Continuous-Flow Matrix Transposition Using Memories

Mario Garrido, *Senior Member, IEEE*, and Peter Pirsch, *Life Fellow, IEEE*

***Abstract*— In this paper, we analyze how to calculate the matrix transposition in continuous flow by using a memory or group of memories. The proposed approach studies this problem for specific conditions such as square and non-square matrices, use of limited access memories and use of several memories in parallel. Contrary to previous approaches, which are based on specific cases or examples, the proposed approach derives the fundamental theory involved in the problem of matrix transposition in a continuous flow. This allows for obtaining the exact equations for the read and write addresses of the memories and other control signals in the circuits. Furthermore, the cases that involve non-square matrices, which have not been studied in detail in the literature, are analyzed in depth in this paper. Experimental results show that the proposed approach is capable of transposing matrices of 8192 x 8192 32-bit data received in series at a rate of 200 mega samples per second, which doubles the throughput of previous approaches.**

***Index Terms*— Continuous flow, external memory, matrix transposition, pipelined architecture, SDRAM.**

I. Introduction

MATRIX transposition is an essential operation in a wide range of signal processing applications. To a large extent, this is due to the fact that it is used for the calculation of multidimensional transforms. This makes it a key component for the 2D fast Fourier transform (FFT) in image processing and machine vision [1], multiple-input multiple-output (MIMO) [2], [3], automotive [4] and synthetic aperture radars [5]–[7]. Likewise, it is required for the 3D FFT in molecular dynamics [8], motion detection [9]; for the 2D discrete cosine transform (DCT) in image compression [10], [11]; for the 2D fast Hartley transform (FHT) in image processing and circular convolution [12], [13]; and for the 3D fast Wavelet transform (FWT) in video encoding [14]. Additionally, matrix transposition is considered in convolutional neural networks (CNN) [15], [16] for artificial intelligence.

In a digital system that processes a continuous flow of data [7], [17]–[19], matrix transposition is used to change data in "row by row" order to "column by column" order. The circuit to do this transposition can be based on registers [20] or memories. In the current paper, we explore the use of memories.

Designing the matrix transposition circuit using memories involves multiple challenges. First, the calculation of matrix transposition in a continuous flow means that the circuit receives a series of matrices one after the other. A strategy to handle this is to use double buffering [18], [21], [22]. This requires two memories of size N, where N is the total number of elements in the matrix. One of the memories stores and transposes the even matrices in the flow and the other one handles the odd ones. More recent techniques improve this approach by making use of a single memory of size N [10], [17], [23], [24].

Second, the access to the memory may be limited. For small data sizes, either registers [25] or small memories that can be read or written arbitrarily may be used [10]. On the contrary, large memories such as a synchronous dynamic random-access memories (SDRAM) [26] are not so easily addressable. For them, the access consists in selecting a row of the memory and reading or writing samples in columns of this row. A change in the row leads to an important overhead due to the fact that several commands need to be executed before new data can be read or written. To circumvent this issue, different strategies to access the memory have been proposed [17], [18], [23], [24], [27]–[30]. A first alternative is to write rows of the matrix in squares of the memory [27]. This, however, does not solve the unbalance between the times required to operate row by row and column by column in the memory. Another alternative is to store rows of the matrix in rows of the memory and then read several columns of the same row of the memory instead of a single sample [18]. However, as only the first sample is required when the first row is read, an auxiliary memory must be used for storing the rest of the samples until they have to be provided at the output. The size of the auxiliary memory will then be $\alpha \cdot N_R$, where α is the number of columns read together and N_R is the number of rows of the matrix. Finally, the most useful alternative is to store squares of the matrix in rows of the memory [17], [23], [24], [28]–[30]. This way, consecutive values of the matrix both in horizontal and vertical directions are stored in the same row of the memory and can be accessed without row changes.

Third, a group of memories in parallel are needed when the throughput of the system needs to be larger than the throughput of a single memory. This usually happens in combination

Manuscript received November 20, 2019; revised February 4, 2020 and March 14, 2020; accepted April 8, 2020. Date of publication April 28, 2020; date of current version September 2, 2020. This work was supported by the Ramón y Cajal Fellowship of the Spanish Ministry of Science, Innovation, and Universities, under Grant RYC2018-025384-I. This article was recommended by Associate Editor M. M. Kermani. *(Corresponding author: Mario Garrido.)*

Mario Garrido is with the Department of Electronic Engineering, ETSI de Telecomunicación, Universidad Politécnica de Madrid, 28040 Madrid, Spain (e-mail: mario.garrido@upm.es).

Peter Pirsch is with the Institute of Microelectronic Systems, Leibniz University Hannover, 30167 Hannover, Germany (e-mail: pirsch@ims.uni-hannover.de).

Digital Object Identifier 10.1109/TCSI.2020.2987736

Fig. 1. Calculation of a 2D transform in a continuous flow.

with the use of limited-access memories. The reason is that the overhead of activating and refreshing these memories reduces the throughput of the memory. In order to mitigate this problem, bank interleaving mapping strategies have been proposed [17], [23], [24].

Finally, the solutions proposed in the literature generally cover square matrices. However, the case of non-square matrices is only considered in a few works [17], [31], [32] and an in-depth analysis of this case has not been presented so far.

In this paper, we provide a detailed analysis of the matrix transposition in a continuous flow using memories under any combination of specific conditions: Square and non-square matrices, use of limited access memories and use of several memories in parallel. For all these cases, efficient solutions that require a total memory size of order $\mathcal{O}(N)$ are presented. Apart from a broad analysis, the proposed approach sets the theoretical fundamentals of matrix transposition in continuous flow. Whereas previous works study the problem based on examples, the proposed approach deepens in the mathematical and logical fundamentals of the problem. This allows for obtaining the exact equations for the read and write addresses of the memories and other control signals. Last but not least, the cases of non-square matrices, which have not been studied in depth in the literature, are analyzed in detail in this paper.

The paper is organized as follows. In Section II, we introduce the matrix transposition in a continuous flow. In Section III, we review the main notions for bit-dimension permutations using memories. Based on this, we analyze the transposition of square and non-square matrices in a continuous flow in Sections IV and V, respectively. Matrix transposition using limited access memories and using multiple memories are addressed in Sections VI and VII, respectively. In Section VIII, we provide a practical application of the proposed approach and in Section IX we show the experimental results. In Section X, we compare the proposed approach to previous ones. Finally, in Section XI, we summarize the main conclusions of the paper.

II. Matrix Transposition in a Continuous Flow

From an algorithmic point of view, the calculation of a 2D transform on a 2D data set can be carried out by a 1D transform of each row followed by a 1D transform of each column.

For a digital system that calculates the 2D transform in a continuous flow, Fig. 1 shows the modules involved in the computations. The first module calculates the row-wise 1D transform, whereas the last one calculates the column-wise 1D transform. These modules demand data arriving row by row and column by column, respectively. This makes it necessary to include an intermediate module to carry out a matrix transposition that transforms the row-wise order provided by the first module into the column-wise order demanded by the last module.

For the matrix transposition module, we have to take into account that it will process a continuous flow of 1 or several samples per clock cycle. Furthermore, the number of rows of the matrix, N_R, and columns, N_C, may be equal or different and, therefore, the matrix may be square or non-square. Additionally, the memory used to store the data during the transposition may be a random-access memory (RAM) where any address can be accesses or a memory with limited access, such as an SDRAM. Finally, multiple samples may be received in parallel every clock cycle, which requires the use of several memories in parallel. All these cases are studied in Sections IV to VII.

III. Review of Bit-Dimension Permutations

A bit-dimension permutation σ is a permutation on n bits that infers a permutation on $N = 2^n$ elements. It has the form [20]

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

where $u_{n-1}u_{n-2}\dots u_0$ are the bits permuted into $u_{\sigma(n-1)}u_{\sigma(n-2)}\dots u_{\sigma(0)}$. The position of an element is calculated as

$$\mathcal{P} = \sum_{i=0}^{n-1} x_i 2^i, \tag{2}$$

where x_{n-1} is the most significant bit and x_0 is the least significant bit. Thus, there are 2^n positions numbered from 0 to 2^n-1 and occupied by the N elements. And the bit-dimension permutation σ changes the position of these elements.

Matrix transposition is a type of bit-dimension permutation. For the case of a square matrix, it has the form

$$\sigma(u_{n-1}\dots u_{n/2}u_{n/2-1}\dots u_0) = u_{n/2-1}\dots u_0u_{n-1}\dots u_{n/2}. \tag{3}$$

This means that the upper half of the bits is exchanged with the lower half. If they correspond to rows bits and column bits of the matrix, respectively, the permutation carries out a transposition of the elements of the matrix.

Bit-dimension permutations, including matrix transposition, can be implemented by using registers or memories. The work [20] explains in detail the theory related to registers, whereas the theory related to memories was introduced in [17]. Next, we review the main ideas related to the latter.

A. Bit-Dimension Permutations Using Memories

Given a set of $N = 2^n$ data arriving in series, the time of arrival, t, can be counted by a counter $c_{n-1}, \dots, c_0$ where

$$t = \sum_{i=0}^{n-p-1} c_i 2^i. \tag{4}$$

The write address to the memory, W, and the read address, R, are obtained by a permutation of the bits of the counter according to

$$W = \sigma_W(c_{n-1}\dots c_1c_0),$$
$$R = \sigma_R(c_{n-1}\dots c_1c_0). \quad (5)$$

Any bit-dimension permutation σ can be calculated from the permutations that lead to the read and write addresses as

$$\sigma = \sigma_R^{-1} \circ \sigma_W. \quad (6)$$

In a continuous flow, it must be fulfilled that data are written into the memory in the same address that is being read. In other words, the write address for the i-th set of data is the same as the read address for the $(i-1)$-th set of data, i.e.,

$$\sigma_{W_i} = \sigma_{R_{(i-1)}}. \quad (7)$$

The read address of the i-th set of data is then calculated from (6) as

$$\sigma_{R_i} = \sigma_{W_i} \circ \sigma^{-1}. \quad (8)$$

IV. Transposition of Square Matrices in a Continuous Flow

For a square matrix, it is fulfilled that $N_R = N_C$. Let us consider that the total number of elements is $N = 2^n = N_RN_C$. When the matrix arrives in series row by row in a continuous flow, the matrix transposition is defined by (3). The size of the memory used for the transposition, M, must be $M \geq N$. Thus, a memory of $M = N$ addresses is usually used in order not to waste memory. Here, we assume that N_R, N_C and M are powers of 2.

The write address for the first set of data can be chosen to be equal to the value of the count. According to this, $W_1 = c_{n-1}\dots c_0$. Then, R_1 is calculated from (8) as $R_1 = c_{n/2-1}\dots c_0c_{n-1}\dots c_{n/2}$. According to (7), $W_2 = R_1$. By applying (8) again, $R_2 = c_{n-1}\dots c_0$ is obtained and from (7) $W_3 = R_2 = c_{n-1}\dots c_0$ is derived. Note that $W_3 = W_1$ and in general $W_{i+2} = W_i$. Likewise, $R_{i+2} = R_i$. This occurs because $\sigma^2 = \mathrm{Id}$.

As a result, the read and write addresses for the i-th set of data are

$$W_i = \begin{cases} c_{n-1}\dots, c_{n/2}c_{n/2-1}\dots c_0, & \text{if } i \text{ odd}, \\ c_{n/2-1}\dots c_0c_{n-1}\dots c_{n/2}, & \text{if } i \text{ even}, \end{cases} \quad (9)$$

$$R_i = \begin{cases} c_{n/2-1}\dots c_0c_{n-1}\dots c_{n/2}, & \text{if } i \text{ odd}, \\ c_{n-1}\dots c_{n/2}c_{n/2-1}\dots c_0, & \text{if } i \text{ even}. \end{cases} \quad (10)$$

Figure 2 shows how the memory address is obtained from the counter of the system. It can be checked that the address is the same for reading the i-th data set and writing the $(i+1)$-th one, due to the fact that the $(i+1)$-th data set is written at the same time that the i-th data set is read.

Regarding hardware resources, the circuit requires a memory of size N and no multiplexer. The circuit achieves a throughput of one sample per clock cycle in a continuous flow. As the circuit reads data of the i-th set whereas it writes the $i+1$ set, the latency of the circuit is N clock cycles, which is the time difference between two consecutive sets.

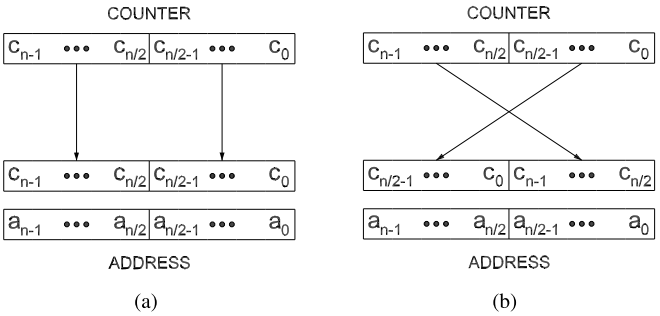


Fig. 2. Obtention of the read and write addresses for the transposition of square matrices in a continuous flow. (a) Write address for odd-indexed matrices and read address for even-indexed matrices. (b) Write address for even-indexed matrices and read address for odd-indexed matrices.

V. Transposition of Non-Square Matrices in a Continuous Flow

The transposition of non-square matrices is defined by the permutation

$$\sigma(u_{n-1}\dots u_ju_{j-1}\dots u_0) = u_{j-1}\dots u_0u_{n-1}\dots u_j, \quad (11)$$

where $j \neq n/2$. Here, $N_R = 2^{n-j} \neq N_C = 2^j$, and the each data set includes $N = 2^n = N_RN_C$ elements.

The period of the permutation, k, is the minimum natural number that fulfills $\sigma^k = \mathrm{Id}$. This number can be calculated as

$$k = \frac{n}{\gcd(n, j)}, \quad (12)$$

where $\gcd(n, j)$ is the greatest common divisor of n and j.

By applying (7) and (8), the read and write addresses for the i-th set of data are calculated as

$$W_i = \sigma_{W_1} \circ \sigma^{-(i-1)},$$
$$R_i = \sigma_{W_1} \circ \sigma^{-i}. \quad (13)$$

For $W_1 = \sigma_{W_1}(c_{n-1}\dots c_0) = c_{n-1}\dots c_0$, this simplifies to

$$W_i = \sigma^{-(i-1)},$$
$$R_i = \sigma^{-i}. \quad (14)$$

As σ has a periodicity k, $W_i = W_{\bmod(i,k)}$ and $R_i = R_{\bmod(i,k)}$. Thus, there are k periodical read and write addresses. This means that the controller that generates the address needs to select among k different permutations of the counter bits. When k is small, this selection can be done by using multiplexers. However, for large k the number of multiplexers may be large. A better alternative for large k is to use the circular counter in Fig. 3. This counter can be configured so that any of the bits can be the least significant bit (LSB) of the count. The control signal ctr of that bit is set to 1 and the rest of ctr signals are set to 0. This way, the address $a_{n-1}\dots a_0$ will be the shifted version of a counter, which is what is needed to generate the read and write addresses to the memory.

This circuit, as in the case of square matrices, requires a memory of size N and no multiplexer. The throughput is one sample per clock cycle and the latency is N clock cycles.

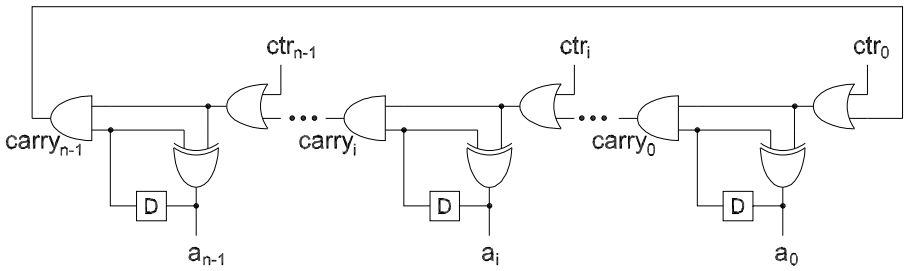


Fig. 3. Circular counter for the address generation for permutations with large k.

VI. Matrix Transposition Using Limited Access Memories

In the previous section it has been assumed that any memory address can be accessed arbitrarily. However, this is not always possible, as in case of SDRAM [26] memories. These memories organize the addresses in rows and columns, and the access is burst-oriented and row by row. This means that it is possible to access addresses in the same row at a relatively high rate. By contrast, the access column by column is slow.

A. Problem Formulation

The M addresses of the memory are distributed in M_R rows and M_C columns, being $M = M_R M_C$. Read and write operations in the memory must be done in bursts of L data in a single row.

As the matrix to be transposed must fit in the memory, it must be fulfilled that

$$N_R N_C \leq M_R M_C. \tag{15}$$

Here, the worst case $N_R N_C = M_R M_C$ is considered, and it is assumed that N, N_R, N_C, M_R, M_C and L are powers of 2. Note also that the fact that $M = N$ does not necessary mean that the memory and the matrix have the same number of rows and columns.

According to this, the challenge consist in finding a read and write strategy that respects the read and write conditions of the memory and achieves high throughput in continuous flow.

B. Square Matrices

In the limited access memory, data are read/written in bursts of L data in the same row. To guarantee that the row is the same during each burst, only column bits of the address can vary, whereas row bits of the address must remain constant. As in a group of L consecutive data only the $\lambda = \log_2 L$ least significant bits (LSB) change, these bits must be assigned to column bits of the memory address. This condition must be met for each read/write pattern, as shown next.

When transposing square matrices, there are two read/write patterns. According to (9) and (10), the read/write addresses can be one of the following cases

$$\begin{aligned} &c_{n-1} \ldots c_{n/2} c_{n/2-1} \ldots c_0, \\ &c_{n/2-1} \ldots c_0 c_{n-1} \ldots c_{n/2}. \end{aligned} \tag{16}$$

For each of the patterns, the λ LSBs must be assigned to column bits of the memory address:

$$\begin{aligned} &c_{n-1} \ldots c_{n/2} c_{n/2-1} \ldots c_\lambda \underbrace{c_{\lambda-1} \ldots c_0}_{\text{Mem. columns}}, \\ &c_{n/2-1} \ldots c_0 c_{n-1} \ldots c_{n/2+\lambda} \underbrace{c_{n/2+\lambda-1} \ldots c_{n/2}}_{\text{Mem. columns}}. \end{aligned} \tag{17}$$

This way, data in a burst do not modify row bits of the memory address.

The memory address is then obtained by the assignment shown in Fig. 4. Note that $c_{\lambda-1} \ldots c_0$ and $c_{n/2+\lambda-1} \ldots c_{n/2}$ are always assigned to column bits of the memory. This requires to fulfill the condition $2\lambda \leq h$, where $M_C = 2^h$, which is the same as

$$L^2 \leq M_C. \tag{18}$$

This condition limits the maximum burst length for a given M_C.

According to Fig. 4 the read and write addresses are

$$W_i = \begin{cases} c_{n-1} \ldots c_{n/2+\lambda} c_{n/2-1} \ldots c_\lambda \\ c_{n/2+\lambda-1} \ldots c_{n/2} c_{\lambda-1} \ldots c_0, & \text{if } i \text{ odd}, \\ c_{n/2-1} \ldots c_\lambda c_{n-1} \ldots c_{n/2+\lambda} \\ c_{\lambda-1} \ldots c_0 c_{n/2+\lambda-1} \ldots c_{n/2}, & \text{if } i \text{ even}, \end{cases} \tag{19}$$

$$R_i = \begin{cases} c_{n/2-1} \ldots c_\lambda c_{n-1} \ldots c_{n/2+\lambda} \\ c_{\lambda-1} \ldots c_0 c_{n/2+\lambda-1} \ldots c_{n/2}, & \text{if } i \text{ odd}, \\ c_{n-1} \ldots c_{n/2+\lambda} c_{n/2-1} \ldots c_\lambda \\ c_{n/2+\lambda-1} \ldots c_{n/2} c_{\lambda-1} \ldots c_0, & \text{if } i \text{ even}. \end{cases} \tag{20}$$

Fig. 5 shows how samples are read and written in the memory according to the assignment of Fig. 4. In Fig. 5, the allocation of the first row and the first column of the matrix is indicated by the two types of shaded regions. As can be observed, all the samples of every $L \times L$ square of the matrix are stored in the same row of the memory. These squares are represented by letters $A, B, \ldots, U, V, \ldots$.

Considering that data are provided row by row, the first L data that arrive at the system are those of the first row of A. For these data, all bits $c_{n-1}, \ldots, c_\lambda$ are equal to zero, so they are stored in the first row of the memory. Next, the data of the first row of B are received. In this case $c_\lambda = 1$. According to Fig. 4, c_λ is assigned to $a_{2\lambda}$, so room is made in the first row of the memory for the rest of data in A, which will be stored in the same row. Consequently, the first row of B starts to

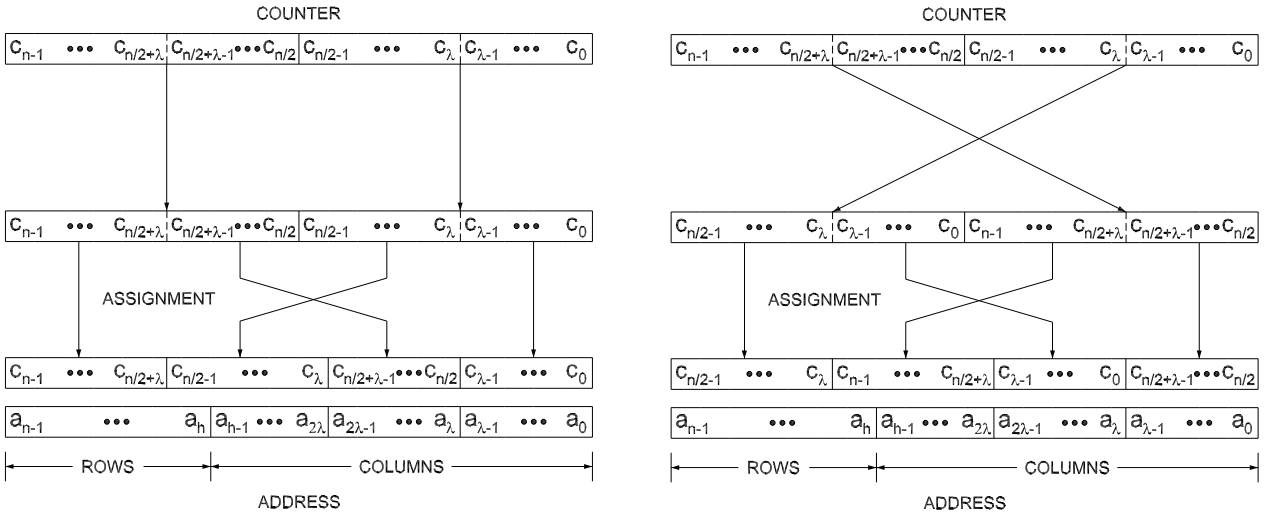


Fig. 4. Assignment of the bits of the counter to the bits of the address for the transposition of square matrices by using a memory with access limitations. (a) Write address for odd-indexed matrices and read address for even-indexed matrices. (b) Write address for even-indexed matrices and read address for odd-indexed matrices.

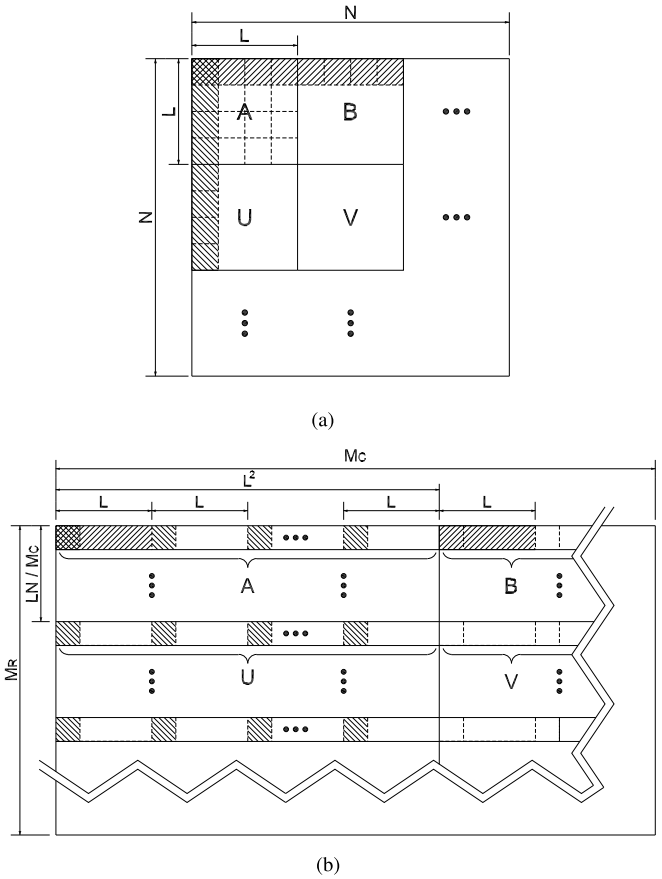


Fig. 5. Storage of a square matrix in a memory with access limitations. (a) Matrix. (b) Memory.

be stored after L^2 memory addresses. Likewise, L^2 memory addresses are also reserved for square B after those reserved for A. Once an entire row of the memory is reserved, the next square will be written in the following row of the memory.

When a whole row of the matrix has been written in the memory, the following row is stored in the address space reserved for the corresponding square. For example, the second row of A is stored in the first row of the memory after the first row of A. Once L rows of the matrix have been stored in the memory, the first LN/M_C rows of the memory will be full. Next, samples of squares U, V, and so on, will be written following the same strategy used for A and B.

As can be observed in Fig. 5, every group of L samples of the first column of the matrix is stored in the same row of the memory, as was desired. They are interleaved with those of other rows of the same square, but they can be read at a high rate since the row of the memory does not change. Consequently, the transposition of the matrix is carried out efficiently and using only a memory of size N and no multiplexer. This is possible because every sample is stored in the memory address released by the sample provided at the output at the same instant. This reduces the size of the memory with respect to double buffering strategies and simplifies control since data are written in those addresses that are being read.

The suggested address mapping allows for a throughput of one sample per clock cycle. To guarantee this rate, the access protocol in terms of the sequence of commands to access the limited access memory must support this rate. The solutions for this is based on making use of several memory banks, B, where each of the banks supports a rate of $1/B$ samples per clock cycle. This is studied later in Section VIII and applies to all the scenarios that use limited access memories.

C. Non-Square Matrices

For non-square matrices, $\sigma^k = \mathrm{Id}$ is fulfilled for a certain $k > 2$. By following the same strategy used for square matrices, it would be necessary that the λ bits of each of the k read/write patterns correspond to columns of the memory, which requires that $L^k \leq M_C$. This leads to a low value of L. Indeed, if k is large, the condition may not be fulfilled.

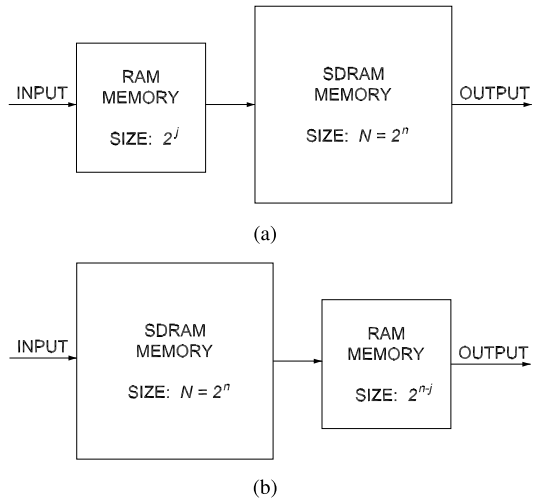


Fig. 6. Transposition of non-square matrices with access limitations. (a) Small RAM folowed by a large SDRAM for $j > n/2$. (b) Large SDRAM followed by a small RAM for $j < n/2$.

In this case, the strategy for square matrices presented in the previous section is not feasible for non-square matrices.

In order to solve this issue, we propose an alternative strategy that makes use of a large memory with limited access and a small auxiliary memory. The strategy consists in dividing the matrix transposition into a permutation for the large memory and a permutation for the small one.

Let us consider the matrix transposition in equation (11), which is a general equation for any size size of the matrix. When $j > n/2$, the matrix transposition can be calculated as $\sigma = \sigma_2 \circ \sigma_1$ where

$$\begin{aligned}\sigma_1(u_{n-1}\dots u_0) &= u_{n-1}\dots u_j u_{n-j-1}\dots u_0 u_{j-1}\dots u_{n-j},\\ \sigma_2(u_{n-1}\dots u_0) &= u_{j-1}\dots u_{2j-n} u_{n-1}\dots u_j u_{2j-n-1}\dots u_0.\end{aligned} \tag{21}$$

The permutation σ_1 is a perfect shuffle [33] of 2^j elements, whereas the permutation σ_2 fulfills $\sigma_2^2 = \mathrm{Id}$ and has a periodicity $k = 2$. Therefore, σ_2 can be implemented by using a limited access memory of size $M = N = 2^n$ and following the strategy for square matrices in previous section, whereas σ_1 is a smaller memory of size $J = 2^j$. Note that in this case the auxiliary memory is placed before the large limited access memory, as shown in Fig. 6(a).

If $j < n/2$, the permutation σ in equation (11) can be calculated as $\sigma = \sigma_2 \circ \sigma_1$ where

$$\begin{aligned}&\sigma_1(u_{n-1}\dots u_0)\\ &\quad = u_{n-j-1}\dots u_{n-2j} u_{n-1}\dots u_{n-j} u_{n-2j-1}\dots u_0,\\ &\sigma_2(u_{n-1}\dots u_0)\\ &\quad = u_{n-1}\dots u_{n-j} u_{n-2j-1}\dots u_0 u_{n-j-1}\dots u_{n-2j}.\end{aligned} \tag{22}$$

In this case, the permutation σ_1 has a periodicity $k = 2$ and uses the limited access memory, whereas σ_2 is calculated with an auxiliary memory of size $J = 2^{n-j}$. Note that in this case the auxiliary memory is placed after the large limited access memory, as shown in Fig. 6(b).

As the matrices are usually images, the ratio between the number of rows and columns will be close to 1, unless the image is extremely wide or high. Therefore, the size of the auxiliary memory will be close to $\sqrt{N}$, being significantly smaller than the large limited access memory.

This strategy requires a total memory of size $N + J$ and no multiplexer. It achieves a throughput of one sample per clock cycle and a latency of $N + J$ clock cycles.

VII. Using Multiple Memories in Parallel

Several memories in parallel must be used when the size of the matrix is larger than the size of a single memory and also when it is necessary to increase the throughput of the system. On the one hand, if the size of the matrix is larger than the size of the memory, the use of more memory chips serves to increase the total number of memory addresses, but does not add additional constraints. Thus, the group of memory chips can be considered as a single memory, where several bits of the address directly select the memory chip. On the other hand, the use of several memory chips for increasing the throughput demands a specific management of the data in order to make the most of the chips and use them simultaneously at the highest rate. The cases of square and non-square matrices are explained in Sections VII-A and VII-B, respectively.

A. Square Matrices

Let us assume that a continuous flow of sets of $N = 2^n$ data is received, where every clock cycle $P = 2^p$ data are provided in parallel. Thus, the data set arrives during 2^{n-p} clock cycles. According to this, the terminals are numbered as $T = 0 \dots 2^p - 1$ and the time of arrival of the N data is defined as $t = 0 \dots 2^{n-p} - 1$ [20]. Based on this, the control counter has $n - p$ bits, being $c_{n-p-1} \dots c_0$.

In this context, the matrix transposition is defined as

$$\begin{aligned}&\sigma(u_{n-1}\dots u_{n/2} u_{n/2-1}\dots u_p | u_{p-1}\dots u_0)\\ &= u_{n/2-1}\dots u_p u_{p-1}\dots u_0 u_{n-1}\dots u_{n/2+p} | u_{n/2+p-1}\dots u_{n/2},\end{aligned} \tag{23}$$

where the vertical bar is used to separate the serial and parallel parts [20]. Here, it is assumed that $n/2 > p$.

The solution in this case is based on splitting σ in

$$\sigma = \sigma_3 \circ \sigma_2 \circ \sigma_1, \tag{24}$$

where

$$\begin{aligned}&\sigma_1(u_{n-1}\dots u_{n/2} u_{n/2-1}\dots u_p | u_{p-1}\dots u_0)\\ &\quad = u_{n-1}\dots u_p | (u_{p-1} \oplus u_{n/2+p-1}) \dots (u_0 \oplus u_{n/2}),\\ &\sigma_2(u_{n-1}\dots u_{n/2} u_{n/2-1}\dots u_p | u_{p-1}\dots u_0)\\ &\quad = u_{n/2-1}\dots u_p (u_{p-1} \oplus u_{n/2+p-1}) \dots (u_0 \oplus u_{n/2}) u_{n-1}\dots\\ &\qquad \dots u_{n/2+p} | u_{p-1}\dots u_0,\\ &\sigma_3(u_{n-1}\dots u_{n/2} u_{n/2-1}\dots u_p | u_{p-1}\dots u_0)\\ &\quad = u_{n-1}\dots u_p | (u_{p-1} \oplus u_{n/2+p-1}) \dots (u_0 \oplus u_{n/2}).\end{aligned} \tag{25}$$

The permutations σ_1 and σ_3 only modify the parallel part, whereas σ_2 only modifies the serial part. Therefore, σ_1 and σ_3 can be implemented with multiplexers that route the data

in the parallel branches, whereas σ_2 can be implemented by using memories.

First, let us analyze the permutations σ_1 and σ_3. In parallel-parallel permutations, the circuit that carries out the permutation is obtained by defining the input and output terminals of the parallel part of the permutation [20]. By taking into account that $u_i = c_{i-p}$ for $i > p$, the input terminal for σ_1 and σ_3 is $T_0 = u_{p-1} \dots u_0$ and the output terminal is $T_1 = (u_{p-1} \oplus c_{n/2-1}) \dots (u_0 \oplus c_{n/2-p})$. The values $u_{p-1} \dots u_0$ define the terminal and are constant for each terminal. Thus, if $c_{n-p-1} \dots c_0 = 0 \dots 0$, then $T_1 = T_0$ and if $c_{n-p-1} \dots c_0 = 10 \dots 0$, then $T_1 = \bar{u}_{p-1} u_{p-2} \dots u_0$, i.e., T_1 is the terminal that results from negating the MSB of T_0.

The assignment of T_0 to T_1 is carried out with a multiplexer network as that in Fig. 7, which represents the case for $p = 3$. Each switching element consist of two multiplexers that make the inputs either continue in the same path or change paths. Each stage of multiplexers is controlled by one of the bit of the counter. If the bit is equal to 0, then the stage passes the inputs without changing their path. If the bit is equal to 1, the stage exchanges the paths. The hardware cost of this network is $P \log_2 P$ multiplexers.

Second, the permutation σ_2 only affects data in series. Thus, parallel paths are not mixed and the permutation is carried out with a group of P memories in parallel. By setting the writing address for the first data set to $W_1 = c_{n-p-1} \dots c_0$, then, according to (8) and taking into account that $u_i = c_{i-p}$ for $i > p$, the reading address is

$$R_1 = c_{n/2-p-1} \dots c_0 (u_{p-1} \oplus c_{n/2-1}) \dots \dots (u_0 \oplus c_{n/2-p}) c_{n-p-1} \dots c_{n/2}. \quad (26)$$

According to (7), $W_2 = R_1$ and, as $\sigma^2 = \mathrm{Id}$, then $R_2 = W_1$. This leads to

$$W_i = \begin{cases} c_{n-1}, \dots, c_0, & \text{if } i \text{ odd}, \\ c_{n/2-p-1} \dots c_0 (u_{p-1} \oplus c_{n/2-1}) \dots & \\ \dots (u_0 \oplus c_{n/2-p}) c_{n-p-1} \dots c_{n/2}, & \text{if } i \text{ even}, \end{cases} \quad (27)$$

$$R_i = \begin{cases} c_{n/2-p-1} \dots c_0 (u_{p-1} \oplus c_{n/2-1}) \dots & \\ \dots (u_0 \oplus c_{n/2-p}) c_{n-p-1} \dots c_{n/2}, & \text{if } i \text{ odd}, \\ c_{n-1}, \dots, c_0, & \text{if } i \text{ even}, \end{cases} \quad (28)$$

In theses equations, $u_{p-1} \dots u_0$ are constants that indicate the terminal or parallel path. Thus, the memories in parallel calculate different permutations with small variations.

Fig. 8 shows the circuit that obtains the read and write addresses to the memories. To write odd-indexed matrices and read even-indexed ones, the memory address is obtained directly from the bits of the counter, as shown in Fig. 8(a). To write even-indexed matrices and read odd-indexed ones, the upper $n/2 - p$ bits are exchanged with the lower $n/2 - p$ bits, and the intermediate bits of the address are obtained by connecting them with a wire or an inverter to the corresponding bits of the counter, depending on the constant value that each u_i takes. This, is shown in Fig. 8(b).

If the memories have limited access, the read and write addresses are obtained as in Section VI, i.e., the λ LSBs for each read and write pattern are assigned to columns of the memory.

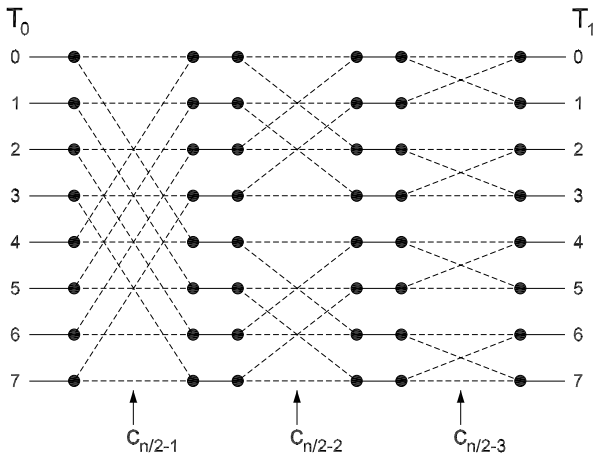


Fig. 7. Switching network for the permutations σ_1 and σ_3 in equation (25) for the particular case of $p = 3$.

Both for limited access memories and non limited access memories, the circuit requires a total memory of size N and $2P \log_2 P$ multiplexers. The throughput is P samples per clock cycle and the latency is N/P clock cycles.

B. Non-Square Matrices

The transposition of non-square matrices with $P = 2^p$ parallel paths is defined by the permutation

$$\sigma(u_{n-1} \dots u_j u_{j-1} \dots u_p | u_{p-1} \dots u_0) = u_{j-1} \dots u_p u_{p-1} \dots u_0 u_{n-1} \dots u_{j+p} | u_{j+p-1} \dots u_j. \quad (29)$$

Here, it is assumed that $n - p \geq j \geq p$.

As for square matrices, for non-square matrices the permutation σ is split in three permutations

$$\begin{aligned} &\sigma_1(u_{n-1} \dots u_p | u_{p-1} \dots u_0) \\ &\quad = u_{n-1} \dots u_p | (u_{p-1} \oplus u_{j+p-1}) \dots (u_0 \oplus u_j), \\ &\sigma_2(u_{n-1} \dots u_p | u_{p-1} \dots u_0) \\ &\quad = u_{j-1} \dots u_p (u_{p-1} \oplus u_{j+p-1}) \dots (u_0 \oplus u_j) u_{n-1} \dots \\ &\qquad \dots u_{j+p} | u_{p-1} \dots u_0, \\ &\sigma_3(u_{n-1} \dots u_p | u_{p-1} \dots u_0) \\ &\quad = u_{n-1} \dots u_p | (u_{p-1} \oplus u_{n-j+p-1}) \dots (u_0 \oplus u_{n-j}). \end{aligned} \quad (30)$$

The difference with respect to the case of square matrices is that now $\sigma_1 \neq \sigma_3$ and $\sigma_2^2 \neq \mathrm{Id}$. For σ_1, the following is obtained:

$$\begin{aligned} T_0 &= u_{p-1} \dots u_0, \\ T_1 &= (u_{p-1} \oplus c_{j-1}) \dots (u_0 \oplus c_{j-p}), \end{aligned} \quad (31)$$

and for σ_3,

$$\begin{aligned} T_0 &= u_{p-1} \dots u_0, \\ T_1 &= (u_{p-1} \oplus c_{n-j-1}) \dots (u_0 \oplus c_{n-j-p}). \end{aligned} \quad (32)$$

These permutation require the same network of multiplexers as in Fig. 7, with the only difference in the counter bits used for the commutators.

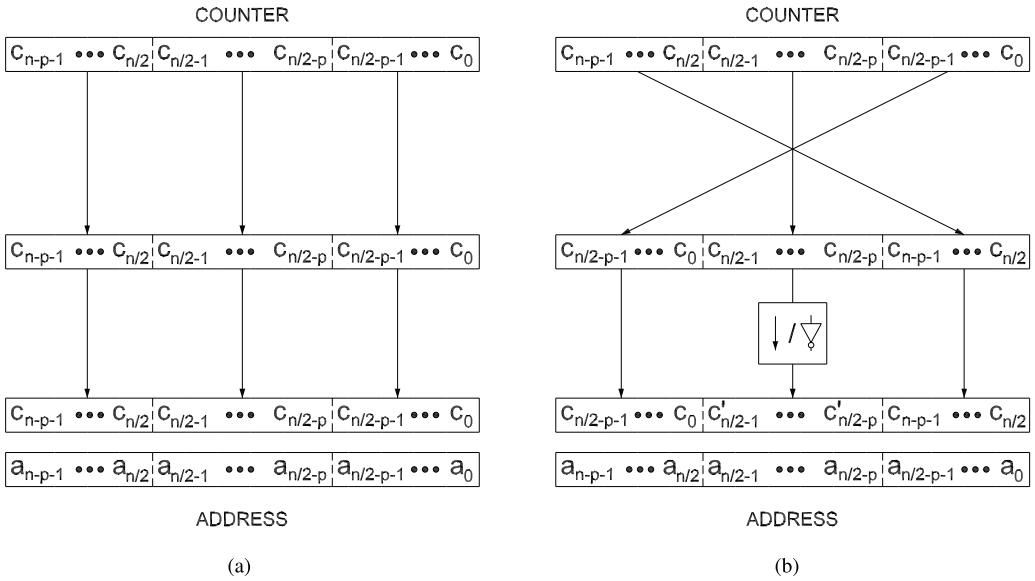


Fig. 8. Assignment of the bits of the counter to the memory address when using multiple memories in parallel to computer the matrix tranposition in a continuous flow. (a) Write address for odd-indexed matrices and read address for even-indexed matrices. (b) Write address for even-indexed matrices and read address for odd-indexed matrices.

The permutation σ_2 can also be implemented as the case of square matrices with the difference that its period is $k > 2$. This is solved by making use of the circular counter in Fig. 3.

Finally, if a memory with limited access is used, the permutation σ_2 can be split in the way explained in Section VI-C.

For non limited access memories, the circuit requires a total memory of size N and $2P \log_2 P$ multiplexers, has a throughput of P samples per clock cycle and a latency of N/P clock cycles. For limited access memories, the circuit uses a total memory of size $N + J$ and $2P \log_2 P$ multiplexers, has a throughput of P samples per clock cycle, and a latency of $(N + J)/P$ clock cycles.

VIII. Implementation

This section applies the previous ideas to a real system where matrices must be transposed in a continuous flow.

A. Specification

A pipelined 2D transform must be designed for the computation of a continuous flow of SAR images with a throughput of at least 40 MS/s. The size of the 2D transform is up to 8192×8192 points, and data are complex with $16 + 16$ bits of word length.

For the 1D transforms, there exist hardware architectures that can achieve throughputs much higher than 40 MS/s, so the bottleneck is to design the circuit for the matrix transposition.

The target FPGA and SDRAM memory are a Virtex-7 XC7VX330T -1 FFG1157 [34] and a dual data rate (DDR) SDRAM [26] with the following characteristics:

- Model: Micron MT46V32M16.
- Size: 512Mb (4 banks of 8M x 16-bit words).
- Programmable burst lengths: BL = 2, 4 or 8.
- Clock rate up to 200 MHz for Speed Grade -5B and CAS (READ) latency CL=3.

TABLE I

Command Times in ns and Clock Cycles for $f = 200$ MHz

Parameter	Symbol	Time	Cycles
PRECHARGE command period.	tRP	15 ns	3
AUTO REFRESH command period.	tRFC	70 ns	14
ACTIVE-to-READ or WRITE delay.	tRCD	15 ns	3
ACTIVE bank a to ACTIVE bank b	tRRD	10 ns	2
Write Recovery Time	tWR	15 ns	3
CAS Latency	CL	15 ns	3

The largest image has 8192×8192 32-bit data, which is equivalent to 128M 16-bit. This means that we need to make use of the 4 memory banks of 32M 16-bit words for the matrix transposition.

The configuration of the SDRAM for 16-bit words defines a memory where the addresses consist of 2 bits for the bank, 13 bits for the row and 10 bits for the column. Note that although 10 bits are used for selecting the column of the SDRAM, both the real and imaginary parts of the data are stored in consecutive memory addresses and, thus, the effective number of columns is $M_C = 2^9$.

The SDRAM memory is controlled by a set of commands, where the most important ones are: PRECHARGE, AUTO REFRESH, ACTIVE, READ, WRITE and NOP (No Operation). Times between commands are summarized in Table I. Rows of the memory banks are activated with ACTIVATE after a PRECHARGE. A new activation for the same bank requires to execute PRECHARGE and ACTIVATE again. Once a row of any of the banks is activated, it is possible to carry out any number of READ and WRITE operations on this row in bursts of length (BL). The READ and WRITE commands transfer 2 16-bit data per clock cycle, each of them in half of the clock period. Consequently, one complex sample can be read or written every clock cycle when the READ or the WRITE commands are executed. Finally, the AUTO REFRESH of

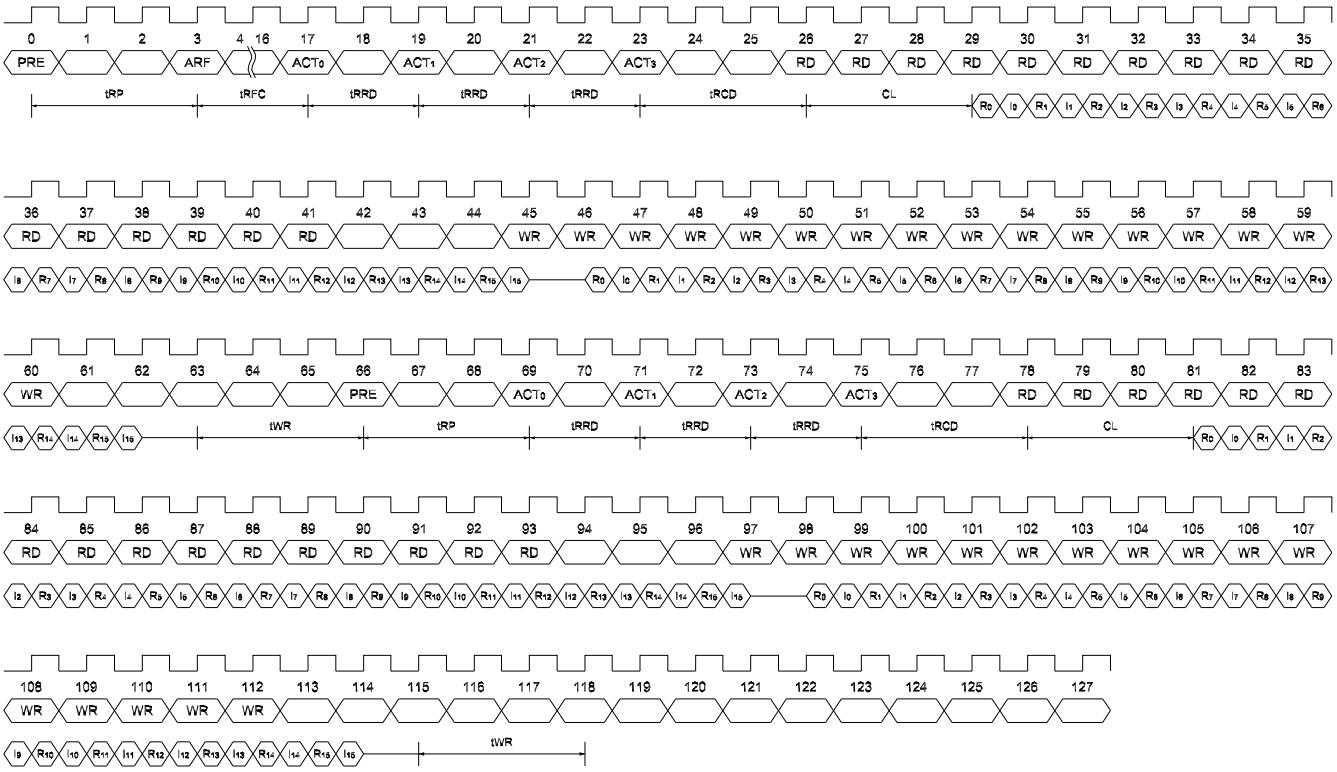


Fig. 9. Timing diagram of the SDRAM memory.

TABLE II
AREA OF THE SDRAM CONTROLLER FOR TRANSPOSING A CONTINUOUS FLOW OF MATRICES FOR VARIOUS MATRIX SIZES

Rows	Columns 128	256	512	1024	2048	4096	8192	Units
128	3515	3643	3751	3848				FF
	4206	4728	4944	5137	-	-	-	LUT
	0	1	1	1				BRAM
256	3671	3731	3889	3968	4089			FF
	4682	4852	5041	5463	5676	-	-	LUT
	1	0	1	1	2			BRAM
512	3786	3888	3952	4090	4123	4315		FF
	4942	5088	5379	5671	5744	5888	-	LUT
	1	1	0	1	2	4		BRAM
1024	3887	4005	4122	4143	4315	4401	4530	FF
	5091	5463	5672	5583	5881	6180	6323	LUT
	1	1	1	0	2	4	8	BRAM
2048	-	4125	4149	4338	4374	4530	4644	FF
		5675	5742	6024	6095	6335	6608	LUT
		2	2	2	0	4	8	BRAM
4096	-	-	4339	4438	4560	4589	4756	FF
			6031	6198	6337	6475	6803	LUT
			4	4	4	0	8	BRAM
8192	-	-	-	4563	4668	4806	4811	FF
				6342	6604	6802	6853	LUT
				8	8	8	0	BRAM

the memory has to be done periodically and executed after a PRECHARGE and before ACTIVATE.

B. Proposed Solution

In order to achieve continuous flow, in a period of T clock cycles T data must be read and written, which corresponds to

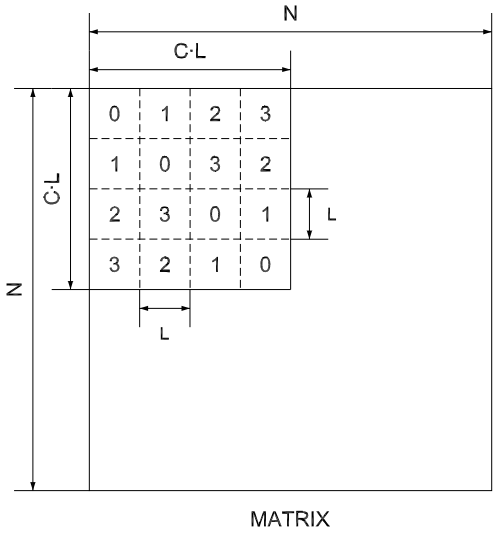


Fig. 10. Management of 4 memory chips. This scheme is adequate for reading and writing samples simultaneously in all the chips.

$2T$ operations. Each memory bank can read or write 1 sample per clock cycle, giving a total of 4 operations per clock cycle. According to this, the $2T$ operations are calculated in $T/2$ clock cycles and the remaining $T/2$ clock cycles are left for the rest of the commands. The proposed protocol is shown in Fig. 9. In it, $T = 128$ and a memory row change happens every 16 read and write operations. Note that first data are read and then new data are written in the same addresses. This means that in each group of 64 consecutive samples,

TABLE III
COMPARISON OF HARDWARE CIRCUITS FOR MATRIX TRANSPOSITION USING MEMORIES

Scenario			Shreesha [18]		Langemeyer [23]		Shang [10]		Baozhao [29]		Proposed	
Square matrix	Limited access	Parallel memories	Total mem.	Total mux	Total mem.	Total mux	Total mem.	Total mux	Total mem.	Total mux	Total mem.	Total mux
Yes	No	No	–	–	–	–	–	–	–	–	N	0
No	No	No	–	–	–	–	–	–	–	–	N	0
Yes	Yes	No	$2N$	0	N	0	–	–	N	0	N	0
No	Yes	No	–	–	–	–	–	–	–	–	$N+J$	0
Yes	No	Yes	–	–	–	–	N	$2P^2-2P$	–	–	N	$2P\log_2 P$
No	No	Yes	–	–	–	–	–	–	–	–	N	$2P\log_2 P$
Yes	Yes	Yes	–	–	N	Not Provided	–	–	–	–	N	$2P\log_2 P$
No	Yes	Yes	–	–	–	–	–	–	–	–	$N+J$	$2P\log_2 P$

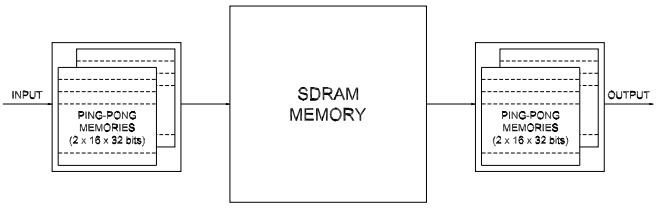


Fig. 11. Wrapper of the SDRAM.

16 will go to each memory. This is guaranteed by using the reading/writing scheme in Fig. 10, where $L = 16$, $C = 4$ is the number of chips and $C \cdot L = 64$. In this way, from every 64 consecutive data, 16 go to each bank, both row-wise and column-wise. Furthermore, as $L = 16$ and $M_C = 2^9$, the condition in equation (18) is met.

The burst length that we consider is $BL = 2$. Note the difference between BL, i.e., the number of values transferred in a read/write command, and L, i.e., the number of consecutive data stored in the same memory row.

In order to adapt the access protocol to the SRAM memories to 1 sample per clock cycle at the input and output of the system, small ping-pong memories are used for storing the input and the output data, as shown in Figure 11. They consist of 2 RAM memories of 16 words and 32 bits per word at the input and another two at the output of the SDRAM.

For non-square matrices, the system adds an auxiliary memory, as explained in Section VI-C. With this, the system works for any size of the matrix from 32×32 to 8192×8192 points and for non-square matrices whose the quotient between the larger and the smaller dimensions is $Q \leq 8$.

IX. EXPERIMENTAL RESULTS

Table II shows the experimental results of the designed controller on a Virtex-7 FPGA, XC7VX330T -1 FFG1157. The area of the controller ranges from 3515 to 4811 FFs and from 4206 to 6853 LUTs. Square matrices do not require additional BRAM memories, whereas non-square matrices use several BRAM memories. The controller works at 200 MHz. As the sytem processes one sample per clock cycle, it reaches a throughput of 200 MS/s, which is much higher than the expected 40 MS/s.

The latency of the circuit in clock cycles is

$$\text{Lat} = \begin{cases} N_R N_C + 134, & \text{if } N_R = N_C \\ N_R N_C + 134 + \max\{N_R, N_C\} + 1, & \text{if } N_R \neq N_C \end{cases} \quad (33)$$

Consequently, the latency mainly depends on the size of the matrix with a small overhead due to the auxiliary memories that are included.

At 200 MHz, the case of a 8192×8192 matrix takes $6.7 \cdot 10^7$ clock cycles to do the transposition, being the system capable of transposing 2.98 matrices of 8192×8192 32-bit data per second. The latency of this circuit is 0.33 s.

X. COMPARISON

Table III compares the hardware cost of matrix transposition circuits using memories. Previous approaches provide alternative solutions to some of the scenarios presented in this paper. First, the approaches in [18], [23], [29] target square matrices with limited access memories. They use either a single memory of size N [23], [29] or a double buffering strategy with two memories of size N [18]. In this case, the proposed approach achieves the same results in terms of memory and multiplexers as [23], [29].

Second, the case of transposing square matrices using non limited access memories in parallel is addressed in [10] and uses a total memory size of N as well as $2P^2 - 2P$ multiplexers. Compared to it, the proposed approach reduces the number of multiplexers to $2P \log_2 P$, while keeping a total memory of size N.

Third, a solution for square matrices with limited access memories in parallel is presented in [23]. While the memory size is N, the number of multiplexers is not reported. For this case, the proposed approach requires a memory of size N and $2P \log_2 P$ multiplexers.

In addition to this, the proposed approach provides solution for other scenarios, including the cases of non-square matrices not reported in previous works. For non-square matrices using limited access memories, the memory is N and the number of multiplexers is $2P \log_2 P$. For non-square matrices using limited access memories in parallel, the memory is $N + J$, where $J = 2^{n-j}$ if $j < n/2$ and $J = 2^j$ if $j > n/2$, and the number of multiplexers is $2P \log_2 P$.

The proposed approach can also be compared to previous circuits for matrix transposition based on registers. The costs in terms of memory and multiplexers for these cases are shown in Table IV. First of all, it can be noted that previous approaches based on registers only consider square matrices, but not non-square ones. Likewise, these cases are not applicable when the data set is large and a limited access memory is needed. For the case of square matrices and serial data, some works [20],

TABLE IV
COMPARISON OF HARDWARE CIRCUITS FOR MATRIX TRANSPOSITION USING REGISTERS

Scenario		Panchanathan [35]		Majumdar [36]		Järvinen [37], Garrido [20]		Wang [38]	
Square matrix	Parallel data	Total mem.	Total mux	Total mem.	Total mux	Total mem.	Total mux	Total mem.	Total mux
Yes	No	–	–	–	–	$(\sqrt{N}-1)^2$	$2\log_2 N$	$(\sqrt{N}-1)^2$	$2\sqrt{N}-2$
Yes	Yes, $P=\sqrt{N}$	N	P^2+2P	$N-\sqrt{N}$	High	$N-\sqrt{N}$	$P\log_2 P$	$N-\sqrt{N}$	P^2-P
Yes	Yes, any P	–	–	–	–	–	–	$N-2\sqrt{N}+P$	$P^2-3P+2\sqrt{N}$

TABLE V
COMPARISON OF EXPERIMENTAL RESULTS FOR CONTINUOUS-FLOW MATRIX TRANSPOSITION

Parameters	Langemeyer [23]	Shang [10]	Proposed
Matrix size	8192×8192	32×32	8192×8192
Limited access	Yes	No	Yes
Device	Virtex-5	-	Virtex-7
f_{CLK} (MHz)	100	94	200
$Th.$ (MS/s)	100	94	200
Slices	1542†	-	2259
BRAM	16†	-	0
FF	-	-	4811
LUT	-	-	6853
Latency (s)	-	-	0.33
Power (mW)	-	-	177

†: Uses 2 SDRAM controllers with 771 slices and 8 BRAM each.

[37] provide the optimum solution in terms of total memory, whereas the number of multiplexers is larger than in the proposed approach, which does not require any multiplexer. For the case of square matrices and $P = \sqrt{N}$, [20], [37] require less multiplexers and memory than the proposed approach. Finally, [38] provides an alternative for any P that is not restricted to $P = \sqrt{N}$. Compared to the proposed approach, the total memory is slightly smaller, but the number of multiplexers are of order $\mathcal{O}(P^2)$, whereas the proposed reduces the complexity to $\mathcal{O}(P \log P)$.

Regarding the experimental results, Table V compares the proposed approach to previous ones. Both the proposed approach and [23] allow for transposing matrices of 8192×8192 elements and, therefore, need to use limited access memories. Conversely, [10] transposes a smaller memory of 32×32 for which a limited access memory is not needed.

In terms of performance, the proposed approach doubles the throughput with respect to previous approaches. With respect to area, the proposed approach and [23] use a similar amount of resources, taking into account that the proposed approach occupies more slices but does not use BRAMs. Finally, the proposed approach has a latency of 0.33 s and a power consumption of 177 mW.

XI. CONCLUSIONS

This paper has analyzed the problem of matrix transposition in a continuous flow in a hardware system. The study is carried out from a theoretical perspective, which was missing in the literature. This allows to draw general conclusions and derive the exact equations for the read and write addresses of the memories and other control signals. Of particular interest are the cases of non-square matrices, which had not been studied in detail so far.

The comparison with previous works show that the proposed approach not only provides solutions for scenarios that have not been studied before, but also improves previous approaches in already studied scenarios.

REFERENCES

[1] F. Mahmood, M. Toots, L.-G. Ofverstedt, and U. Skoglund, "2D discrete Fourier transform with simultaneous edge artifact removal for real-time applications," in *Proc. Int. Conf. Field Program. Technol. (FPT)*, Dec. 2015, pp. 236–239.

[2] S. Jardak, S. Ahmed, and M.-S. Alouini, "Low complexity joint estimation of reflection coefficient, spatial location, and Doppler shift for MIMO-radar by exploiting 2D-FFT," in *Proc. Int. Radar Conf.*, Oct. 2014, pp. 1–5.

[3] S. Jardak, S. Ahmed, and M.-S. Alouini, "Low complexity moving target parameter estimation for MIMO radar using 2D-FFT," *IEEE Trans. Signal Process.*, vol. 65, no. 18, pp. 4745–4755, Sep. 2017.

[4] M. Song, J. Lim, and D.-J. Shin, "The velocity and range detection using the 2D-FFT scheme for automotive radars," in *Proc. 4th IEEE Int. Conf. Netw. Infrastructure Digit. Content*, Sep. 2014, pp. 507–510.

[5] M. Pfitzner, S. Langemeyer, P. Pirsch, and H. Blume, "A flexible real-time SAR processing platform for high resolution airborne image generation," in *Proc. IEEE CIE Int. Conf. Radar*, Oct. 2011, pp. 26–29.

[6] S. Zhang and C. Liu, "Two-dimensional non-uniform FFT for image formation of high-squint SAR," in *Proc. IEEE Geosci. Remote Sens. Symp.*, Jul. 2014, pp. 644–647.

[7] H. Izumi, K. Sasaki, K. Nakajima, and H. Sato, "An efficient technique for corner-turn in SAR image reconstruction by improving cache access," in *Proc. 16th Int. Parallel Distrib. Process. Symp.*, 2002, pp. 3–8.

[8] U. Nidhi, K. Paul, A. Hemani, and A. Kumar, "High performance 3D-FFT implementation," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2013, pp. 2227–2230.

[9] A. Kojima, N. Sakurai, and J. I. Kishigami, "Motion detection using 3D-FFT spectrum," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, Apr. 1993, pp. 213–216.

[10] Q. Shang, Y. Fan, W. Shen, S. Shen, and X. Zeng, "Single-port SRAM-based transpose memory with diagonal data mapping for large size 2-D DCT/IDCT," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 11, pp. 2422–2426, Nov. 2014.

[11] P. G. Fernandez, A. Garcia, J. Ramirez, and A. Lloris, "Fast RNS-based 2D-DCT computation on field-programmable devices," in *Proc. IEEE Workshop SiGNAL Process. SYSTEMS. SiPS. Design Implement.*, Oct. 2000, pp. 365–373.

[12] R. Kumaresan and P. K. Gupta, "Vector-radix algorithm for a 2-D discrete Hartley transform," *Proc. IEEE*, vol. 74, no. 5, pp. 755–757, Apr. 1986.

[13] J. S. Wu, H. Z. Shu, L. Senhadji, and L. M. Luo, "Radix-3×3 algorithm for the 2-D discrete Hartley ttRANSFORM," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 55, no. 6, pp. 566–570, Jun. 2008.

[14] R. Fernandez, J. M. Garcia, G. Bernabe, and M. E. Acacio, "Optimizing a 3D-FWT video encoder for SMPs and HyperThreading architectures," in *Proc. 13th Euromicro Conf. Parallel, Distrib. Network-Based Process.*, 2005, pp. 76–83.

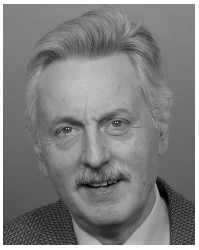
[15] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2016, *arXiv:1603.07285*. [Online]. Available: http://arxiv.org/abs/1603.07285

[16] D. Im, D. Han, S. Choi, S. Kang, and H.-J. Yoo, "DT-CNN: Dilated and transposed convolution neural network accelerator for real-time image segmentation on mobile devices," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[17] M. Garrido, "Efficient hardware architectures for the computation of the FFT and other related signal processing algorithms in real time," Ph.D. dissertation, Dept. Signals, Syst. Radiocommun., Univ. Politécnica de Madrid, Madrid, Spain, Dec. 2009.
[18] V. Shreesha and D. Nasoff, "Modular architecture for image transposition memory using synchronous DRAM," U.S. Patent 64 961 92, Dec. 17, 2002.
[19] L. J. Heung, "Device for generating memory address and mobile station using the address for writing/reading data," U.S. Patent 6788617, Sep. 7, 2004.
[20] M. Garrido, J. Grajal, and O. Gustafsson, "Optimum circuits for bit-dimension permutations," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 5, pp. 1148–1160, May 2019.
[21] R. B. Sheeparamatti, B. G. Sheeparamatti, M. Bharamagoudar, and N. Ambali, "Simulink model for double buffering," in *Proc. 32nd Annu. Conf. IEEE Ind. Electron. (IECON)*, Nov. 2006, pp. 4593–4597.
[22] Y.-W. Bai and C.-C. Liu, "The performance improvement of a photo card reader by the use of a high-integration chip solution with double FIFO buffers," *IEEE Trans. Consum. Electron.*, vol. 51, no. 2, pp. 329–334, May 2005.
[23] S. Langemeyer, P. Pirsch, and H. Blume, "Using SDRAMs for two-dimensional accesses of long $2^n \times 2^m$-point FFTs and transposing," in *Proc. Int. Conf. Embedded Comput. Syst., Archit., Modeling Simulation*, Jul. 2011, pp. 242–248.
[24] S. Langemeyer, P. Pirsch, and H. Blume, "Using SDRAM memories for high-performance accesses to two-dimensional matrices without transpose," *Int. J. Parallel Program.*, vol. 41, no. 2, pp. 331–354, Apr. 2013.
[25] M. El-Hadedy, X. Guo, M. Margala, M. R. Stan, and K. Skadron, "Dual-data rate transpose-memory architecture improves the performance, power and area of signal-processing systems," *J. Signal Process. Syst.*, vol. 88, no. 2, pp. 167–184, Aug. 2017.
[26] (Apr. 2014). *Micron-SDRAM Catalog*. [Online]. Available: http://www.micron.com/prod-ucts/dram/sdram
[27] Y. Dou, J. Zhou, Y. Lei, and X. Zhou, "FPGA SAR processor with window memory accesses," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2007, pp. 95–100.
[28] J. Zhu, P. Liu, and D. Zhou, "An SDRAM controller optimized for high definition video coding application," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2008, pp. 3518–3521.
[29] B. Tu, D. Li, and C. Han, "Two-dimensional image processing without transpose," in *Proc. 7th Int. Conf. Signal Process. (ICSP)*, Aug./Sep. 2004, pp. 523–526.
[30] H. Kim and I.-C. Park, "Array address translation for SDRAM-based video processing applications," *Electron. Lett.*, vol. 35, no. 22, pp. 1929–1931, 1999.
[31] M. W. Czekalski, "Corner turn memory address generator," U.S. Patent 4 484 265, Nov. 20, 1984.
[32] I. de Lotto and D. Dotti, "Large-matrix-ordering technique with applications to transposition," *Electron. Lett.*, vol. 9, no. 16, p. 374, 1973.
[33] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, no. 2, pp. 153–161, Feb. 1971.
[34] (Feb. 2018). *7 Series FPGAs Data Sheet: Overview*. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf
[35] S. Panchanathan, "Universal architecture for matrix transposition," *IEE Proc. E Comput. Digit. Techn.*, vol. 139, no. 5, p. 387, 1992.
[36] M. Majumdar and K. K. Parhi, "Design of data format converters using two-dimensional register allocation," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 45, no. 4, pp. 504–508, Apr. 1998.
[37] T. Järvinen, P. Salmela, H. Sorokin, and J. Takala, "Stride permutation networks for array processors," *J. VLSI Signal Process. Syst. Signal, Image, Video Technol.*, vol. 49, no. 1, pp. 51–71, Oct. 2007.
[38] Y. Wang, Z. Ma, and F. Yu, "Pipelined algorithm and modular architecture for matrix transposition," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 4, pp. 652–656, Apr. 2019.

Mario Garrido (Senior Member, IEEE) received the M.Sc. 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 Post-Doctoral Researcher with the Department of Electrical Engineering, Linköping University. From 2012 to 2019, he was an Associate Professor with the Department of Electrical Engineering. In 2019, he moved back to UPM, where he holds a Ramón y Cajal Research Fellowship. 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, rotators, such as the CORDIC algorithm, circuits to calculate statistical and mathematical operations, and neural networks. His research covers high-performance circuits for real-time computation, designs for small area, and low-power consumption.



Peter Pirsch (Life Fellow, IEEE) received the Dipl.Ing. and Dr.Ing. degrees in electrical engineering from the University of Hannover, in 1973 and 1979, respectively.

After some time in research organizations of the industry, he became a Professor with the Faculty of Electrical and Computer Engineering, Leibniz University of Hannover, in 1987. Since 2008, he has been a Emeritus Professor. He is the author or coauthor of more than 250 technical articles. His research includes VLSI architectures and VLSI implementations for high-performance signal processing applications with focus on image processing. He was a member or a chair of several technical program committees of international conferences and organizer of special sessions and preconference courses. He was a recipient of the IEEE Circuits and Systems Golden Jubilee Medal in 1999.