# Optimal Circuits for Streamed Linear Permutations Using RAM

François Serre, Thomas Holenstein, and Markus Püschel

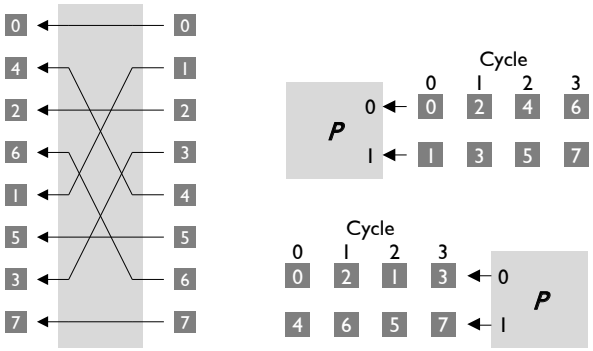
Department of Computer Science
ETH Zurich
{serref, holthoma, pueschel}@inf.ethz.ch

## ABSTRACT

We propose a method to automatically derive hardware structures that perform a fixed linear permutation on streaming data. Linear permutations are permutations that map linearly the bit representation of the elements addresses. This set contains many of the most important permutations in media processing, communication, and other applications and includes perfect shuffles, stride permutations, and the bit reversal. Streaming means that the data to be permuted arrive as a sequence of chunks over several cycles. We solve this problem by mathematically decomposing a given permutation into a sequence of three permutations that are either temporal or spatial. The former are implemented as banks of RAM, the latter as switching networks. We prove optimality of our solution in terms of the number of switches in these networks.

## Keywords

Streaming datapath; Data reordering; Connection network; Matrix factorization; Stride permutation; Matrix transposition; Bit-reversal




(a) not streaming (b) streaming

**Figure 1: Sketch of two implementations of the bit reversal permutation on $2^3$ elements. On the left, the structure has as many ports as the dataset. Thus a simple rewiring is enough. On the right side, data are streamed on two ports. Therefore, the dataset enters within 4 cycles (top), and is retrieved within 4 cycles (bottom).**

## 1. INTRODUCTION

Many algorithms and applications implemented on FPGAs require permutations or data reorderings as intermediate stages. If all data are available in one cycle, a hardware implementation is simply a set or wires as shown in Fig. 1(a)[1]. However, if data arrive streamed in chunks over several cycles as in Fig. 1(b), usually memory is required, as data may be reordered also in time. Accordingly, the efficient implementation becomes non-obvious [1, 2, 3, 4, 5].

In this paper, we present a method to implement streamed linear permutations (SLPs) on $2^n$ elements with proven minimal logic. Linear permutations are the permutations that operate as linear mappings on the bit representation of indices. They include many of the most important occurring permutations including stride permutations and the bit reversal. They are needed in fast Fourier transforms (FFTs; see Fig. 2(a)), fast cosine transforms, sorting networks (see Fig. 3(a)), Viterbi decoders, and many other applications.

Streamed means that the $2^n$ elements arrive in chunks of size $2^k$ over $2^t$ cycles, where $n = k+t$. Therefore, the resulting architecture has $2^k$ input and output ports. In Fig. 1(b), $2^t = 4$ and $2^k = 2$. Streaming permutations enable the implementation of designs that scale with large datasets (see Fig. 2(b) and 3(b) for instance) while maintaining a high throughput.

Our contribution is a systematic method to construct SLPs with proven minimal logic under the assumption that routing is done only by wires and $2\times2$-switches. Specifically:

- We prove a lower bound for the switching complexity for an SLP, i.e., for the number of switches needed.
- We provide a method to derive a (switching)-optimal SLP. The method decomposes a given linear permutation into a sequence of spatial and temporal permutations that can be implemented, respectively, as (memoryless) switching networks and banks of RAM.

[1]Because of the mathematical formalism used later, we view circuits with inputs coming from the right.

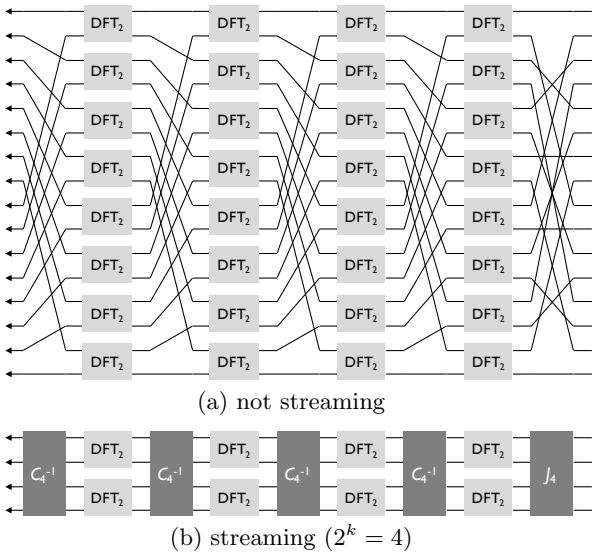


*FPGA'16, February 21 - 23, 2016, Monterey, CA, USA*



ISBN 978-1-4503-3856-1/16/02...$15.00

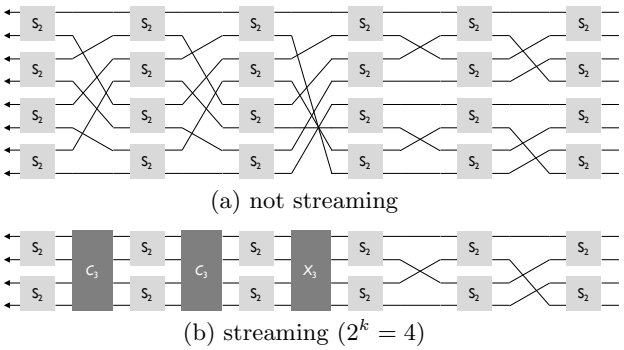
DOI: http://dx.doi.org/10.1145/2847263.2847277




(a) not streaming

(b) streaming ($2^k = 4$)

**Figure 2: On the top, dataflow of a Pease FFT on $2^4$ elements. After a bit-reversal permutation, a set of 8 parallel DFTs on 2 elements followed by a stride permutation is repeated 4 times. This graph can be directly used for a direct fully-parallel implementation. On the bottom, the same implementation is "folded" with $k = 2$, allowing to reduce the use of DFTs to sets of 2 parallel units.**




(a) not streaming

(b) streaming ($2^k = 4$)

**Figure 3: A sorting network working on $2^3$ elements [6]. The "$S_2$" blocks represent two input sorters. On the top, a fully-parallel implementation. On the bottom, the same implementation "folded" with $k = 2$, allowing to halve the number of sorters [7].**

We show that this decomposition is equivalent to a matrix factorization problem in which the minimization of certain ranks of submatrices is equivalent to minimizing the logic of the resulting circuit.

- Finally, we demonstrate our method by generating streamed bit reversal permutations for a Virtex FPGA, and by comparing our optimal solutions to prior art.

## 2. BACKGROUND AND NOTATION

We provide background on linear permutations, starting with two special cases before we give a general definition.

**Bit-reversal permutation.** Used in FFTs, the bit-reversal permutation has been studied extensively [8]. It maps each element to the position given by reversing the binary representation of its index. Formally, we denote the binary representation of an index $i$ with a column vector $i_b$ of $n$ bits, such that the most significant bit is at the top. For example, if $n = 3$,

$$6_b = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix},$$

which the bit reversal maps by flipping upside down to obtain $3_b$. Formally, it maps positions as $i_b \mapsto i'_b = J_n \cdot i_b$, where

$$J_n = \begin{pmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{pmatrix}. \tag{1}$$

This $n \times n$ bit matrix describes how the bit reversal operates "on the bits," and should not be confused with the $2^n \times 2^n$ permutation matrix that encodes how it maps the data.

**Perfect shuffle.** The perfect shuffle on $2^n$ elements interleaves the first and the second half:

$$i \mapsto \begin{cases} 2i, & \text{if } 0 \le i < 2^{n-1}, \\ 2i - 2^n + 1, & \text{if } 2^{n-1} \le i < 2^n. \end{cases}$$

On the bit representation it can be represented as cyclic shift: $i_b \mapsto i'_b = C_n \cdot i_b$, where

$$C_n = \begin{pmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ 1 & & & \end{pmatrix} \tag{2}$$

is a bit matrix.

If $P$ is an $n \times n$ matrix that describes the way a permutation works on the binary representation of the elements, we denote this permutation with $\pi(P)$. Formally, $\pi(P)$ is the permutation of $\{0, \ldots, 2^n - 1\}$ such that, for all $i$ in this set,

$$(\pi(P)(i))_b = P \cdot i_b.$$

**General linear permutations.** Generalizing the previous special cases we consider an arbitrary invertible bit matrix[2] $P$. Then the mapping $i_b \mapsto P \cdot i_b$ defines a permutation on $\{0, \ldots, 2^n - 1\}$ that we denote with $\pi(P)$. We call such permutations linear [9, 10] and there are $\prod_{i=0}^{n-1}(2^n - 2^i)$ of them. In particular, not every permutation on $2^n$ elements is linear; for example, linear permutations always leave the first element unchanged (since $0_b$ is the all-zero vector and thus mapped to $P \cdot 0_b = 0_b$).

For instance, if

$$V_n = \begin{pmatrix} 1 & & \\ \vdots & \ddots & \\ 1 & & 1 \end{pmatrix},$$

then $\pi(V_3)$ is the permutation: $0 \mapsto 0$, $1 \mapsto 1$, $2 \mapsto 2$, $3 \mapsto 3$, $4 \mapsto 7 \mapsto 4$, $5 \mapsto 6 \mapsto 5$. More generally, $\pi(V_n)$ is the permutation of $2^n$ elements that leaves the first $2^{n-1}$

[2]Mathematically, $P \in \mathrm{GL}_n(\mathbb{F}_2)$, where $\mathbb{F}_2$ is the Galois field with two elements. Hence, the set of linear permutations is a group, i.e., closed under multiplication and inversion.

elements unchanged, and that reverses the list of the others. It occurs in fast cosine transforms [11].

**Composition of linear permutations.** Composing two linear permutations corresponds to multiplying the associated matrices:

$$\pi(P) \circ \pi(Q) = \pi(PQ).$$

Additionally, we have $\pi(I_n) = I_{2^n}$ and therefore[3]:

$$\pi(P^{-1}) = \pi(P)^{-1}.$$

As an example, every stride permutation on $2^n$ elements is a power $r$ of the perfect-shuffle $\pi(C_n)$. Therefore, these are linear permutations as well with the associated matrix $C_n^r$.

# 3. STREAMING LINEAR PERMUTATIONS (SLPS): THEORY

Based on the prior formalism, we introduce the problem of streaming linear permutations (as in Fig 1(b)) using bit matrices. Then we discuss two special cases: temporal permutations that do not permute across ports and thus can be implemented using banks of RAM only, and spatial permutations that only permute elements within each cycle and thus can be implemented using switching networks (SNWs). Our approach is then to decompose the general case into these special cases, for which implementations can readily be derived.

Finally, we prove a lower bound on the switching complexity of a given permutation. This bound will later turn out to be sharp and is one main contribution of this paper.

**Matrix formalism.** As in the introduction, we index each element from 0 to $2^n - 1$ such that for $2^k$ ports, the element with index $i = c \cdot 2^k + p$ enters during the $c^{th}$ cycle on the $p^{th}$ input port. This means $c_b$ are exactly the upper $t = n - k$ bits of $i_b$ and $p_b$ are the lower $k$ bits. For instance, for $t = 3$ and $k = 2$, the element with the index

$$22_b = \left(\begin{array}{c}1\\0\\1\\\hline 1\\0\end{array}\right) = \left(\begin{array}{c}5_b\\\hline 2_b\end{array}\right)$$

will arrive during the 5th cycle on port 2.

Therefore, it is natural to block a given bit matrix $P$ as

$$P = \begin{pmatrix} P_4 & P_3 \\ P_2 & P_1 \end{pmatrix}, \quad \text{such that } P_4 \text{ is } t \times t. \tag{3}$$

Hence, the associated streaming permutation maps the input element that arrives on port $p$ during cycle $c$ to the output port $P_1 p_b + P_2 c_b$ at cycle $P_4 c_b + P_3 p_b$.

Next we introduce two special cases of SLPs that will form the building blocks of our general solution.

**Spatial permutations.** We define (memoryless) spatial permutations as SLPs that permute only within cycles. Therefore, $P$ must leave the upper $t$ bits $c_b$ of each address unchanged, i.e., satisfy $P_4 c_b + P_3 p_b = c_b$, which yields the form

$$P = \begin{pmatrix} I_t & \\ P_2 & P_1 \end{pmatrix}. \tag{4}$$

[3] $\pi$ is a group-homomorphism.

These can be implemented using a switching network that consists of controlled $2 \times 2$-switches (see Fig. 4 later). The cycle number controls the setting of the switches. The implementation using a shortened Omega network will be discussed in Section 4.1.

If, in addition, the same reordering is performed in each cycle, we call the spatial permutation steady. This is the case if and only if $P_2 = 0$. Such permutations can be implemented with a simple rewiring without control (similar to Fig. 1(a)), and we consider its cost to be zero.

**Temporal permutations.** These are the dual of spatial permutations, in the sense that they leave the port number unchanged but permute across cycles. Hence, these permutations are represented by matrices of the form

$$P = \begin{pmatrix} P_4 & P_3 \\ & I_k \end{pmatrix}. \tag{5}$$

They can be implemented using $2^k$ banks of RAM as explained in Section 4.2.

**General linear permutations: Switching complexity.** We implement general linear permutations $\pi(P)$ by first decomposing them into temporal and spatial permutations, i.e., by factoring $P$ (blocked as in (3)) into matrices of the form (4) and (5). We will later see that three such matrices always suffice. Interestingly, with this assumption on the building blocks we can already prove a lower bound on the number of switches needed. The reason is that only the switches can map between ports, and their number is thus determined by "how much variety" in mapping between ports is required across the different cycles.

THEOREM 1. *A full-throughput implementation of an SLP for $P$ with $2^k$ ports that only uses $2 \times 2$-switches for routing requires at least $\operatorname{rk} P_2 \cdot 2^{k-1}$ many switches, where $\operatorname{rk} P_2$ denotes the rank of the matrix $P_2$.*

PROOF. As the implementation has full throughput, each element passes at most one time through a given switch. We denote with $\ell_{p,c}$ the number of switches that the element that arrives on port $p$ at cycle $c$ passes through.

If we accumulate across cycles for all inputs at port $p$, the bit representations of the corresponding output ports, we get

$$\{P_1 p_b + P_2 c_b \mid 0 \le c < 2^t\} = P_1 p_b + \operatorname{im} P_2.$$

This set (as a coset of direction $\operatorname{im} P_2$) contains $2^{\operatorname{rk} P_2}$ elements. This means that each input port has to communicate with $2^{\operatorname{rk} P_2}$ different output ports.

Let now $p'$ be one of the $2^{\operatorname{rk} P_2}$ possible output ports for an element from input port $p$. Further, let $\bar{c}$ be an input cycle of an arbitrary element which transits from $p$ to $p'$. The set of cycles for which an element transits from $p$ to $p'$ is:

$$\{c_b \mid p'_b = P_1 p_b + P_2 c_b\} = \bar{c}_b + \ker P_2.$$

This set (as a coset of direction $\ker P_2$) contains $2^{t - \operatorname{rk} P_2}$ elements. As this number is independent from $p'$, the distribution over the possible output ports is uniform. Therefore, elements that arrive on port $p$ must at least go through $\operatorname{rk} P_2$ switches in average (since $\log_2(2^{\operatorname{rk} P_2}) = \operatorname{rk} P_2$ bits are needed to describe the output port):

$$\frac{1}{2^t} \sum_{c=0}^{2^t - 1} \ell_{p,c} \ge \operatorname{rk} P_2, \quad \text{for every } p. \tag{6}$$

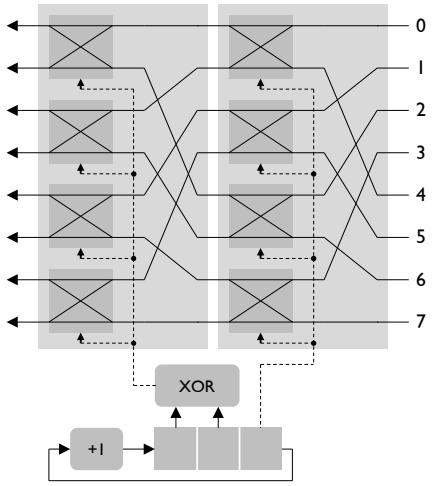



**Figure 4: An SNW consisting of two Omega network stages. Each stage contains a perfect shuffle followed by a column of $2^{k-1}$ switches controlled by a single common bit. Here, the first stage is controlled by a single bit of a counter, while the second one is controlled by the sum of the two other bits of this counter.**

We now denote with $s$ the number of switches in an implementation. Since each switch has two inputs, two elements per cycle pass through it. In total, $2 \cdot 2^t$ elements pass through a single switch. Hence

$$\sum_{\substack{0 \le c < 2^t \\ 0 \le p < 2^k}} \ell_{p,c} \le 2 \cdot s \cdot 2^t. \tag{7}$$

Combining (6) and (7), we get:

$$s \ge \frac{1}{2^{t+1}} \sum_{p=0}^{2^k-1} \sum_{c=0}^{2^t-1} \ell_{p,c} \ge \frac{1}{2} \sum_{p=0}^{2^k-1} \operatorname{rk} P_2,$$

which yields the desired result. □

As examples, we see that the number of switches for a spatial permutation is at least $\operatorname{rk} P_2 \cdot 2^{k-1}$, whereas for a temporal permutation that lower bound is 0, as expected, since no switches are needed.

# 4. IMPLEMENTATION OF SPATIAL AND TEMPORAL PERMUTATIONS

In this section, we explain how to implement the two special cases of SLPs. In the next section we solve the general case by optimally decomposing it into these.

## 4.1 Spatial Permutations

We show how to optimally implement a given spatial permutation using a switching network (SNW) with $\operatorname{rk} P_2 \cdot 2^{k-1}$ 2×2-switches, thus matching the lower bound of Theorem 1. The network we construct is an Omega network [10] with $k - \operatorname{rk} P_2$ stages removed. An optimal solution is already given in [2]; our description here is somewhat simpler and included for completeness.

A stage of an Omega network consists of a perfect shuffle followed by a column of $2^{k-1}$ 2×2-switches: see Fig. 4, which shows 2 stages. We first consider one column of switches. If these switches are all controlled by a common bit, then, when this bit is set, pairs of elements are exchanged:

$$\begin{cases} p \mapsto p+1 & \text{if } p \text{ is even} \\ p \mapsto p-1 & \text{if } p \text{ is odd,} \end{cases} \tag{8}$$

otherwise the column of switches leaves the data unchanged.

We add a counter $c$ of $t$ bits that is incremented at every cycle. Then, for a fixed vector $v$ of $t$ bits, it is possible to compute $c_b \cdot v$ using xor gates, and we use the result to control the column of switches. This structure performs the permutation (8) when $c_b \cdot v = 1$, and does nothing otherwise. In other words, we have implemented $\pi(K_v)$, where

$$K_v = \begin{pmatrix} I_t & & & \\ & 1 & & \\ & & \ddots & \\ v^T & & & 1 \end{pmatrix}.$$

The perfect shuffle that precedes within the stage is a steady spatial permutation, i.e., a rewiring. Therefore, with our formalism, one stage in Fig. 4 is described by the matrix:

$$S_v = K_v \cdot \begin{pmatrix} I_t & \\ & C_k \end{pmatrix}.$$

We now construct an implementation for a spatial permutation given by (4). First, we find an invertible $k \times k$-matrix $L$ such that $LP_2$ has $\operatorname{rk} P_2$ non-zero lines $v_i^T$ at the top (Gauss elimination):

$$LP_2 = \begin{pmatrix} v_1^T \\ \vdots \\ v_{\operatorname{rk} P_2}^T \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Direct computation shows that:

$$P = \begin{pmatrix} I_t & \\ & L^{-1} C_k^{k-\operatorname{rk} P_2} \end{pmatrix} S_{v_{\operatorname{rk} P_2}} \dots S_{v_1} \begin{pmatrix} I_t & \\ & LP_1 \end{pmatrix}.$$

This yields an implementation with $\operatorname{rk} P_2$ Omega network stages framed by two rewirings. Thus, the number of switches used is $\operatorname{rk} P_2 \cdot 2^{k-1}$.

Finally, 2×2-switches can easily be implemented using two 2-to-1 multiplexers. However, some platforms may support larger multiplexers more efficiently. In this case, it is possible to group several switches of different stages as shown in Fig. 5 with an example.

## 4.2 Temporal Permutations

We consider a temporal permutation associated with a matrix (5), and implement it using $2^k$ RAM banks, each capable of storing $2^t$ elements.

**Implementation principle.** Each port is associated with one bank: the input port $p$ is connected to the write port of the $p^{th}$ bank, and the read port of this bank is connected to the corresponding output port. A possible scheme consists in writing incoming elements linearly in the bank (using a counter $c$ of $t$ bits, as in the spatial permutation case), and to retrieve them in the permuted order, i.e. at

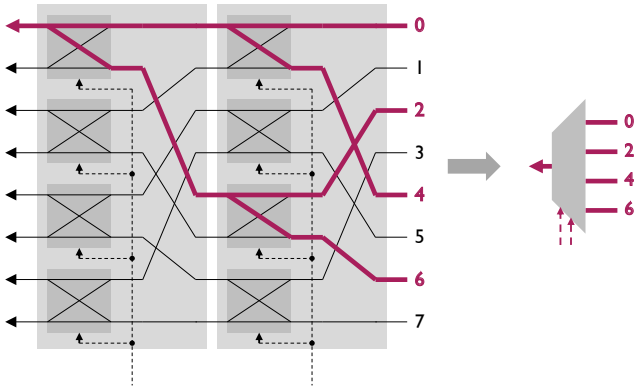



**Figure 5: Implementation of the first output port of a switching network using a 4-to-1 multiplexer.**

the address $P_4^{-1}c_b + P_4^{-1}P_3p_b$. This address can be computed jointly for every banks using xor gates on the bits of $c$. Then, inverters specialize these addresses for each bank by adding $P_4^{-1}P_3p_b$.

However, depending on the permutation, this scheme may not be suitable for full-throughput, as some elements of a dataset may be written to a memory address that contains an element of the previous dataset that has not been retrieved yet. Depending on the technology available for the memory, different strategies can be used to overcome these conflicts.

**Single-ported RAM.** In the case where it is only possible to write or to read an element during a cycle, [4] proposes a double-buffering method. Each port is associated with two RAM banks. One set is written in one of them, while elements of the previous set are retrieved from the second one. This method doubles the memory consumption, and requires an additional multiplexer per port, but has little overhead in control complexity.

If the RAM allows a simultaneous read and write at the same address, [5] proposes a method that uses only one bank per port to perform a temporal permutation $\sigma$. Each incoming element is written at the address where the element of the previous set is being read. For example, if the first set is written linearly in the memory, then the second set is written where the first set is read, i.e. at address $\sigma^{-1}(c)$. The $i^{th}$ set is then read at address $\sigma^{-i}(c)$.

In the case of linear permutations, this address becomes:

$$P_4^{-i}c_b + (P_4^{-i} + \cdots + P_4^{-1})P_3p_b. \tag{9}$$

This method is well suited in the case where $P_4$ is the identity, equal to its inverse, or more generally, if $(P_4^i)_i$ has a low period. In this case, all possible addresses can be computed using xor gates, and a counter $i$ suffices to control a multiplexer choosing the appropriate address. Otherwise, it becomes interesting to store the different values of $P_4^{-i}$ and of $(P_4^{-i} + \cdots + P_4^{-1})P_3$ in a ROM. In the worst case, this ROM would contain $(k+n)\cdot t \cdot 2^t$ bits[4]. The address (9) can then be computed using and and xor gates.

**Dual-ported RAM.** If the RAM used allows two simultaneous read and write at two different addresses, it is possible to absorb a potential array of 2×2-switches that would

[4]The period of $(P_4^{-i}+\cdots+P_4^{-1})_i$ is at most twice the period of $(P_4^i)_i$, which is itself at most $2^t$ [12].

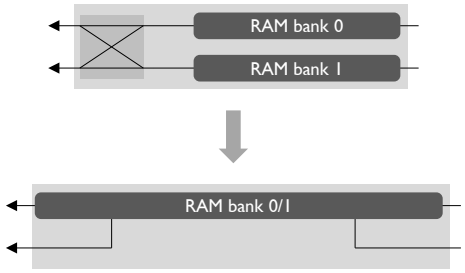



**Figure 6: Merging two banks with a $2\times2$-switch in a large dual-ported bank.**

follow the temporal permutation. Two banks connected to the same switch are fused into one large bank (see Fig. 6), and the read/write addresses corresponding to the two ports are swapped according to the control bit of the switch.

**Reuse.** If $0 < r \leq t$, and $P$ has the form:

$$\begin{pmatrix} I_r & & \\ & * & * \\ & & I_k \end{pmatrix},$$

it means that the associated temporal permutation is periodic with a period of $2^{t-r}$ cycles[5]. Therefore, it is possible to divide the memory consumption by $2^r$ by implementing only the permutation represented by the lower principal submatrix, and reuse it $2^r$ times[6].

## 5. GENERAL LINEAR PERMUTATIONS

In this section, we discuss the implementation of a general SLP $\pi(P)$ using the previous structures. This is equivalent to decomposing $P$ into spatial and temporal permutations, i.e., permutations of the form (4) and (5).

A first idea is to use one spatial and one temporal permutation. Indeed, if the block $P_4$ is invertible, Gauss elimination yields

$$P = \begin{pmatrix} I_t & \\ P_2P_4^{-1} & P_1 + P_2P_4^{-1}P_3 \end{pmatrix} \begin{pmatrix} P_4 & P_3 \\ & I_k \end{pmatrix}.$$

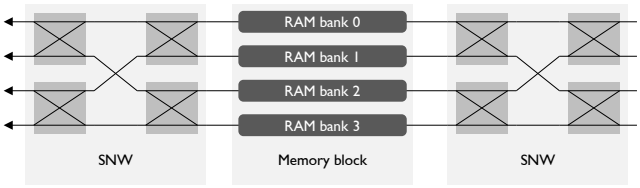
This means that $\pi(P)$ can be implemented using a memory block followed by an SNW. For the spatial part, $\operatorname{rk} P_2P_4^{-1} = \operatorname{rk} P_2$, i.e., our implementation will have $\operatorname{rk} P_2 \cdot 2^{k-1}$ switches, which matches the lower bound of Theorem 1.

Conversely, it is possible to decompose an SLP using an SNW followed by a memory block, if $P_1$ is invertible. Again, the construction will be optimal.

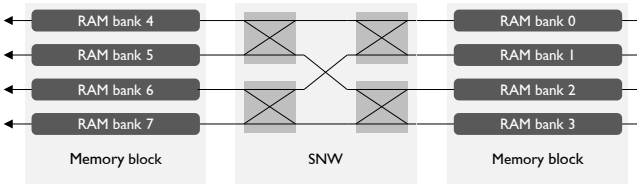
However, if neither $P_1$ nor $P_4$ are invertible, none of the solutions above exist. Hence, three blocks are needed and two possibilities exist, depicted in Fig. 7: the SNW-RAM-SNW structure (Section 5.1), and the RAM-SNW-RAM structure (Section 5.2).

[5]This is a consequence of $\pi(I_r \oplus A) = I_{2^r} \otimes \pi(A)$ in the notation of [2].

[6]This optimization has the theoretical advantage of yielding an empty implementation for the trivial temporal permutation $\pi(I_n)$.




(a) SNW-RAM-SNW




(b) RAM-SNW-RAM

**Figure 7: Two possible architectures for a streaming permutation.**

## 5.1 SNW-RAM-SNW

An SNW-RAM-SNW implementation (Fig. 7(a)) corresponds to the factorization

$$P = \begin{pmatrix} I_t & \\ L_2 & L_1 \end{pmatrix} \begin{pmatrix} M_4 & M_3 \\ & I_k \end{pmatrix} \begin{pmatrix} I_t & \\ R_2 & R_1 \end{pmatrix}. \tag{10}$$

Using our method of implementation, the number of switches involved equals $(\operatorname{rk} L_2 + \operatorname{rk} R_2)2^{k-1}$. Thus we want to minimize $\operatorname{rk} L_2 + \operatorname{rk} R_2$ for an optimal implementation. This decomposition has been studied in [13], summarized in the following theorem:

THEOREM 2. *If $P$ is an invertible $n \times n$ matrix, then* (10) *verifies:*

$$\operatorname{rk} L_2 + \operatorname{rk} R_2 \geq \max(\operatorname{rk} P_2, n - \operatorname{rk} P_4 - \operatorname{rk} P_1).$$

*Further, there exists a decomposition* (10) *reaching this bound.*

This theorem provides the minimal number of switches possible for the assumed architecture SNW-RAM-SNW, along with the existence of a solution reaching this bound. An algorithm to compute this solution in cubic arithmetic time in $n$ is provided in [13][7].

However, if $\operatorname{rk} P_4 + \operatorname{rk} P_2 + \operatorname{rk} P_1 < n$, the solution has more switches than suggested by Theorem 1 (which does not fix the architecture). It turns out that in this case the next architecture is optimal in terms of the number of switches, at the price of twice the RAM.

## 5.2 RAM-SNW-RAM

A RAM-SNW-RAM implementation (Fig. 7(b)) corresponds to the factorization

$$P = \begin{pmatrix} L_4 & L_3 \\ & I_k \end{pmatrix} \begin{pmatrix} I_t & \\ M_2 & M_1 \end{pmatrix} \begin{pmatrix} R_4 & R_3 \\ & I_k \end{pmatrix}. \tag{11}$$

[7]The "rank exchange" section in [13] can be used in some cases to balance the ranks of $L_2$ and $R_2$. For instance, if $\operatorname{rk} L_2$ and $\operatorname{rk} R_2$ are both odd, it is interesting to reduce the rank of $L_2$ by one and increase the rank of $R_2$ by one, thus making them both even, and therefore easier to implement using 4-input multiplexers.

A switching-optimal solution is guaranteed by the following theorem:

THEOREM 3. *If $P$ is an invertible $n \times n$ matrix, there exists a decomposition* (11) *that verifies* $\operatorname{rk} M_2 = \operatorname{rk} P_2$.

The existence of such a decomposition is again shown in [13], with an algorithm that computes such a decomposition in cubic arithmetic time in $n$.

In summary, the RAM-SNW-RAM solution is always optimal in terms of the number of switches. However, if $\operatorname{rk} P_4 + \operatorname{rk} P_2 + \operatorname{rk} P_1 \geq n$, SNW-RAM-SNW offers a better solution with half the RAM.

# 6. RESULTS

We evaluate our method in two ways. First, we consider one particular, but important example: the streamed bit reversal. We compare our two proposed architectures (one of which is optimal) against a prior solution. Second, we compare our streamed permutations against all four prior solutions that we found in the literature. We show a table summarizing the similarities and differences and illustrate these with three example settings.

**Example: Bit-reversal.** We consider for $k = t = n/2$ the bit-reversal permutation $\pi(J_n)$. Since $P_2 = J_k$, Theorem 1 states that at least $k \cdot 2^{k-1}$ switches are needed. However, Theorem 2 shows that an SNW-RAM-SNW structure requires twice this amount: $k \cdot 2^k$ switches, based on, for example,

$$P = J_n = \begin{pmatrix} I_k & \\ J_k & I_k \end{pmatrix} \begin{pmatrix} I_k & J_k \\ & I_k \end{pmatrix} \begin{pmatrix} I_k & \\ J_k & I_k \end{pmatrix}.$$

If, on the other hand, we choose a RAM-SNW-RAM structure, we can reach the minimal number of switches with, for example,

$$P = J_n = \begin{pmatrix} I_k & J_k \\ & I_k \end{pmatrix} \begin{pmatrix} I_k & \\ J_k & I_k \end{pmatrix} \begin{pmatrix} I_k & J_k \\ & I_k \end{pmatrix}.$$

The price is twice the RAM capacity. Note in both cases the simplicity of the control logic: only a $k$-bit counter and $k$ inverters are needed.

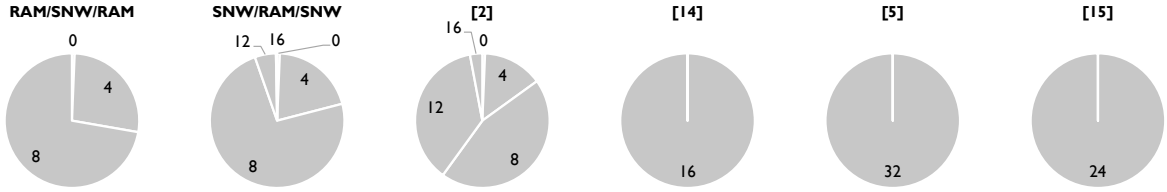
Fig. 8 shows throughput versus area for a bit reversal on $2^{11}$ 16-bit elements for the two different architectures implemented with $k \in \{1, \ldots, 5\}$, i.e., 2 to 32 ports, and $t = 11-k$. In this case, our SNW-RAM-SNW solution is equal to the one proposed by [2]. For each of the two solutions we also implemented the FPGA-specific optimization that uses 4-input multiplexers as sketched in Fig. 5, which yields significant area gains.

We compare against the RAM-SNW-RAM solution in [14], which is more general in that it can handle (fixed) arbitrary, also non-linear permutations. The target is a Virtex-7 xc7vx1140tflgl1930 FPGA, using Xilinx Vivado 2014.4.

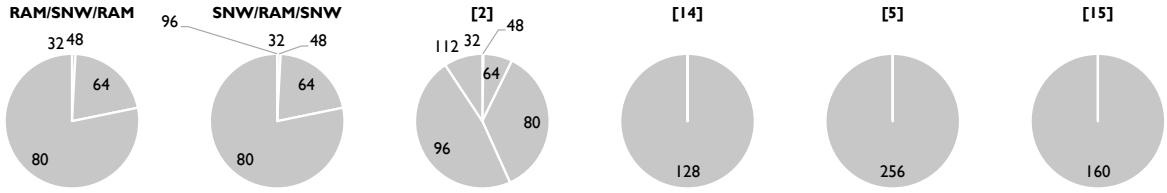
**Comparison against prior work.** Table 1[8] summarizes the similarities and differences between our solutions (SNW-RAM-SNW and RAM-SNW-RAM) and four prior works. As the table shows, only ours provide guaranteed optimal switching complexity at similar RAM cost.

To show the difference with an example, Fig. 9 compares, for different streaming scenarios, the number of switches

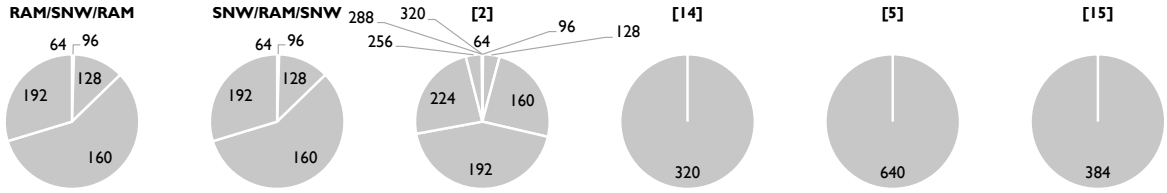
[8]We suppose here that [5] uses a switch based Beneš permutation network to implement their crossbars.




(a) Permutations of $2^n = 32$ elements with $2^k = 8$ ports




(b) Permutations of $2^n = 4096$ elements with $2^k = 32$ ports




(c) Permutations of $2^n = 4096$ elements with $2^k = 64$ ports

**Figure 9: Number of switches needed for $10^7$ random SLPs with different architectures.**

| Architecture | Permutations | Memory | Number of switches | Optimal routing? |
|---|---|---|---|---|
| RAM/SNW/RAM | Linear only | $2^{k+1}$ banks of $2^t$ words | $\operatorname{rk} P_2 \cdot 2^{k-1}$ | Always |
| SNW/RAM/SNW | Linear only | $2^k$ banks of $2^t$ words | $\max(\operatorname{rk} P_2, n - \operatorname{rk} P_4 - \operatorname{rk} P_1) \cdot 2^{k-1}$ | Iff $\operatorname{rk} P_4 + \operatorname{rk} P_2 + \operatorname{rk} P_1 \geq n$ |
| [2] | Linear only | $2^k$ banks of $2^{t+1}$ words | $\geq \max(\operatorname{rk} P_2, n - \operatorname{rk} P_4 - \operatorname{rk} P_1) \cdot 2^{k-1}$ | Generally not |
| [14] | All | $2^{k+1}$ banks of $2^{t+1}$ words | $(k - 1/2) \cdot 2^k$ | Never for SLPs with $k \geq 2$ |
| [5] | All | $2^k$ banks of $2^t$ words | $(k - 1/2) \cdot 2^{k+1}$ | Never for SLPs |
| [15] | All | $2^k$ banks of $2^t$ words | $k \cdot 2^k$ | Never for SLPs |

**Table 1: Comparison of different architectures using RAMs, in the case of a full-throughput SLP.**

used by the different architectures. In (a) all specified SLPs are considered, in (b) and (c), the full number is too large and we chose $10^7$ random samples instead. The pie charts show the distribution of the number of switches needed for these SLPs. As shown in the paper, one of our solutions (the two leftmost in the table) always minimizes the number of switches needed. We observe the improvement over prior work and also that for larger scenarios, most of the permutations can be implemented optimally using SNW-RAM-SNW. As we have seen, this is not true for the bit-reversal.

# 7. RELATED WORK

**Switching networks for sets of permutations.** Switching networks that can execute all permutations (in a non-streamed way) are a classic topic in computer science [16, 17]. A variant of this problem occurred in Section 4.1 where we implemented streamed spatial permutations. Namely, we had to build a minimal switching network capable of passing a subset of permutations.[9] Our solution was based on a reduced Omega network and we proved optimality. The complete Omega network has been heavily studied in [9, 10, 18, 19]. Beyond that, the problem of finding a minimal switching network to perform a given set of permutations appears to have not received much attention in the literature. An exception is the last section in [18], which, however, produces only upper bounds for few cases.

**SNW-RAM-SNW structure.** We now restrict ourselves to the structure proposed in Section 5.1. This architecture has already been proposed for streamed linear permutations in [2], which also proves optimality for the special case of permutations that permute the bits of the indexes (a group called PIPID in [10] or BP class in [19]), i.e., where $P$ has only one 1 in each row and column. In particular, this includes stride permutations (2) and bit-reversal (1). For these permutations, our solution is equal (Fig. 8 shows one example).

However, [2] has two shortcomings that we resolve in this paper. First, the method to derive an SNW-RAM-SNW implementation is in general not optimal (see Fig. 9). Second, [2] does not consider the alternative architecture RAM-SNW-RAM, which, in some cases provides solutions with fewer switches at the cost of twice the RAM. In this paper we resolve both problems completely by establishing an

[9]Specifically a coset $Hg$, where $g$ is a linear permutation, and $H$ a subgroup of bit complement permutations, i.e., permutations that map an index $i$ to $i_b + v$, where $v$ is a given bit vector.

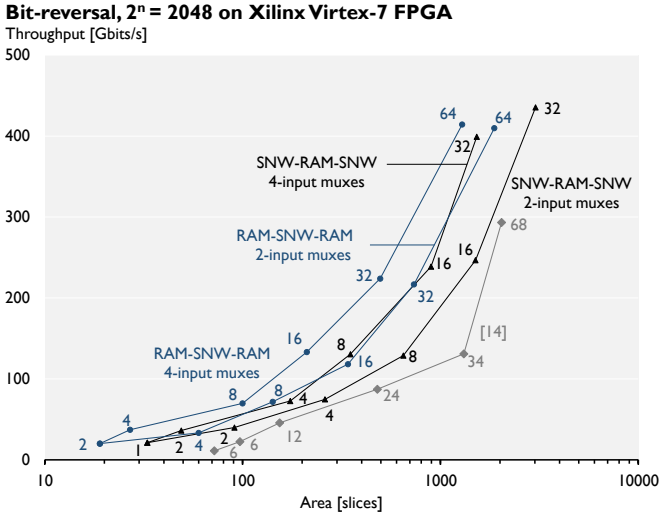



**Figure 8: Comparison of our two structures for a bit-reversal permutation on 2048 16-bit elements for different multiplexer sizes vs. [14]. Labels: number of BRAM tiles. In this example, the SNW-RAM-SNW structure that uses 2-input muxes is equivalent to [2].**

architecture-oblivious sharp lower bound for the number of switches needed and a technique for obtaining that optimal solution using the SNW-RAM-SNW or RAM-SNW-RAM architecture. We precisely characterize the cases where the latter wins.

As a minor point, the solution in [2] uses a double-buffering method to achieve full-throughput (as they mention a memory requirement of $2^{n+1}$ words in the last section). We propose an alternative method in Section 4.2 that does not require additional RAM capacity.

This SNW-RAM-SNW architecture has also been used in [5] to implement the streaming permutations needed in a bitonic sorting network (which are all linear). They achieve an efficient memory usage, but the method used (folding a Clos permutation network) doesn't harness the specificity of the particular permutations they consider, and the resulting design requires two complete switching networks (that allow any permutation), which also makes the control logic much more complex.

Similarly, [15] offers a solution based on a Beneš network to build a streamed solution for any, also non-linear, given permutation on $2^n$ elements. Because it is more general, it is not optimal for the linear case. Additionally, the generated datapath is independent of the desired permutation. The control logic is also more complex, as it uses ROM look-up tables to store memory addresses and the control bit of every switches for every cycles. This allows flexibility in the sense that different permutations can be implemented simply by modifying these tables, but is clearly suboptimal for a single fixed permutation. In Fig. 9, we showed how our solutions outperform this method.

**RAM-SNW-RAM structure.** The RAM-SNW-RAM structure was considered in [14] to implement any (including non-linear) streaming permutation of any size. A shortcoming is that the central SNW has to be able to pass any spatial permutation. Further, it considers only double-buffering for its temporal permutations. We compared our different architectures in Fig. 8 and 9.

**Other architectures for streamed permutations.** Other approaches for building a fixed permutation technique include [1], which proposes a register based implementation, and [20], which is specific to implementing stride permutations. These two methods have in common that they use registers to delay elements. In this paper we choose a more regular architecture using RAM banks instead, which are available on FPGAs, to spare logic.

## Acknowledgement

We thank Peter A. Milder for his help with implementing [14], and the anonymous reviewer who suggested to use 4-input multiplexers on FPGAs, which we incorporated in our results (Fig. 8).

# 8. CONCLUSIONS

The main theoretical result of this paper is the exact switching complexity of streamed linear permutations. We established this result by first proving a lower bound, and then providing a constructive method that achieves this lower bound. Our method implements optimal SLPs using switches and RAMs using two different architectures. One always has optimal switching complexity, but requires a RAM capacity of twice the size of the dataset. The other proposed architecture is switching-optimal for some permutations (that we precisely characterized) and requires only half the RAM capacity. We have implemented the technique to test on given permutations; but the main contribution of the paper is the theory and the underlying key idea: to phrase the problem as a specific matrix factorization and apply techniques from linear algebra to construct solutions and prove their optimality.

# 9. REFERENCES

[1] K. K. Parhi, "Systematic synthesis of DSP data format converters using life-time analysis and forward-backward register allocation," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, no. 7, pp. 423–440, 1992.

[2] M. Püschel, P. A. Milder, and J. C. Hoe, "Permuting streaming data using RAMs," *Journal of the ACM*, vol. 56, no. 2, pp. 10:1–10:34, 2009.

[3] M. Püschel, P. A. Milder, and J. C. Hoe, "System and method for designing architecture for specified permutation and datapath circuits for permutation," 2008. US Patent 8,321,823.

[4] P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel, "Computer generation of hardware for linear digital signal processing transforms," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 17, no. 2, 2012.

[5] R. Chen, S. Siriyal, and V. Prasanna, "Energy and memory efficient mapping of bitonic sorting on FPGA," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, pp. 240–249, 2015.

[6] D. E. Knuth, *The Art of Computer Programming, 2Nd Ed. (Addison-Wesley Series in Computer Science and*

*Information*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 1978.

[7] M. Zuluaga, P. A. Milder, and M. Püschel, "Streaming sorting networks," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2016. Accepted for publication.

[8] A. H. Karp, "Bit reversal on uniprocessors," *SIAM Review*, vol. 38, pp. 1–26, Mar. 1996.

[9] M. C. Pease, "The indirect binary n-cube microprocessor array," *IEEE Transactions on Computers*, vol. 26, no. 5, pp. 458–473, 1977.

[10] J. Lenfant and S. Tahé, "Permuting data with the Omega network," *Acta Informatica*, vol. 21, no. 6, pp. 629–641, 1985.

[11] G. Steidl and M. Tasche, "A polynomial approach to fast algorithms for discrete Fourier-cosine and Fourier-sine transforms," *Mathematics of Computation*, vol. 56, no. 193, pp. 281–296, 1991.

[12] M. Darafsheh, "The maximum element order in the groups related to the linear groups which is a multiple of the defining characteristic," *Finite Fields and Their Applications*, vol. 14, no. 4, pp. 992 – 1001, 2008.

[13] F. Serre and M. Püschel, "A lower-upper-lower block triangular decomposition with minimal off-diagonal ranks," *ArXiv e-prints*, 2014. arXiv:1408.0994.

[14] P. A. Milder, J. C. Hoe, and M. Püschel, "Automatic generation of streaming datapaths for arbitrary fixed permutations," in *Design, Automation and Test in Europe (DATE)*, pp. 1118–1123, 2009.

[15] R. Chen and V. Prasanna, "Automatic generation of high throughput energy efficient streaming architectures for arbitrary fixed permutations," in *Field Programmable Logic and Applications (FPL)*, pp. 1–8, 2015.

[16] V. E. Beneš, *Mathematical Theory of Connecting Networks and Telephone Traffic*. Academic Press, 1965.

[17] A. Waksman, "A permutation network," *Journal of the ACM*, vol. 15, no. 1, pp. 159–163, 1968.

[18] D. Steinberg, "Invariant properties of the shuffle-exchange and a simplified cost-effective version of the Omega network," *IEEE Transactions on Computers*, vol. 32, no. 5, pp. 444–450, 1983.

[19] D. Nassimi and S. Sahni, "A self-routing Benes network and parallel permutation algorithms," *IEEE Transactions on Computers*, vol. 30, no. 5, pp. 332–340, 1981.

[20] T. Järvinen, P. Salmela, H. Sorokin, and J. Takala, "Stride permutation networks for array processors," in *International Conference on Application-Specific Systems, Architectures and Processors Proceedings (ASAP)*, pp. 376–386, 2004.