Energy-Efficient Architecture for Stride Permutation on Streaming Data

Ren Chen and Viktor K. Prasanna
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, USA 90089
Email: {renchen, prasanna}@usc.edu

***Abstract*—Stride permutation is widely used in various digital signal processing algorithms when implementing on FPGAs. Permuting a long data sequence through hardware wiring leads to high area consumption and routing complexity. A more scalable approach is to build a hardware structure to permute streamed data inputs. In this paper, we present an energy-efficient architecture to perform stride (power-of-two) permutation on streaming data. A three-stage structure, composed of two stages of interconnection networks and one stage of data buffers, is used as a baseline architecture. To improve the energy efficiency, we develop a data remapping technique to reduce the memory consumed by the data buffers. This technique can be used to reduce the required memory by 50% at the expense of small amount of extra logic. We also present a multiplexer-based cyclic shift interconnection network designed for low power purpose. Our proposed architecture is evaluated using two performance metrics: the composite Energy$\times$Area$\times$Time (EAT) metric and the energy efficiency defined as points (real number) per Joule. The experimental results show that the proposed data remapping technique reduces up to 40% dynamic power consumption compared with the baseline architecture. The proposed architecture shows a high energy efficiency of up to 75.3 giga points/Joule, and has an EAT ratio of 0.31 to 0.35 over the baseline architecture for various streaming width $w (2 \leq w \leq 32)$.**

I. Introduction

Stride permutation is widely used in various digital signal processing algorithms, such as the Fast Fourier Transform (FFT), the Walsh-Hadamard Transform and the Viterbi coding [1], [2]. Given a stride t, a stride permutation reorders an m-element data vector, such that data elements with an index distance of t are shifted into adjacent locations. We call this permutation as a stride-by-t permutation. A stride permutation can be simply implemented by a reordering of hardware wires if all data elements are available concurrently. However, for a long data sequence, this solution is not technically feasible due to the routing complexity.

An alternative approach is to build a hardware structure to perform permutation on streaming data. This hardware can be called as a permutation block. Data elements are fed into the permutation block in a streaming manner. For example, the m-element data vector to be permuted with a stride t can be divided into several w-element sub-vectors, which are fed into the permutation block over m/w consecutive cycles. w is called as the *streaming width* defined as number of parallel input/output points (real number) in each cycle. The permuted results are output in the same streaming manner. Through pipelining techniques, the permutation block can handle multiple data vectors consecutively. Thus in each cycle, this block takes w elements as inputs, produces w outputs and supports non-stop processing.

Such approach introduces intermediate data permutations and storage. Based on the type of data storage used, previous work on the architecture design of permutation block can be classified into memory-based designs and register-based designs. According to [3], [4], delay commutators are proposed to perform stride permutation in the traditional VLSI implementation for FFT. By using register-based commutators, those folded FFT architectures achieve high computational performance per unit area. In [5], the authors propose a register-based stride permutation network for array processor. The proposed network supports any stride of power-of-two and achieves a high area efficiency by approaching the lower bound in the number of registers. In [6], a memory-based stride permutation approach has been proposed; the permutation network is proved mathematically control-cost optimal using dual-ported memories. Those works mainly focus on minimizing the memory consumption and the permutation network complexity while maximizing the throughput. However, energy efficiency is not considered.

For long data sequences, as a considerable amount of storage is required, memory-based designs are more scalable solutions than register-based designs. In this paper, we propose an energy efficient architecture for stride permutation on streaming data using memories. A three-stage permutation block using two stages of interconnection networks and one stage of memory-based data buffers is used as a baseline architecture. Based on this baseline, we reduce the memory consumption using our proposed data remapping technique by a factor of two. We further employ a multiplexer-based cyclic shift interconnection network for high energy efficiency purpose. Specifically, our main contributions are the following:

1) An energy efficient architecture for stride permutation on streaming data (Section III).
2) A data remapping technique that reduces the memory consumption by 50% compared with the baseline archi-

This work has been funded by DARPA under grant number HR0011-12-2-0023.

tecture (Section III-C).

3) A low power implementation of multiplexer-based cyclic shift interconnection network (Section III-D).
4) Implementations which achieve 68 to 75 giga points/Joule on the target platform (Section IV-E).

The rest of the paper is organized as follows. Section II introduces the stride permutation on streaming data. Section III presents our proposed streaming permutation architecture and the data remapping technique. Section IV presents experimental results and analysis. Section V concludes the paper.

II. Stride permutation on streaming data

In this section, we introduce stride permutation on streaming data. Then we present a three-stage approach to implement stride-by-t permutation block, and discuss application examples that employ such an architecture.

A. Stride permutation

A stride permutation can be defined by matrix representation. Given an m-element data vector x and a stride t $(0 \leq t \leq m-1)$, and the data vector y is produced by the stride-by-t permutation over x, then

$$y = P_{m,t}x \tag{1}$$

where $P_{m,t}$ is called the permutation matrix. $P_{m,t}$ is an invertible $m \times m$ bit matrix such that

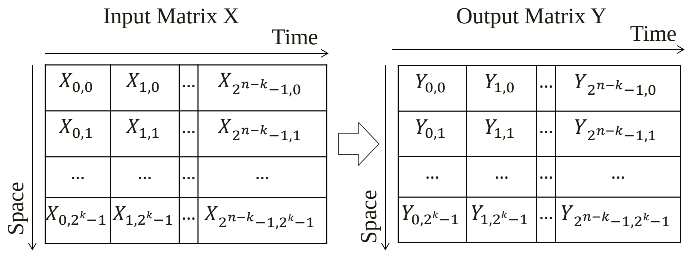
$$P_{m,t}[i][j] = \begin{cases} 1 & \text{if } j = (t \times i)\ mod\ m + (\lfloor t \times i/m \rfloor); \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

where mod is the modulus operation and $\lfloor \cdot \rfloor$ is the floor function. For example, a stride-by-2 permutation over a data vector of four data elements can be represented as

$$\begin{pmatrix} y_0 & y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \end{pmatrix}$$

Consider an application consisting of a stride permutation that processes a data vector of length 2^n. Suppose a permutation block with 2^k input/output ports is employed to perform the stride permutation. Accordingly, the *streaming width* w is 2^k. We can use a matrix $X(Y)$ of size $2^k \times 2^{n-k}$ to represent the streamed data vector $x(y)$. $x(y)$ is mapped to $X(Y)$ in column major order.

To illustrate the process of stride permutation on streaming data, we use Fig. 1 to show the data mapping from X (input) to Y (output) when $t = w = 2^k$. Indexes $i(0 \leq i \leq 2^{n-k}-1)$ and $j(0 \leq j \leq 2^k - 1)$ represent the temporal order and spatial order respectively. $X_{i,j}$ located in row j will be fed into the permutation block in cycle i. In each cycle, 2^k data elements enter the permutation block. During the permutation process, Y will be output over 2^{n-k} cycles after a certain amount of delay. Some data elements in X are reordered both in time and space. For example, data element $X_{1,0}$ is input in cycle 1 in row 0. Then it is permuted and produced as $Y_{0,1}$, which is output in cycle 0.


Input Matrix X
Time
Space
$X_{0,0}$ $X_{1,0}$ ... $X_{2^{n-k}-1,0}$
$X_{0,1}$ $X_{1,1}$ ... $X_{2^{n-k}-1,1}$
... ...
$X_{0,2^k-1}$ $X_{1,2^k-1}$ ... $X_{2^{n-k}-1,2^k-1}$
Output Matrix Y
Time
Space
$Y_{0,0}$ $Y_{1,0}$ ... $Y_{2^{n-k}-1,0}$
$Y_{0,1}$ $Y_{1,1}$ ... $Y_{2^{n-k}-1,1}$
... ...
$Y_{0,2^k-1}$ $Y_{1,2^k-1}$ ... $Y_{2^{n-k}-1,2^k-1}$

$$X_{i,j} = Y_{\lfloor l/2^r \rfloor, l(mod)2^r}, l = \Big((i2^k + j)mod(2^r)\Big) \cdot (2^{n-r}) + \left\lfloor \frac{(i \cdot 2^k + j)}{2^r} \right\rfloor$$

Fig. 1: Stride-by-2^k permutation on streaming data

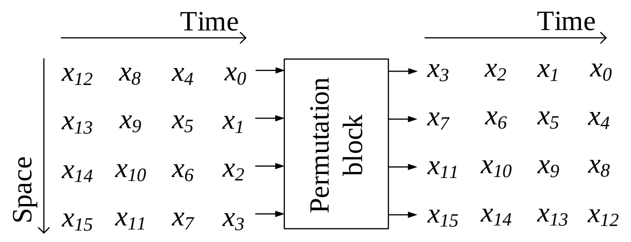

Time
Time
Space
x_{12} x_8 x_4 x_0
x_{13} x_9 x_5 x_1
x_{14} x_{10} x_6 x_2
x_{15} x_{11} x_7 x_3
Permutation block
x_3 x_2 x_1 x_0
x_7 x_6 x_5 x_4
x_{11} x_{10} x_9 x_8
x_{15} x_{14} x_{13} x_{12}

Fig. 2: Stride-by-4 permutation on 16 streamed data elements

Fig. 2 shows a stride-by-4 permutation on 16 data elements with a streaming width of 4. The data elements of vector x are mapped onto matrices X and Y to show how permutation works: the input matrix X is reflected over its principal diagonal. This process is equivalent to a 4×4 matrix transpose.

B. Three-stage permutation block

The stride permutation on streaming data is equivalent to a matrix rotate-sort problem. According to [7], a rotate-sort over arbitrary matrix X can be decomposed into order-row permutations and order-column permutations. Similarly, the stride permutation on streaming data can be decomposed into steps including permutation in time and permutation in space. To permute streaming data both in time and space, a three-stage permutation block similar with the design in [6], can be used as an approach for stride permutation on streaming data.

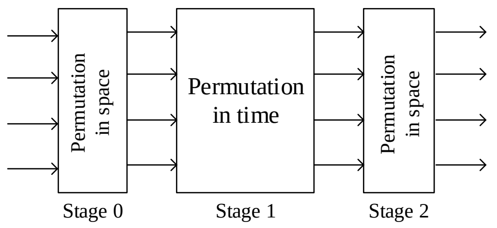
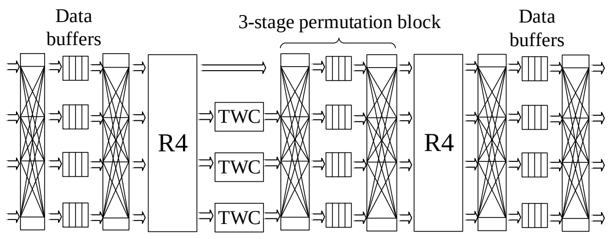


Fig. 3: Three-stage permutation block for stride permutation on streaming data

Fig. 3 shows the architecture overview of a three-stage permutation block. The basic idea is to design two interconnection networks to permute data vertically in space in stage 0 and 2. Several data buffers are employed to reorder data in time in stage 1. For a given long data vector, using RAMs is a more suitable approach for data reordering in time than using registers. In real applications, cyclic shift interconnect network is frequently employed for high resource efficiency by setting the streaming width to be the required stride. Benes interconnection network can be used for general cases [8]. In this paper, our design goal is to minimize the memory power consumption and maximize throughput denoted as the number of parallel outputs per cycle.

C. Application example

Stride permutation is frequently employed in various application implementations, such as Cooley-Tukey algorithm based FFT architectures. In [9], a parameterized FFT architecture is proposed to identify the design trade-offs in achieving energy efficiency. The parameterized architecture for 16-point FFT is shown in Fig. 4. This design is composed of cascaded butterfly computation units and data buffers. 3-stage permutation block is employed between the butterfly computation stages. During the processing of one FFT task, each butterfly unit is reused several times by multiple data streams consecutively. In this way, this compact FFT architecture achieves high computational performance per unit area.



R4: Radix-4 butterfly computation unit; TWC: Twiddle factor computation unit

Fig. 4: Radix-4 based 16-point FFT architecture

Power profile of this FFT architecture shown in Fig. 4 is identified in [10]. Fig. 5 shows the power profile of this streaming architecture for 16-bit fixed point FFT when implementing on state-of-the-art FPGA platform. Two major power components including communication power and computation power are evaluated by varying the FFT size. Communication power is consumed by the 3-stage permutation block. Computation power is consumed by the butterfly computation units. As shown in this figure, communication power is dominant in the entire dynamic power for various FFT sizes. This indicates that the design energy efficiency can be improved by optimizing the power consumption of 3-stage permutation block.

III. Architecture and implementation

In this section, we first introduce the baseline architecture. Then we present our proposed data remapping technique based on the baseline. We use a three-stage permutation block for stride permutation on streaming data. We assume stride is power-of-two. Cyclic shift interconnection network is employed in our design by setting streaming width to the required stride.

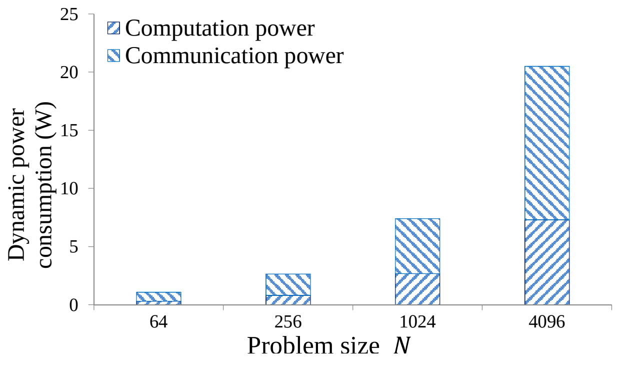


Fig. 5: Power profile of streaming architecture for N-point FFT

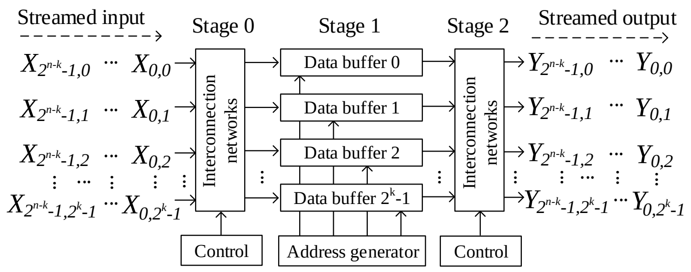


Fig. 6: Architecture overview

A. Baseline architecture

The baseline architecture is shown in Fig. 6. Two interconnection networks supporting cyclic shift are used. Data buffer access address are generated by the address generator. The permutation process can be decomposed into three phases: input phase in stage 0, buffering phase in stage 1, output phase in stage 2. The three permutation stages have been discussed in Section II-B.

Algorithm 1 shows the operation of the baseline architecture. This architecture supports processing multiple input data vectors successively. Each data vector is streamed and represented using matrix. To simplify the algorithm description, here we introduce how the permutation block works for two consecutive input data vectors.

During the input phase in stage 0, streamed $2^k \times 2^{n-k}$ matrices X and X' are fed into the interconnection network over 2^{n-k+1} cycles. As discussed in Section II-A, $X_{i,j}$ is input at ith cycle after X starts entering into the permutation block. In each cycle, 2^k data elements are permuted in space by the interconnection network. This phase is a preliminary step for buffering phase. Cyclic shift is performed in this phase so that, during the buffering phase, all the data elements can be reordered in time by 2^k data buffers over 2^{n-k+1} cycles without any memory conflict.

In the buffering phase in stage 1, 2^k data buffers are employed for data reordering in time. Each data buffer is responsible for permutation in time over 2^{n-k} data elements using 2^{n-k+1} cycles, hence has a size of 2^{n-k}. For each data buffer, data elements are written with incrementing memory addresses in successive cycles. Then the stored data elements are output with a temporally reordered sequence, and read with disconnected memory addresses. When processing multiple data vectors, the read and write operations are executed simultaneously on the same data buffer. As shown in this algorithm, in the cycle 2^{n-k} to $2^{n-k+1}-1$, data buffer i is accessed by both X' and X in different addresses. This particular access pattern requires the data buffer to be implemented with a dual-ported memory or two single-ported memories. When the streaming widht is 2^k, we have three considerations to choose 2^k data buffers. First, a wide memory may also be used for buffering purpose, however this solution is not scalable owing to large area consumption for wide streaming width. Second, in the worst case, 2^k different buffer read addresses are required in each cycle. Third, two or more data buffers may have the same access pattern in each cycle under some circumstance. For this special scenario, we can implement those data buffers using only one memory block. However, the required number of memory ports is still not reduced, thus the area and power won't be optimized.

In the output phase, the streamed $2^k \times 2^{n-k}$ matrices Y and Y' are produced over 2^{n-k+1} cycles. Cyclic shift is performed in each cycle so that the outgoing 2^k data elements are output with a correct order in the vertical space.

Algorithm 1 Operation of the baseline architecture

Input: Two $2^k \times 2^{n-k}$ matrices X and X'
Output: Two $2^k \times 2^{n-k}$ matrices Y and Y'
Parameters: Stride $t = 2^k$, Streaming width $w = 2^k$, Constant $s = 2^{n-k}$
Variables: k-bit Shift distance d_1, d_2, buffer write address $addr\_w$, buffer read address $addr\_r_i (0 \le i \le 2^k - 1)$, Cycle number m, data buffer index $i (0 \le i \le 2^k - 1)$, all addresses are represented using $(n-k)$ bits in binary

```
1: {Initialization}
2: $d_1 = 0, d_2 = 0, addr\_w = 0$
3: for $i = 0$ to $2^k - 1$ do
4:     $addr\_r_i = i$
5: end for
6: {During the input phase in stage 0}
7: for $m = 0$ to $2^{n-k} - 1$ do
8:     Do cyclic shift downward over $X_{m,0}, ..., X_{m,2^k-1}$
9:     Update cyclic shift distance: $d_1 = d_1 + 1$
10: end for
11: for $m = 2^{n-k}$ to $2^{n-k+1} - 1$ do
12:     Do cyclic shift downward over $X'_{m,0}, ..., X'_{m,2^k-1}$
13:     Update cyclic shift distance: $d_1 = d_1 + 1$
14: end for
15: {During the buffering phase in stage 1}
16: for $i = 0$ to $2^k - 1$, in parallel do
17:     for $m = 0$ to $2^{n-k} - 1$ do
18:         Write $X_{m,i}$ into data buffer $i$ with write address $addr\_w$
19:         Update buffer write address: $addr\_w = addr\_w + 1$
20:     end for
21:     for $m = 2^{n-k}$ to $2^{n-k+1} - 1$ do
22:         Write $X'_{m-s,i}$ into data buffer $i$ with write address $addr\_w$
23:         Update buffer write address: $addr\_w = addr\_w + 1$
24:         Read $X_{m-s,i}$ from data buffer $i$ with read address $addr\_r_i$
25:         Update buffer read address: $addr\_r_i = addr\_r_i - 1$ (cyclic)
26:     end for
27:     for $m = 2^{n-k+1}$ to $2^{n-k+2} - 1$ do
28:         Read $X'_{m-2s,i}$ from data buffer $i$ with read address $addr\_r_i$
29:         Update buffer read address: $addr\_r_i = addr\_r_i - 1$ (cyclic)
30:     end for
31: end for
32: {During the output phase in stage 2}
33: for $m = 2^{n-k}$ to $2^{n-k+1} - 1$ do
34:     Do cyclic shift upward over outgoing $X_{m-s,0}, ..., X_{m-s,2^k-1}$
35:     Update cyclic shift distance: $d_2 = d_2 + 1$
36: end for
37: for $m = 2^{n-k+1}$ to $2^{n-k+2} - 1$ do
38:     Do cyclic shift upward over outgoing $X'_{m-2s,0}, ..., X'_{m-2s,2^k-1}$
39:     Update cyclic shift distance: $d_2 = d_2 + 1$
40: end for
```

B. Energy efficient architecture using data remapping

In the baseline architecture, data buffers need to be implemented with dual-ported memories. In-order write and out-of-order read with different addresses are performed respectively on the dual ports. Hence the total memory consumption of each data buffer is 2^{n-k+1}. To reduce memory consumption, we develop a data remapping technique based on the baseline architecture. Accordingly, we update the algorithm for the buffering phase. The algorithms for input/output phase in the baseline are still used.

Algorithm 2 shows the updated algorithm used for the buffering phase. During the buffering phase, each data buffer is read and written simultaneously on the same memory location. Hence it requires the sing-ported memory block supporting simultaneous read-write operation on one memory location. Using the data remapping technique, each data element in the data buffers will be updated with a new value once the old data is output and consumed. In this way, the total memory consumed by the data buffers can be halved at the expense of extra logic resources for address generation.

C. Data remapping technique

We develop a data remapping technique to reduce the memory consumption for permutation in time, thus optimize the memory power. Permutation in time can also be represented using Eq. (1). Input data vector x is reordered in time and output data vector y is generated. Streaming data vectors are fed into the data buffer without interrupts. Continuously incoming data vectors are mapped onto the data buffer with different data layouts sequentially. Different access patterns on the data buffer are required by the read and write operations.

1) Data buffer access scheme: Fig. 7 shows the data buffer access process when $2^{n-k} = 4$. In the figure, several four-element data vectors enter into a data buffer one after another. Each data vector is permuted with the same temporally reordered sequence. In Fig. 7, $a_i, b_i, c_i, d_i (0 \le i \le 3)$ are successively permuted in time by the data buffer. A permutation of $[x_0, x_1, x_2, x_3] \rightarrow [x_0, x_3, x_2, x_1]$ is performed on each data vector. Fig. 7a shows the buffer access process without using data remapping. The data buffer consists of two parts of memories. The two memories are alternatively read and written during subsequent time periods. For the write part, data elements are input sequentially and each is written with the incrementing memory address. For the read part, data elements are output with a reordered sequence determined by the given memory address. For example, $b_i (0 \le i \le 3)$

Algorithm 2 Operation of the proposed architecture in the buffering phase

```
New input: $2^k \times 2^{n-k}$ matrix $X''$
New variables: Buffer access addresses $addr\_a_i$ and $addr\_b_i$ $(0 \le i \le 2^k - 1)$
1:  {Initialization}
2:  for $i = 0$ to $2^k - 1$ do
3:      $addr\_a_i = 0, addr\_b_i = i$
4:  end for
5:  {During the buffering phase in stage 1}
6:  for $i = 0$ to $2^k - 1$, in parallel do
7:      for $m = 2^{n-k}$ to $2^{n-k+1} - 1$ do
8:          Write $X'_{m-s,i}$ into data buffer $i$ with address $addr\_a_i$
9:          Read $X_{m-s,i}$ from data buffer $i$ with address $addr\_a_i$
10:         Update buffer read address: $addr\_a_i = addr\_a_i + 1$ (cyclic)
11:     end for
12:     for $m = 2^{n-k+1}$ to $2^{n-k+2} - 1$ do
13:         Write $X''_{m-2s,i}$ into data buffer $i$ with address $addr\_b_i$
14:         Read $X_{m-2s,i}$ from data buffer $i$ with address $addr\_b_i$
15:         Update buffer read address: $addr\_a_i = addr\_a_i + 1$ (cyclic)
16:     end for
17: end for
18: {End of the buffering phase}
```

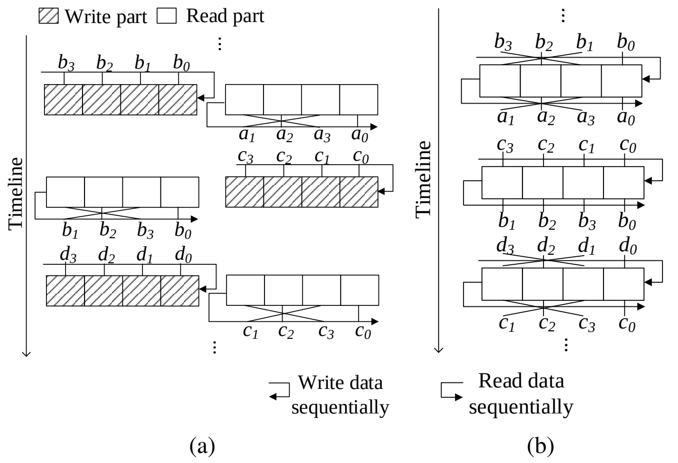
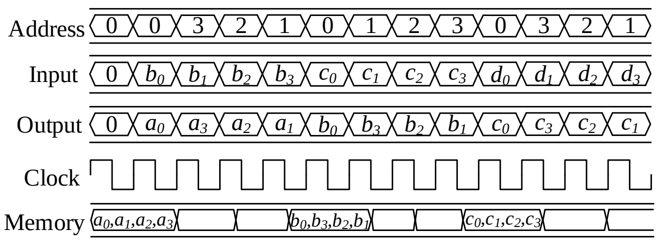


Fig. 7: Access scheme in one data buffer (a) without using data remapping, (b) with using data remapping.

enter into the data buffer sequentially and each is written with the incrementing memory address. Then the four data elements are output one after another and each is read with the cyclic decrementing address. Hence, different read and write addresses need to be provided in each cycle. Assuming each data vector has a size of 2^{n-k}, the memory consumption of each data buffer is 2^{n-k+1}.

Fig. 7b shows the buffer access process using data remapping. Read and write operations are performed simultaneously on the same memory location in each cycle. For each data vector, data elements are not only read out of order but also written out of order. Each data vector is provided an address vector for buffer read/write. As shown in Fig. 7b, two address vectors are used in turn. In each cycle, each memory entry is updated with a new value once the old value is output and consumed. Finally all the data vectors are reordered correctly in time. Fig. 8 shows the timing diagram of the data buffer



Address 0 0 3 2 1 0 1 2 3 0 3 2 1
Input 0 b_0 b_1 b_2 b_3 c_0 c_1 c_2 c_3 d_0 d_1 d_2 d_3
Output 0 a_0 a_3 a_2 a_1 b_0 b_3 b_2 b_1 c_0 c_3 c_2 c_1
Clock
Memory a_0,a_1,a_2,a_3 b_0,b_3,b_2,b_1 c_0,c_1,c_2,c_3

Fig. 8: Timing diagram of data buffer access process using data remapping ($2^{n-k} = 4$)

access process using the proposed data remapping technique. Note that the sing-ported memory supporting simultaneous read-write operation (read first) is required.

2) Address vectors generation: To implement our proposed data remapping technique, an address generation method is required. We use Eq. (1) to represent a permutation in time. For a given invertible bit permutation matrix P, assuming $A_0 = [0, 1, 2, ..., 2^{n-k}]$, we can compute all the address vectors by iteratively computing:

$$A_{i+1} = P \times A_i, 0 \le i \le q - 1 \quad (3)$$

such that $A_0 = P \times A_q$. According to [11], there always exists a q so that $P^{q+1} = I$ (Identity matrix) for any given permutation matrix P. The number of address vectors q depends on the permutation matrix P. In this paper, totally there are only two address vectors required in the case that stride is set to streaming width. In Section IV-D, we evaluate the required extra logic resources using our proposed data remapping technique.

D. Implementation of interconnection network

The interconnection network actually is a cyclic shifter, i.e., shift the data elements downward or upward in space cyclically based on a given distance. Using different interconnection networks can significantly affect the energy efficiency of the designs. Three well known networks (see Fig.9), including crossbar network, tree-based network, as well as dynamic network, can be utilized to perform the cyclic shift. As shown in Fig. 9, assuming the number of inputs is p, then crossbar network requires $O(p)$ control bits, tree-based network and dynamic network requires $O(p \log p)$ control bits. However, on the state of the art FPGAs, switch matrix with a structure of the crossbar network is inaccessible to users. Dynamic networks such as Benes network [8] or Clos network [7] can be employed to perform cyclic shift with arbitrary distance. But due to a large number of network stages, the permutation latency increases significantly with the growing p. Although the dynamic network can be pipelined using registers to improve latency, the power consumption becomes considerable simultaneously. Hence we select the tree-based network implemented with multiplexers on FPGAs to perform the cyclic shift. Multiplexer is a low power component on FPGA. We use them for implementation of interconnection networks for

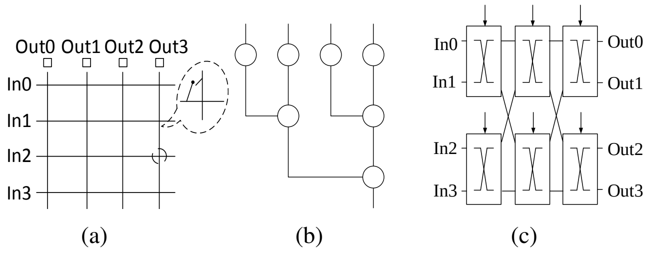


Fig. 9: (a) Crossbar network, (b) Tree-based network, (c) Dynamic network

power optimization purpose. The physical layout of the tree-based network is flexible to be inserted with pipeline registers between the tree layers. In each cycle, only one shift distance is required to perform the cyclic shift. Furthermore, the shift distance is updated using gray coding so that the number of glitches between cycles can be minimized, therefore optimize the glitch power.

IV. Experiments

A. Experimental setup

The baseline and proposed architectures are implemented in Verilog on Virtex-7 FPGA (XC7VX980T, speed grade -2L) using Xilinx ISE 14.4. The designs are simulated to verify the functional correctness. The input test vectors (16-bit fixed-point) for simulation were randomly generated and had an average toggle rate of 50%. Post place-and-route simulation was also performed to ensure that the design was synthesized and mapped correctly on FPGA. We used VCD files (value change dump file) as inputs to Xilinx XPower Analyzer to produce accurate power dissipation estimation [12]. We set the operating frequency to 333 MHz for power evaluation purpose.

B. Performance Metrics

Two metrics for performance evaluation are considered in this paper:

1) *Energy efficiency* is calculated as number of data points produced per unit energy, and we still use points/Joule as the measure unit. As throughput is defined as number of data points produced per second, then we can compute as:

$$\text{Energy efficiency} = \frac{\text{Throughput}}{\text{Average power}} \tag{4}$$

2) *Energy $\times$ Area $\times$ Time (EAT)* is measured as the product of three key metrics: energy, area, and time. We use EAT ratio for performance comparison between different designs. *Area* is the area usage of the design, i.e. the number of LUTs or flip-flops (the larger one will be selected) occupied by the entire design. The BRAM slides will be transferred to a certain amount of LUTs based on the memory size, thus we can obtain the area of BRAMs. *Time* is the latency for pipelining the proposed architecture.

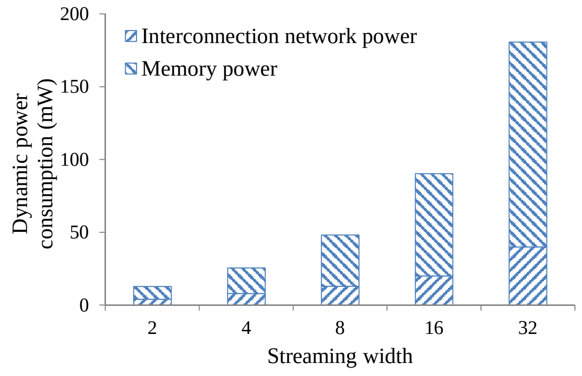


Fig. 10: Power profile for the baseline architecture

C. Power profile for the baseline architecture

We conduct power profile for the baseline architecture and vary the streaming width w from 2 to 32. Stride-by-w permutation is performed. As shown in Fig 10, the memory power is dominant in the stride permutation architecture for various streaming width. The memory power dissipation accounts for $64\% \sim 78\%$ of the total dynamic power consumption of the design. As streaming width grows, this percentage slowly decreases due to increasing routing power in the design of interconnection networks. The figure also shows that the memory power tends to increase linearly with the streaming width. As the memory consumption also increases with the growing streaming width, this tendency is caused by the required more BRAM blocks. The profile result indicates that the energy efficiency of the stride permutation architecture can be optimized by reducing memory power. Our proposed data remapping technique is inspired by this observation.

D. Optimization results using the proposed data remapping technique

We conduct experiments to evaluate the resource and power consumption results of both the baseline architecture and our proposed architecture. 512-element data vectors are taken as inputs for permutation. Both streaming width and data width are varied for performance evaluation.

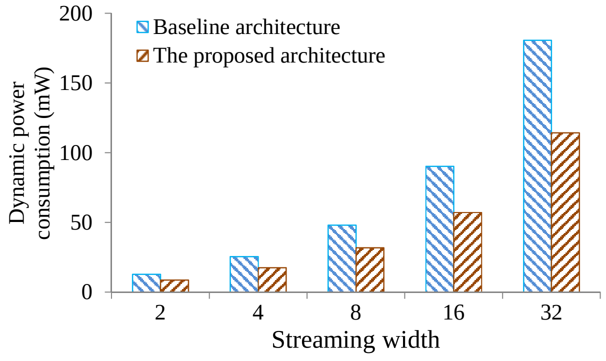


Fig. 11: Power optimization results with varying streaming width

TABLE I: Resource consumption results with varying streaming width

Streaming width	Baseline architecture		The proposed architecture	
	# of BRAM (18-kb block)	LUT Slices	# of BRAM (18-kb block)	LUT Slices
2	4	56	2	68
4	8	110	4	132
8	16	210	8	226
16	32	405	16	454
32	64	789	32	812

1) Power and resource consumption results with varying streaming width: We vary the streaming width w from 2 to 32 when taking 8-bit data elements as inputs. Stride-by-w permutation is performed. As shown in Fig. 11, the power consumption is highly reduced by using our proposed data remapping architecture. As streaming width increases, the total dynamic power of the baseline architecture is reduced by $31\% \sim 36\%$ using the proposed remapping technique. This suggests that extra required logic required by the proposed data remapping technique only introduces a slight amount of power dissipation for various streaming width.

Table I shows the resource consumption results of the two target architectures. When the streaming width increases, the total memory consumption, i.e., the number of BRAM blocks used on FPGAs, is reduced by 50%. As BRAM power on FPGAs is mainly determined by the number of BRAM blocks used, this indicates that the total BRAM power consumption can be reduced by up to 50%, which matches with our power optimization results. The number of LUT slices increases by 8% to 15% as the streaming width increases. The required extra LUT slices are introduced by our proposed data remapping technique for address generation.

2) Power and resource consumption results with varying data width: We vary data width from 4 to 64 bits when setting the streaming width as 4. Stride-by-4 permutation is performed. Fig.12 shows the dynamic power dissipation of the two target architectures by varying the data width. It shows that the power optimization results are not affected by data width, i.e, our proposed data remapping technique is not sensitive to the data width. As data width increases, the total dynamic power of the baseline architecture is reduced by $32\% \sim 40\%$ using the proposed data remapping technique.

Table II shows the resource consumption results of the two target architectures with varying the data width. From the figure, the number of BRAM blocks required is affected by the data width. When the data width is less than 16-bit, the number of BRAM blocks used in the two target architectures is a constant. However, when further doubling the data width, the number of BRAM blocks is also doubled accordingly. This tendency is due to the limited maximum data width supported by the 18-kb BRAM block. We also observe that the consumed LUT slices in our proposed architecture only increase by $10\% \sim 15\%$.

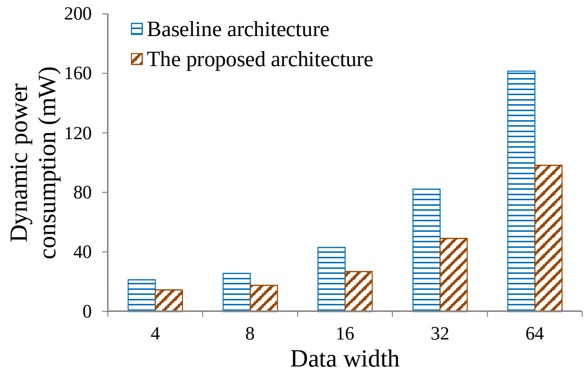


Fig. 12: Power optimization results of the proposed architecture with varying data width

TABLE II: Resource consumption results with varying data width

Data width	Baseline architecture		The proposed architecture	
	# of BRAM (18-kb block)	LUT Slices	# of BRAM (18-kb block)	LUT Slices
4	8	89	4	108
8	8	110	4	132
16	8	138	4	151
32	16	172	8	205
64	32	251	16	296

E. Energy efficiency of the proposed architecture

In this experiment, we compare the energy efficiency and EAT ratio between our proposed architecture and the baseline architecture. We take 512 real numbers as inputs. We vary the data width w from 2 to 32. Stride-by-w permutation is performed. As shown in Fig. 13, our proposed architecture achieves a high energy efficiency up to 75.3 giga points/Joule. As the streaming width increases, the proposed architecture can still sustain a high energy efficiency. Our proposed architecture owns a EAT ratio of 0.31 to 0.35 over the baseline architecture. This indicates that our proposed design is both power and area efficient comparing with the baseline architecture.

V. Conclusion

In this paper, we present an energy efficient architecture for stride permutation on streaming data. Stride permutation on power-of-two data elements is performed. A three-stage structure using interconnection networks and data buffers is employed. We develop a data remapping technique, which can halve the memory consumed by the data buffers, hence optimize the memory power. A low power multiplexer-based cyclic shift interconnection network is implemented. The experimental result show that, by reducing the memory power, our proposed streaming permutation architecture could sustain a high energy efficiency for various streaming width. Besides, the data remapping technique only introduces a slight amount of extra logic resource consumption. In the future, we plan

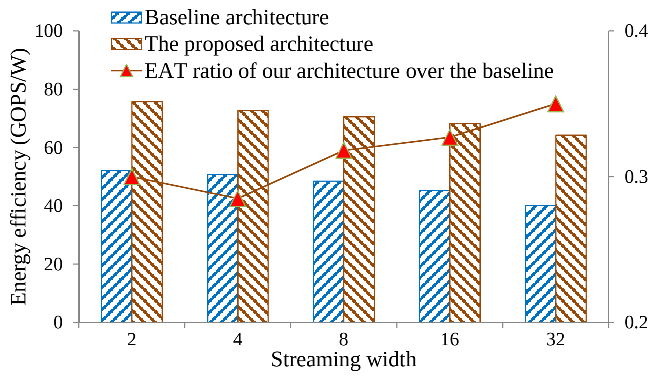


Fig. 13: Energy efficiency comparison between the baseline architecture and the proposed architecture with varying streaming width

to apply our proposed data remapping technique in more application implementations.

References

[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[2] M. Boo, F. Arguello, J. Bruguera, R. Doallo, and E. Zapata, "High-performance VLSI architecture for the Viterbi algorithm," *Communications, IEEE Transactions on*, vol. 45, no. 2, pp. 168–176, 1997.

[3] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Proceedings of IPPS '96*, pp. 766–770.

[4] G. Bi and E. Jones, "A pipelined FFT processor for word-sequential data," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 12, pp. 1982–1985, 1989.

[5] T. Jarvinen, P. Salmela, H. Sorokin, and J. Takala, "Stride permutation networks for array processors," in *Application-Specific Systems, Architectures and Processors, 2004. Proceedings. 15th IEEE International Conference on*, 2004, pp. 376–386.

[6] 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.

[7] H. M. Alnuweiri and V. K. Prasanna, "Optimal multipass self-routing algorithms for clos-type multistage networks." in *ICPP*, 1992, pp. 118–122.

[8] D. Nassimi and S. Sahni, "A self routing benes network," in *Proc. of the 7th annual symposium on Computer Architecture*. ACM, 1980, pp. 190–195.

[9] R. Chen, H. Le, and V. K. Prasanna, "Energy efficient parameterized FFT architecture," *To appear in Proc. of IEEE International Conference on FPL*, 2013.

[10] R. Chen, N. Park, and V. K. Prasanna, "High throughput energy efficient parallel FFT architecture on FPGAs," *To appear in Proc. of IEEE International Conference on HPEC*, 2013.

[11] J. Granata, M. Conner, and R. Tolimieri, "Recursive fast algorithm and the role of the tensor product," *IEEE Transactions on Signal Processing*, vol. 40, no. 12, pp. 2921–2930, 1992.

[12] "XST user guide for Virtex-6, Spartan-6, and 7 series devices," http://www.xilinx.com/support/documentation.