cycle closes when the pivot dimension is found again.
- 3: Below each subindex different to that of the pivot dimension, write the subindex of the pivot dimension.
- 4: The EBEs of the optimum permutation appear from left to right.

D. Number of multiplexers

In cycles that only include serial dimensions, all the EBEs are serial-serial. Therefore, each serial path includes two multiplexers per EBE, leading to a total of

$$M = (s_C - 1)2P \quad (44)$$

where s_C is the number of serial dimensions in the cycle and $s_C - 1$ is equal to the number of EBEs in the cycle.

In cycles with at least one parallel dimension, the optimum permutation consists of a sequence of serial-parallel permutations. These permutations require one multiplexer per parallel branch and per serial-parallel permutation, i.e.,

$$M = s_C P \quad (45)$$

Based on this, for any bit-dimension permutation, the upper bound for the number of multiplexers in the proposed approach is

$$M_{UB} = 2P \log_2 \left(\frac{N}{2P} \right) \quad (46)$$

VIII. COMPARISON

Table II compares the cost of several bit-dimension permutations in terms of delays/memory and multiplexers. The first column shows the references. The second column shows if the approach is memory-based or delay-based. The remaining columns show the cost in terms of delays/memory and 2-input multiplexers for the permutations (A), (B), (C) and (D) under study.

The case (A) is a $P \times P$ matrix transposition with $P = \sqrt{N}$, which corresponds to $\sigma(u_{n-1} \dots u_{n/2} | u_{n/2-1} \dots u_0) = u_{n/2-1} \dots u_0 | u_{n-1} \dots u_{n/2}$. It is implemented with $n/2$ serial-parallel elementary bit-exchanges $\sigma : x_i \leftrightarrow x_{i+n/2}$, $i = 0, \dots, n/2 - 1$, and its cost is

$$D(\sigma) = \sum_{i=0}^{n/2-1} 2^{i+n/2} = N - \sqrt{N}. \quad (47)$$

$$M(\sigma) = \sum_{i=0}^{n/2-1} 2^p = P \log_2 P \quad (\text{given that } P = \sqrt{N}) \quad (48)$$

For this permutation, the proposed approach and other delay-based approaches in Table II have less complexity

than memory-based approaches either in the amount of delays/memory, or the number of multiplexers, or both.

The permutation (B) is a bit reversal of N data arriving in P parallel streams with $N > P^2$. By using the proposed approach, the bit reversal is broken down into the EBEs $\sigma_i : x_i \leftrightarrow x_{n-1-i}$, $i = 0, \dots, \lfloor n/2 - 1 \rfloor$.

For either parallel data with $N > P^2$ or serial data, the permutation consists of p serial-parallel EBEs $\sigma_i : x_i \leftrightarrow x_{n-1-i}$, $i = 0, \dots, p-1$, and $\lfloor n/2 - p \rfloor$ serial-serial EBEs $\sigma_i : x_i \leftrightarrow x_{n-1-i}$, $i = p, \dots, \lfloor n/2 - 1 \rfloor$. The cost of the circuit is

$$D(\sigma) = \sum_{i=0}^{p-1} 2^{n-1-i} + \sum_{i=p}^{\lfloor n/2-1 \rfloor} (2^{n-1-i} - 2^i). \quad (49)$$

which results in

$$D(\sigma) = \begin{cases} N - 2\sqrt{N} + P, & n \text{ even,} \\ N - \sqrt{2N} - \sqrt{\frac{N}{2}} + P, & n \text{ odd,} \end{cases} \quad (50)$$

and

$$M(\sigma) = p2^p + \lfloor n/2 - p \rfloor 2^{p+1} = \begin{cases} P \log_2 \left(\frac{N}{P} \right), & n \text{ even,} \\ P \log_2 \left(\frac{N}{2P} \right), & n \text{ odd,} \end{cases} \quad (51)$$

The resulting circuits for serial and parallel bit reversal using the proposed approach are the same as those in [9] and [10], respectively. Therefore, the proposed approach is capable of obtaining optimum circuits for bit reversal for any P , and [9], [10] are only specific cases of the framework provided in this paper.

Note also that the cost of the bit reversal permutation, both for serial and parallel data, corresponds to the upper bound defined in (40). This means that the bit reversal is the most costly bit-dimension permutation.

Compared to previous approaches in Table II, the proposed approach requires less delays/memory than previous memory-based approaches. As we consider $N > P^2$, the $P \log_2 P$ multiplexers in [27] are less than the $P \log_2 (N/P)$ multiplexers of the proposed approach. Therefore, there is a trade-off between delays/memory and multiplexers.

The permutation (C) is $\sigma(u_4 u_3 u_2 u_1 | u_0) = u_2 u_1 u_0 u_4 | u_3$, which is a stride permutation that has been used in [8]. Fig. 9(b) and Fig. 9(c) show the proposed solution and the timing diagram, respectively. In this case, memory-based approaches require less multiplexer at the cost of noticeably more delays/memory. All delay-based approaches require the theoretical minimum amount of delays/memory and the proposed approach requires the least amount of multiplexers among them.

The permutation (D) is $\sigma(u_4 u_3 u_2 | u_1 u_0) = u_3 u_0 u_1 | u_4 u_2$, which is not a stride permutation. The proposed solution is shown in Fig. 10. In this case, the proposed approach saves 68% of the memory and uses 50% more multiplexers with respect to [25], [27], and saves 37% of the memory plus 25% of the multiplexers with respect to [26], [27]. Previous delay-based approaches do not consider this permutation [8] or require a large number of multiplexers [24].

Finally, there are some general conclusions. On the one hand, the proposed approach reduces the memory requirement

TABLE II
COST OF IN TERMS OF DELAYS/MEMORY AND MULTIPLEXERS OF SEVERAL BIT-DIMENSION PERMUTATIONS.

Approach	Circuit Type	(A) Matrix transposition (†)		(B) Bit reversal (*)		(C)		(D)	
		D/Mem.	Mux.	D/Mem.	Mux.	D/Mem.	Mux.	D/Mem.	Mux.
Püschel [25]	Memory	$2N$	$2P \log_2 P$	$2N$	$2P \log_2 P$	64	4	64	16
Serre [27]	Memory	$2N$	$P \log_2 P$	$2N$	$P \log_2 P$	64	4	64	8
Chen [26], Serre [27]	Memory	N	$2P \log_2 P$	N	$2P \log_2 P$	32	4	32	16
Takala [28], [29]	Memory	N	NP	N	NP	32	NP	NA	NA
Majumdar [24]	Delays	$N - \sqrt{N}$	High	$N - 2\sqrt{N} + P$	High	22	35	20	34
Järvinen [6], [8]	Delays	$N - \sqrt{N}$	$P \log_2 P$	NP	NP	22	14	NP	NP
Proposed	Delays	$N - \sqrt{N}$	$P \log_2 P$	$N - 2\sqrt{N} + P$	$P \log_2(\frac{N}{P})$	22	8	20	12

NP: Not provided. NA: Not applicable. (†): For the case where $P = \sqrt{N}$. (*): For $N > P^2$.

with respect to memory-based approaches and in most cases the reduction is significant. This is derived from Fig. 11, which shows the maximum and mean number of delays of the proposed circuits among all the permutations with the corresponding dimensions and parallelization, normalized to N . As some memory-based approaches [26]–[29] require a total memory of N , the values of the graph correspond to the ratio between the delays/memory of the proposed approach and in those memory-based approaches.

On the other hand, the proposed approach reduces the number of multiplexers compared to previous delay-based approaches [8], [24], while having the minimum number of delays/memory. It also widens the scope, as some previous approaches [8] restrict to strides.

IX. CONCLUSIONS

This paper has presented a new approach to design optimum circuits for bit-dimension permutations. It consists in breaking down any permutation into elementary bit-exchanges in an optimum way and, then, implementing these elementary bit-exchanges with hardware circuits.

In order to achieve optimum results, the paper analyzes the cost of the bit-dimension permutations in terms of the number of delays. A methodology to calculate this minimum number of delays and obtain the corresponding circuit is proposed.

Comparison to previous approaches shows that the proposed approach reduces the delays/memory with respect to previous memory-based approach and the number of multiplexers with respect to previous delay-based approaches.

APPENDIX A PRACTICAL CASE

This section illustrates the entire procedure to design the circuits for bit-dimension permutations. For this purpose, we consider the permutation $\sigma(u_7 u_6 u_5 u_4 u_3 u_2 u_1 | u_0) = u_6 u_1 u_0 u_3 u_5 u_7 u_2 | u_4$. This permutation has $p = 1$ parallel dimensions and, therefore, $P = 2^p = 2$.

The permutation σ has two cycles that involve $\{x_7 x_6 x_2 x_1\}$ and $\{x_5 x_4 x_3 x_0\}$, respectively, which is shown in Fig. 12. According to (41), the circuit consists of $n - c = 8 - 2 = 6$ EBEs, 3 for each cycle. Note that the first cycle only involves serial dimensions, whereas the second one involves serial and parallel dimensions.

The latency and the number of delays and multiplexers can be calculated as follows. Following Algorithm 1, the number of delays is $D_{\min} = 166$ according to

$$\begin{array}{cccccccc}
 u_7 & u_6 & u_5 & u_4 & u_3 & u_2 & u_1 & u_0 \\
 128 & 64 & 32 & 16 & 8 & 4 & 2 & 0 \\
 - & 4 & 128 & 8 & 0 & 16 & 2 & 64 & 32 \\
 \hline
 124 & -64 & 24 & 16 & -8 & 2 & -62 & -32 \\
 \hline
 & \searrow & & \searrow \swarrow & & \swarrow & & \\
 & & & 166 & & & &
 \end{array} \quad (52)$$

From the total number of delays, the first cycle has $124 + 2 = 126$ delays and the second cycle $24 + 16 = 40$. For clarity, in equation (52) the columns that correspond to the first cycle are highlighted.

The latency of the circuit is obtained from (35) as $\text{Lat}_{\min} = D_{\min}/P = 166/2 = 83$ clock cycles.

As the first cycle only involves serial dimensions, the number of multiplexers is obtained from equation (44), leading to $M(\sigma_1) = (s_{C1} - 1)2P = (4 - 1) \cdot 2 \cdot 2 = 12$. Likewise, the second cycle involves serial and parallel dimensions and the number of multiplexers is obtained from equation (45), leading to $M(\sigma_2) = s_{C2}P = 3 \cdot 2 = 6$. As a result, the total number of multiplexers is $M(\sigma) = M(\sigma_1) + M(\sigma_2) = 12 + 6 = 18$.

The next step is to calculate the EBEs of the permutation. For the first cycle, we have the exercise with elevators shown in Fig. 13(a). One solution to this problem is the sequence of EBEs (7 6), (2 1), (6 2), which require 64, 2 and 60 delays, leading to the expected total of 126 delays.

For the second cycle, there is only one parallel dimension, which we use as pivot dimension. According to Algorithm 3, we obtain

$$\begin{array}{ccccccc}
 0 & \rightarrow & 5 & \rightarrow & 3 & \rightarrow & 4 & \rightarrow & 0 \\
 & & 0 & & 0 & & 0 & &
 \end{array} \quad (53)$$

which leads to the sequence of EBEs (5 0), (3 0), (4 0). According to (27), this sequence of EBEs requires $32 + 8 + 16 - 8 - 8 = 40$ delays, which corresponds to the expected value.

Finally, the obtained EBEs are implemented with the circuits in Section IV, leading to the circuit in Fig. 13(b). Note that, as the cycles are independent, the order of the circuits that calculate the cycles can be exchanged.

REFERENCES

- [1] D. Fraser, "Array permutation by index-digit permutation," *J. Assoc. Comp. Machinery (ACM)*, vol. 23, no. 2, pp. 298–309, Apr. 1976.

Fig. 9. Circuits for the permutation $\sigma(u_4 u_3 u_2 u_1 | u_0) = u_2 u_1 u_0 u_4 | u_3$. (a) Using the theory of stride permutations in [8] (corresponds to Fig. 14(f) in that paper). (b) Using the proposed approach. (c) Timing diagram of the proposed approach.

- [2] H. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, no. 2, pp. 153–161, Feb. 1971.
- [3] M. Garrido, "Efficient hardware architectures for the computation of the FFT and other related signal processing algorithms in real time," Ph.D. dissertation, Universidad Politcnica de Madrid, Dec. 2009.
- [4] I. de Lotto and D. Dotti, "Large-matrix-ordering technique with applications to transposition," *Electronics Letters*, vol. 9, no. 16, pp. 374–375, Aug. 1973.
- [5] J. Granata, M. Conner, and R. Tolimieri, "Recursive fast algorithm and the role of the tensor product," *IEEE Trans. Signal Process.*, vol. 40, no. 12, pp. 2921–2930, Dec. 1992.
- [6] T. Jrvinen, "Systematic methods for designing stride permutation interconnections," Ph.D. dissertation, Tampere Univ. of Technology, Nov. 2004.

Fig. 10. Circuit for the permutation $\sigma(u_4 u_3 u_2 | u_1 u_0) = u_3 u_0 u_1 | u_4 u_2$. (a) Solution using the proposed approach. (b) Input order. (c) Output order.

Fig. 11. Average and maximum number of delays of the proposed approach normalized to N as a function of the number of dimensions and the parallelization.

- [7] T. Jrvinen, P. Salmela, H. Sorokin, and J. Takala, "Stride permutation networks for array processors," in *Proc. IEEE Int. Application-Specific Syst. Arch. Processors Conf.*, Sep. 2004, pp. 376–386.
- [8] T. Jrvinen, P. Salmela, H. Sorokin, and J. Takala, "Stride permutation networks for array processors," *J. VLSI Signal Process. Syst.*, vol. 49, no. 1, pp. 51–71, Oct. 2007.
- [9] M. Garrido, J. Grajal, and O. Gustafsson, "Optimum circuits for bit reversal," *IEEE Trans. Circuits Syst. II*, vol. 58, no. 10, pp. 657–661, Oct. 2011.
- [10] C. Cheng and F. Yu, "An optimum architecture for continuous-flow parallel bit reversal," *IEEE Signal Process. Lett.*, vol. 22, no. 12, pp. 2334–2338, Dec 2015.
- [11] W. Li, F. Yu, and Z. Ma, "Efficient circuit for parallel bit reversal," *IEEE Trans. Circuits Syst. II*, vol. 63, no. 4, pp. 381–385, Apr. 2016.
- [12] C.-M. Chen, C.-C. Hung, and Y.-H. Huang, "An energy-efficient partial FFT processor for the OFDMA communication system," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 2, pp. 136–140, Feb. 2010.
- [13] S.-J. Huang, S.-G. Chen, M. Garrido, and S.-J. Jou, "Continuous-flow parallel bit-reversal circuit for MDF and MDC FFT architectures," *IEEE Trans. Circuits Syst. I*, vol. 61, no. 10, pp. 2869–2877, Oct. 2014.
- [14] R. Chen and V. K. Prasanna, "Optimal circuits for parallel bit reversal," in *Proc. IEEE Design Automation Conf.*, Jun. 2017, pp. 1–6.
- [15] Y.-N. Chang, "An efficient VLSI architecture for normal I/O order pipeline FFT design," *IEEE Trans. Circuits Syst. II*, vol. 55, no. 12, pp. 1234–1238, Dec. 2008.
- [16] M. Garrido, J. Grajal, M. A. Snchez, and O. Gustafsson, "Pipelined radix-2^k feedforward FFT architectures," *IEEE Trans. VLSI Syst.*, vol. 21, no. 1, pp. 23–32, Jan. 2013.
- [17] M. Garrido, S. J. Huang, and S. G. Chen, "Feedforward FFT hardware architectures based on rotator allocation," *IEEE Trans. Circuits Syst. I*,

Fig. 12. Cycles of the permutation $\sigma(u_7u_6u_5u_4u_3u_2u_1|u_0) = u_6u_1u_0u_3u_5u_7u_2|u_4$. The two cycles involve the dimensions $\{x_7x_6x_2x_1\}$ and $\{x_5x_4x_3x_0\}$, respectively.

(a)

(b)

Fig. 13. Permutation $\sigma(u_7u_6u_5u_4u_3u_2u_1|u_0) = u_6u_1u_0u_3u_5u_7u_2|u_4$. (a) Problem with elevators for the cycle with only serial dimensions. (b) Final permutation circuit.

vol. 65, no. 2, pp. 581–592, Feb. 2018.

- [18] Y. Chen, Y. Lin, Y. Tsao, and C. Lee, “A 2.4-GSample/s DVFS FFT processor for MIMO OFDM communication systems,” *IEEE J. Solid-State Circuits*, vol. 43, no. 5, pp. 1260–1273, May 2008.
- [19] Y.-W. Lin and C.-Y. Lee, “Design of an FFT/IFFT processor for MIMO OFDM systems,” *IEEE Trans. Circuits Syst. I*, vol. 54, no. 4, pp. 807–815, Apr. 2007.
- [20] M. Garrido, S.-J. Huang, S.-G. Chen, and O. Gustafsson, “The serial commutator (SC) FFT,” *IEEE Trans. Circuits Syst. II*, vol. 63, no. 10, pp. 974–978, Oct. 2016.
- [21] M. Garrido, N. K. Unnikrishnan, and K. K. Parhi, “A serial commutator fast Fourier transform architecture for real-valued signals,” *IEEE Trans. Circuits Syst. II*, vol. 65, no. 11, pp. 1693–1697, Nov. 2018.
- [22] D. Akopian, J. Takala, J. Saarinen, and J. Astola, “Multistage interconnection networks for parallel Viterbi decoders,” *IEEE Trans. Commun.*, vol. 51, no. 9, pp. 1536–1545, Sep. 2003.
- [23] K. K. Parhi, “Systematic synthesis of DSP data format converters using life-time analysis and forward-backward register allocation,” *IEEE Trans. Circuits Syst. II*, vol. 39, no. 7, pp. 423–440, Jul. 1992.
- [24] M. Majumdar and K. K. Parhi, “Design of data format converters using two-dimensional register allocation,” *IEEE Trans. Circuits Syst. II*, vol. 45, no. 4, pp. 504–508, Apr. 1998.
- [25] M. Püschel, P. A. Milder, and J. C. Hoe, “Permuting streaming data using RAMs,” *J. ACM*, vol. 56, no. 2, pp. 10:1–10:34, Apr. 2009.
- [26] R. Chen and V. K. Prasanna, “Automatic generation of high throughput energy efficient streaming architectures for arbitrary fixed permutations,” in *Proc. Int. Workshop Field-Programmable Logic Applications*, Sep. 2015, pp. 1–8.
- [27] F. Serre, T. Holenstein, and M. Püschel, “Optimal circuits for streamed linear permutations using RAM,” in *Proc. ACM/SIGDA Int. Symp. FPGAs*, ACM, Feb. 2016, pp. 215–223.
- [28] J. Takala, T. Jarvinen, and H. Sorokin, “Conflict-free parallel memory access scheme for FFT processors,” in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 4, May 2003, pp. IV-524–IV-527.
- [29] J. Takala and T. Järvinen, *Stride Permutation Access In Interleaved Memory Systems*, ser. Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation, S. Bhattacharyya, E. Deprettere and J. Teich. CRC Press, 2003.

- [30] T. Koehn and P. Athanas, “Arbitrary streaming permutations with minimum memory and latency,” in *Proc. IEEE/ACM Conf. Comput.-Aided Design*, Nov. 2016, pp. 1–6.
- [31] T. E. Koehn, “Automatic generation of efficient parallel streaming structures for hardware implementation,” Ph.D. dissertation, Virginia Polytechnic Institute, Nov. 2016.
- [32] A. Edelman, S. Heller, and L. Johnsson, “Index transformation algorithms in a linear algebra framework,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 12, pp. 1302–1309, Dec. 1994.
- [33] M. Garrido, M. Acevedo, A. Ehliar, and O. Gustafsson, “Challenging the limits of FFT performance on FPGAs,” in *Int. Symp. Integrated Circuits*, Dec. 2014, pp. 172–175.
- [34] T. Ahmed, M. Garrido, and O. Gustafsson, “A 512-point 8-parallel pipelined feedforward FFT for WPAN,” in *Proc. Asilomar Conf. Signals Syst. Comput.*, Nov. 2011, pp. 981–984.

Mario Garrido (M’07) received the M.S. degree in electrical engineering and the Ph.D. degree from the Technical University of Madrid (UPM), Madrid, Spain, in 2004 and 2009, respectively. In 2010 he moved to Sweden to work as a postdoctoral researcher at the Department of Electrical Engineering at Linköping University. Since 2012 he is Associate Professor at the same department.

His research focuses on optimized hardware design for signal processing applications. This includes the design of hardware architectures for the calculation of transforms, such as the fast Fourier transform (FFT), circuits for data management, the CORDIC algorithm, and circuits to calculate statistical and mathematical operations. His research covers high-performance circuits for real-time computation, as well as designs for small area and low power consumption.

Jesús Grajal was born in Toral de los Guzmanes (León), Spain, in 1967. He received the Ingeniero de Telecomunicación and the Ph.D. degrees from the Technical University of Madrid, Madrid, Spain in 1992 and 1998, respectively. He is currently Professor at the Signals, Systems, and Radio Communications Department of the Technical School of Telecommunication Engineering of the Technical University of Madrid. His research activities are in the area of hardware-design for radar systems, radar signal processing and broadband digital receivers for radar, and spectrum surveillance applications.

Oscar Gustafsson (S’98–M’03–SM’10) received the M.Sc., Ph.D., and Docent degrees from Linköping University, Linköping, Sweden, in 1998, 2003, and 2008, respectively.

He is currently an Associate Professor and Head of the Computer Engineering Division, Department of Electrical Engineering, Linköping University. His research interests include design and implementation of DSP algorithms and arithmetic circuits. He has authored and coauthored over 140 papers in international journals and conferences on these topics.

Dr. Gustafsson is a member of the VLSI Systems and Applications and the Digital Signal Processing technical committees of the IEEE Circuits and Systems Society. Currently, he serves as an Associate Editor for the IEEE Transactions on Circuits and Systems Part II: Express Briefs and Integration, the VLSI Journal. He has served and serves in various positions for conferences such as ISCAS, PATMOS, PrimeAsia, Asilomar, Norchip, ECCTD, and ICECS.