Novel Reconfigurable Hardware Architecture for Polynomial Matrix Multiplications

Soydan Redif, Senior Member, IEEE, and Server Kasap, Member, IEEE

Abstract—In this paper, we introduce a novel reconfigurable hardware architecture for computing the polynomial matrix multiplication (PMM) of polynomial matrices and/or polynomial vectors. The proposed algorithm exploits an extension of the fast convolution technique to multiple-input multiple-output systems. The proposed architecture is the first one devoted to the hardware implementation of PMM. Hardware implementation of the algorithm is achieved via a highly pipelined, partly systolic field-programmable gate array (FPGA) architecture. The architecture, which is scalable in terms of the order of the input polynomial matrices, has been designed using the Xilinx system generator tool. We verify the algorithmic accuracy of the architecture through FPGA-in-the-loop hardware cosimulations. The application to sensor array signal processing is highlighted, in terms of strong decorrelation. The results are presented to demonstrate the accuracy and capability of the architecture. The results verify that the proposed solution gives low execution times while limiting the number of required FPGA resources.

Index Terms—Field-programmable gate array (FPGA), polynomial matrix computations, polynomial matrix multiplication (PMM), SBR2P, Xilinx system generator for digital signal processor (DSP) tool.

I. INTRODUCTION

POLYNOMIAL matrix techniques equivalent to the singular value decomposition and eigenvalue decomposition (EVD) [1] for scalar matrices have received growing interest in recent years. They have been successfully applied to broadband extensions of narrowband problems, which traditionally have been addressed by the EVD. Applications include broadband sensor array signal processing (SASP) [2]–[6], biomedical engineering [7], multiple-input multiple-output (MIMO) communications and coding [8]–[12], and sub-band coding [13]–[17]. The EVD of a para-Hermitian system, or polynomial matrix EVD (PEVD), yields a factorization of a para-Hermitian polynomial matrix into a product consisting of a diagonal polynomial matrix that is pre and postmultiplied by paraunitary (PU) polynomial matrices. A PU polynomial matrix preserves the total signal power at every frequency [18], and so can be viewed as a lossless (stable, all-pass) filter bank.

McWhirter et al. [4] propose an extension of the EVD to para-Hermitian polynomial matrices, called the second-order sequential best rotation (SBR2) algorithm. It was originally developed for the purpose of generating a finite-impulse response (FIR) PU matrix to diagonalize the para-Hermitian polynomial matrix [4] of signals received by a broadband sensor array. Signals that give rise to a diagonalized para-Hermitian matrix have the strong (total) decorrelation property [4], [13], a desired quality in, e.g., SASP [4], [5] and sub-band coding [17].

Due to the computational complexity involved in manipulating polynomial matrices, the polynomial matrix diagonalization mentioned above is poorly suited for high-speed or real-time applications. In [19], we described a method for parallelizing SBR2, namely SBR2P, which produced a diagonalized para-Hermitian polynomial matrix and the corresponding FIR PU filter bank. We showed how it can be implemented in hardware with a highly pipelined nonsystolic field-programmable gate array (FPGA) architecture. FPGAs provide a significant speedup in computation time compared with PC-based software. They combine the main advantages of application-specific integrated circuits and digital signal processors (DSPs) with the potential for dynamic reconfiguration.

McWhirter et al. [4] and Redif et al. [5] use the SBR2-generated FIR PU filter bank to impose strong decorrelation on the set of sensor signals for the problem of broadband SASP. This operation can be viewed as the polynomial matrix multiplication (PMM) of a polynomial matrix with a polynomial vector (or matrix–vector PMM). We also often want to apply the PU filter bank to the para-Hermitian polynomial matrix directly, as in [9], [16], and [17], to realize PEVD factorization without having to run SBR2 each time. This operation can be looked upon as the product of polynomial matrices (matrix–matrix PMM).

Various signal processing tasks can be realized in real systems with the help of PMM in conjunction with PEVD. Specifically, PMM has been a vital pre and/or postprocessing stage in exploiting the broadband signal subspace estimated by a PEVD algorithm for processing real underwater acoustics data acquired from eight-element array in [5] and for broadband angle of arrival estimation using four-element array [6]. It played an important role when extracting fetal components from real life, contaminated electrocardiogram data derived from eight electrodes in [7]. PMM facilitated efficient application of the PEVD-identified generator and parity check polynomial matrices in [9], and the precoders and equalizers for MIMO communications (involving four to six channels) in [10]–[12]. It has also enabled the employment of PEVD-designed four-channel PU filter banks for optimal sub-band coding [15]–[17].
To make effective use of the SBR2P architecture in real-time applications, we require the hardware implementation of the aforementioned two types of PMM operations. To date, however, no attention seems to have been devoted to the hardware realization of PMM. The development and application of such hardware is the subject of this paper. Specifically, we consider the development of an efficient, highly pipelined, and partly systolic FPGA architecture for matrix-vector and matrix-matrix PMM operations. This architecture implements an extension of the well-known fast Fourier transform (FFT)-based fast convolution technique [20] to MIMO systems, referred to here as fast MIMO convolution. We show how the architecture can be used with the SBR2P architecture in [19] to realize the PEVD factorization and PU filtering of multichannel data in the context of broadband SASP.

The proposed architecture gives low execution times while utilizing limited FPGA resources. It yields a good approximation to the polynomial matrix computation provided by its double-precision counterpart running in MATLAB for both real- and complex-valued data. The architecture has been developed using the Xilinx system generator for DSP tool [21], which offers a visual interface and a number of standard modules for speedy design.

Finally, the proposed architecture is meant as a generic tool for polynomial matrix operations. Therefore, it is not exclusively designed for use with SBR2P, nor is its application limited to SASP. For example, the architecture could be used with a future FPGA implementation of other polynomial matrix methods, such as the polynomial matrix QR decomposition algorithm in [12]. As such, we believe that this paper possesses some tutorial value, continuing in a similar vein to [19]. In particular, it highlights the importance of newly developed techniques for FPGA-based design of numerical methods for polynomial matrix computation.

This paper is organized as follows. Section II presents the problem of PMM and its connection to PEVD in the context of SASP. Section III proposes an algorithm for PMM that can exploit FPGA parallelism, enabling application of the SBR2P-generated PU matrix in hardware. Section IV proposes a novel FPGA architecture for realizing the parallel algorithm. Section V first outlines how the architecture for PMM may be used in conjunction with SBR2P for broadband SASP, and then presents the results of some hardware simulations designed to demonstrate the accuracy of our architecture. Furthermore, an analysis of the timing performance of our architecture as well as the hardware resources consumed in the utilized Xilinx Virtex-5 XC5VLX110T FPGA chip is provided. Finally, the conclusion is given in Section VI.

II. POLYNOMIAL MATRIX MULTIPLICATION

In this section, after a brief review of polynomial matrix fundamentals, we discuss PMM as a tool for use with PEVD algorithms in the context of broadband SASP, and investigate its computational complexity.

A. Preliminaries and Notation

Throughout this paper, matrices are denoted by upper case bold characters, e.g., $X$, and vectors by lower case bold, e.g., $x$. The $(j,k)$ element of the matrix $X$ is denoted by $[X]_{jk}$. The superscripts $\ast$, $T$, and $H$ denote complex conjugation, matrix transposition, and Hermitian transpose, respectively. A matrix $R[\tau]$ is a Hermitian matrix iff $R = R^H$ and a unitary matrix $H \in \mathbb{C}^{p \times p}$ is one that satisfies $H^H H = I_p$, where $I_p$ is the $p \times p$ identity matrix. A polynomial matrix is a matrix with polynomial elements, or equivalently a polynomial with matrix coefficients [22]. Denote a $p \times q$ polynomial matrix in $z^{-1}$ by

$$A(z) = \sum_{\tau=0}^{t_2} A[\tau] z^{-\tau}$$  \hspace{1cm} (1)

where $\tau \in \mathbb{Z}$, $t_1 \leq \tau \leq t_2$. The polynomial matrix $A(z) \in \mathbb{C}^{p \times q}$ has entries $a_{ik}(z) = \sum_{\tau=0}^{t_2} a_{ik}[\tau] z^{-\tau}$ with $a_{ik}[\tau] \in \mathbb{C}$. A transform pair as in (1) is denoted as $A(z) \longrightarrow A[\tau]$. The order of the polynomial matrix $A(z)$ is $(t_2 - t_1)$, where $t_1$ and $t_2$ are not necessarily positive. The matrices $A[t_1], \ldots, A[t_2]$ are referred to as the coefficient matrices of $A(z)$. The degree of $A(z)$ is the number of delay units required for its realization.

A polynomial matrix $H(z) \in \mathbb{C}^{p \times p}$ is said to be PU if it satisfies $H(z)H(z) = H(z)H(z) = I$ [18], where $H(z) = H^{H}(1/z^*)$ is the parac conjugate of $H(z)$. In the context of DSP and linear system theory, $H(z)$ represents a lossless MIMO transfer function [22]. A polynomial matrix $R(z) \in \mathbb{C}^{p \times p}$ is para-Hermitian if $R(z) = R(z)$, so $[R[\tau]]_{ik} = r_{ik}[\tau] = r_{ik}^*(-\tau) = [R[-\tau]]_{ik}, \forall \tau \in \mathbb{Z}$ and $i, k = 1, 2, \ldots, p$. An important example of a para-Hermitian polynomial matrix is the polynomial cross-spectral density (CSD) matrix

$$R(z) = \sum_{\tau=-N_R}^{N_R} R[\tau] z^{-\tau}$$  \hspace{1cm} (2)

where the sequence $\{R[\tau]\}_{\tau=-N_R}^{N_R}$ denotes the set of constant correlation matrix coefficients

$$R[\tau] = \mathbb{E}[x[\tau]x^H[\tau - \tau]], \quad \tau \in \mathbb{Z}$$  \hspace{1cm} (3)

where $\mathbb{E}\{\cdot\}$ denotes the expectation operator.

B. Computing the PEVD

As mentioned already, polynomial matrices are finding application in many areas, including, particularly, broadband SASP. In this context, the signals in $x[\tau]$ can no longer be decorrelated using the EVD [4]. This problem occurs in many other applications where, for example, the source signals are broadband or undergo convolutive mixing, and so they cannot be related in terms of simple phase and amplitude factors.

Instead, it is necessary to impose strong decorrelation [13]

$$\mathbb{E}\left[ \sum_{\tau \neq i} x_i[\tau] x_k[\tau - \tau] \right] = 0$$  \hspace{1cm} (4)

for all $\tau$ and $i \neq k$. One way of satisfying (4) is to find a FIR PU matrix $H[\tau] \in \mathbb{C}^{p \times p}$ with transform $H(z) \longrightarrow H[\tau]$ that diagonalizes the space–time covariance matrix...
\[ R[\tau] = E[x[r]x^H[t - \tau]]. \] In other words, we require the transformation
\[ S[t] = \sum_{\tau=0}^{N_H} \left( \sum_{\tau=0}^{N_H} H[\tau] R[t - \tau] \right) H^H[t - t] \quad (5) \]
or
\[ S[t] = H[\tau] \ast R[t] \ast H^H[-t] \quad (6) \]
where \( N_H + 1 \) is the number of matrix coefficients of \( H[t], S[t] = \text{diag}(s_{11}[\tau], \ldots, s_{pp}[\tau]), \) \( \text{diag}\{\} \) denotes a diagonal matrix, and the asterisk denotes convolution. In terms of polynomial matrices, we require
\[ S(z) = H(z)R(z)\tilde{H}(z) = \text{diag}(s_{11}(z), \ldots, s_{pp}(z)) \quad (7) \]
which is the PEVD of \( R(z) \). The para-Hermitian polynomial matrix \( S(z) \) $\mapsto S[t]$ is the CSD matrix of the PEVD-transformed signals
\[ v[t] = \sum_{\tau=0}^{N_H} H[\tau] x[t - \tau] = H[t] \ast x[t]. \quad (8) \]
The signals \( v[t] \in \mathbb{C}^p \), which satisfy strong decorrelation in (4), are the result of applying the PU matrix \( H[t] \), from the transformation in (6), to the received signal vector \( x[t] \in \mathbb{C}^p \). The MIMO convolution in (8) can be expressed as the product of the PU matrix \( H(z) \) and the power series \( x(z) = \sum_{\tau} x[\tau] z^{-\tau} \), thus
\[ v(z) = H(z)x(z). \quad (9) \]

### C. Fast MIMO Convolution

Equation (9) can be represented in the discrete-time Fourier transform (DTFT) domain by way of a trivial extension of the convolution theorem [20] to MIMO systems. Thus, for all normalized angular frequencies \( \Omega \)
\[ v(e^{j\Omega}) = H(e^{j\Omega}) x(e^{j\Omega}) \quad (10) \]
where \( H(e^{j\Omega}) = H(z)|_{z=e^{\Omega}} \in \mathbb{C}^{p \times p} \) are unitary matrices and \( x(e^{j\Omega}) = x(z)|_{z=e^{\Omega}} \in \mathbb{C}^p \). Equation (10) can be approximated by uniformly sampling the DTFT, which yields
\[ v(e^{j\Omega_k}) = H(e^{j\Omega_k}) x(e^{j\Omega_k}) \quad (11) \]
for \( k = 0, \ldots, N - 1 \), where \( N \) is the number of frequency bins and \( \Omega_k = 2\pi k/N \). Such a transformation can be obtained by taking the \( N \)-point discrete Fourier transform (DFT) of \( v[t] \):
\[ v(e^{j\Omega_k}) = \sum_{m=0}^{N-1} v[m] e^{-j\Omega_k m}, \] where \( v[z] \) is a zero-padded version of \( v[t] \), \( N > N_H + N_s - 1 \), and \( N_s \) is the number of signal samples. Hence, (8) can be approximated by
\[ v[t] \approx \frac{1}{N} \sum_{k=0}^{N-1} v(e^{j\Omega_k}) e^{j\Omega_k t}. \quad (12) \]
As \( N \) gets large, (12) converges to \( v[t] \) in (8).

It is well known that, for reasonably long convolutions (Section II-D), the application of the FFT is more efficient than using the DFT [23]. The DFT in (12) can be implemented as
\[ v[t] = \text{IFFT}_N^p \{ \text{FFT}_N^{p \times p} [H[t]] \text{FFT}_N^p [x[t]] \} \quad (13) \]
where \( \text{FFT}_N^{p \times p} \) denotes the \( N \)-point FFT—along the third-dimension—of a sequence of \( p \times p \) matrices. Similarly, \( \text{IFFT}_N^p \) and \( \text{IFFT}_N^{p \times p} \) denote, respectively, the \( N \)-point FFT and inverse FFT (IFFT) along the third dimension of a sequence of \( p \)-dimensional vectors. The expression in (13) is represented as a block diagram in Fig. 1(a).

In certain applications, such as MIMO encoding [9], [16], we will need to carry out the diagonalization of para-Hermitian polynomial matrices, as in (7), using a predesigned (stored) \( H(z) \). This requires two matrix–matrix PMMs, which can be implemented by trivial extension of (13). Consider, for example, the transformation \( C[t] = H[t] \ast R[t] \), which can be viewed as part of that in (6). This can be implemented as follows:
\[ C[t] = \text{IFFT}_N^{p \times p} \{ \text{FFT}_N^{p \times p} [H[t]] \text{FFT}_N^p [R[t]] \} \quad (14) \]
where \( \text{IFFT}_N^{p \times p} \) denotes the \( N \)-point IFFT along the third dimension of a sequence of \( p \times p \) matrices, as shown in Fig. 1(b).

The two fast MIMO convolution techniques described in this section are fundamental to the aim of applying hardware implemented PEVD algorithms, particularly SBR2P [19], to high-speed or real-time problems.

### D. Complexity of Fast MIMO Convolution

Straightforward calculation of the matrix–vector MIMO convolution in (8) is computationally intensive, requiring \( O(p^2 N^2) \) time. By the same token, matrix–matrix MIMO convolution, as in (6), requires \( O(p^3 N^2) \) time on a sequential processor. However, fast MIMO convolution in (13) is more efficient when \( \log_2(N_H + N_s - 1) < p \min(N_H, N_s) - 1 \). In which case, computation of (8) requires only \( O(p^2 N \log_2(N)) \) operations instead of \( O(p^2 N^2) \). If implemented on a dedicated parallel processing system, such as FPGA, fast MIMO convolution would require \( O(p^2 N \log_2(N)/\kappa) \) time, \( \kappa \) being the number of processing units.

At present, there does not seem to be any contributions focusing on the hardware implementation of PMM. In the following sections, we look to implement PMM by parallelizing the fast MIMO convolutions of (13) and (14).
III. PARALLEL ALGORITHM FOR PMM

In this section, we introduce an algorithm for multiplication of polynomial matrices using the fast MIMO convolution technique of Section II-C, for which an efficient FPGA architecture is described in Section IV. The algorithm starts by taking the FFT of the input polynomial matrices (or vectors), and proceeds with the conventional matrix multiplication of FFT-transformed matrices (or vectors), as in Fig. 1(b) [or Fig. 1(a)]. The parallel matrix-multiplication architecture in [24] was used as a part of the proposed scheme. The process terminates with the IFFT of the sequence of matrix products.

Our design methodology is based on a parallel array design that maps a nested-loops algorithm onto the parallel architecture. Parallel array design effectively exploits the inherent parallelism of PMM. FPGAs can be used to efficiently implement these fine grain arrays since they inherently possess the same regular structure.

Pseudocode of the proposed PMM algorithm is shown in Algorithm 1. After computing the FFT of the input polynomial matrices, the Fourier-domain matrix sequences are decomposed into their real and imaginary components to simplify subsequent computations. Two deeply nested loops then compute, respectively, the real and imaginary components of the output sequence of matrix products, concurrently. Each of the nested-loop bodies comprises two recurrence equations for the calculation of the complex matrix products. These nested loops can easily be unlooped for the \( i \) and \( j \) indices, and then mapped to a 2-D systolic array for parallel execution. The process concludes with computation of the IFFT of the matrix sequence resulting from the complex combination of the outcomes of the two nested loops, forming the output polynomial matrix. Note that the algorithm is readily extended to polynomial matrix–vector product computations, since a polynomial vector is nothing but a single-column or single-row polynomial matrix.

IV. PROPOSED FPGA ARCHITECTURE

A. Polynomial Matrix Calculation in Hardware

In this section, we describe the proposed hardware implementation of the fast MIMO convolution technique that performs PMM. The algorithm comprises the following steps.

1) First, the FFT of the two \( p \times p \) input polynomial matrices is computed. This is done on a row-by-row basis for the multiplier polynomial matrix, and on a column-by-column basis for the multiplicand. The resulting matrix sequences, with real and imaginary components, are stored into two sets of similar memory blocks to be used afterward; sets are correspondingly allocated to each transformed polynomial matrix. Each memory block contains two dual-port memories for real and imaginary components. All computations of this step are performed by two FFT blocks operating in parallel and configured in forward mode—explained later.

2) The next step is to conventionally multiply the matching elements of the two matrix sequences resulting from the first step in such a way that a sequence of matrix products composed of only the real components is obtained, as explained later in this section. In this step, the process takes place in two phases during each of which an exclusive equation is recursively evaluated for the calculation of the complex matrix products. The results are accordingly stored into a second set of similar memory blocks, each of which incorporates two dual-port memories. For this step, only the first dual-port memory within each memory block is utilized.

<table>
<thead>
<tr>
<th>Algorithm 1</th>
<th>Pseudocode for the Proposed Matrix–Matrix PMM Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{input}: H, R polynomial matrices (i.e. sequences of N p \times p matrices)</td>
<td></td>
</tr>
<tr>
<td>\textbf{output}: C polynomial matrix (i.e. a sequence of N p \times p matrices)</td>
<td></td>
</tr>
<tr>
<td>\textbf{recH}<em>{\text{fft}} = \text{Re}{\text{FFT}</em>{N}^{p\times p}{H}};</td>
<td></td>
</tr>
<tr>
<td>\textbf{imH}<em>{\text{fft}} = \text{Im}{\text{FFT}</em>{N}^{p\times p}{H}};</td>
<td></td>
</tr>
<tr>
<td>\textbf{recR}<em>{\text{fft}} = \text{Re}{\text{FFT}</em>{N}^{p\times p}{R}};</td>
<td></td>
</tr>
<tr>
<td>\textbf{imR}<em>{\text{fft}} = \text{Im}{\text{FFT}</em>{N}^{p\times p}{R}};</td>
<td></td>
</tr>
<tr>
<td>for ( t = 1 \rightarrow N ), ( i = 1 \rightarrow p ), ( j = 1 \rightarrow p ) do</td>
<td></td>
</tr>
<tr>
<td>( \text{recC}<em>{\text{fft}}[i, j, t] = 0; \text{imC}</em>{\text{fft}}[i, j, t] = 0; )</td>
<td></td>
</tr>
<tr>
<td>for ( i = 1 \rightarrow p ) do</td>
<td></td>
</tr>
<tr>
<td>for ( j = 1 \rightarrow p ) do</td>
<td></td>
</tr>
<tr>
<td>for ( t = 1 \rightarrow N ) do</td>
<td></td>
</tr>
<tr>
<td>if ( k \leq p ) then</td>
<td></td>
</tr>
<tr>
<td>( \text{recC}<em>{\text{fft}}[i, j, t] = \text{recC}</em>{\text{fft}}[i, j, t] + \text{recH}<em>{\text{fft}}[i, k, t] \times \text{recR}</em>{\text{fft}}[k, j, t]; )</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td></td>
</tr>
<tr>
<td>( \text{recC}<em>{\text{fft}}[i, j, t] = \text{recC}</em>{\text{fft}}[i, j, t] - \text{imH}<em>{\text{fft}}[i, k-p, t] \times \text{imR}</em>{\text{fft}}[k-p, j, t]; )</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>for ( i = 1 \rightarrow p ) do</td>
<td></td>
</tr>
<tr>
<td>for ( j = 1 \rightarrow p ) do</td>
<td></td>
</tr>
<tr>
<td>for ( k = 1 \rightarrow 2p ) do</td>
<td></td>
</tr>
<tr>
<td>for ( t = 1 \rightarrow N ) do</td>
<td></td>
</tr>
<tr>
<td>if ( k \leq p ) then</td>
<td></td>
</tr>
<tr>
<td>( \text{imC}<em>{\text{fft}}[i, j, t] = \text{imC}</em>{\text{fft}}[i, j, t] + \text{recH}<em>{\text{fft}}[i, k, t] \times \text{imR}</em>{\text{fft}}[k, j, t]; )</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td></td>
</tr>
<tr>
<td>( \text{imC}<em>{\text{fft}}[i, j, t] = \text{imC}</em>{\text{fft}}[i, j, t] + \text{imH}<em>{\text{fft}}[i, k-p, t] \times \text{recR}</em>{\text{fft}}[k-p, j, t]; )</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>( C_{\text{fft}} = \text{Complex}(\text{recC}<em>{\text{fft}}, \text{imC}</em>{\text{fft}}); )</td>
<td></td>
</tr>
<tr>
<td>( C = \text{IFFT}<em>{N}^{p\times p}{C</em>{\text{fft}}}; )</td>
<td></td>
</tr>
</tbody>
</table>
These computations are achieved by a 2-D systolic array [24] composed of \( p \times p \) processing elements (PEs) specifically designed for this purpose.

3) The third step is very similar to step 2; however, this time, the matching elements of the resulting two matrix sequences from the first step are multiplied so as to obtain the imaginary components of the matrix product sequence. Therefore, in this step, two new exclusive equations are recursively evaluated in again two phases, respectively. Furthermore, the results of this step are stored into the same memory block set as in step 2, but the memory blocks’ second dual-port memories are solely utilized in this case. The computations of this step are also accomplished by the \( p \times p \) systolic array mentioned above after the completion of step 2.

4) Finally, the algorithm concludes with the IFFT of the contents of the memory blocks populated by the outcomes of steps 2 and 3 to form the output polynomial matrix product. The IFFT process is applied to two memory blocks at a time in a specified order—explained later. These computations are performed by the same two parallel FFT blocks as in step 1; however, this time, they are configured in inverse mode.

The above four steps constituting the proposed hardware algorithm are sequentially executed in a pipelined manner for sets of input polynomial matrices to achieve the maximum throughput possible.

### B. Architecture for Polynomial Matrix Calculation

In this section, we describe the partly systolic highly pipelined architecture implementing the hardware algorithm detailed in Section IV-A for \( 4 \times 4 \) polynomial matrices, (i.e., \( p = 4 \)). The FPGA architecture for PMM has been developed using the Xilinx system generator for DSP 14.5 tool. It consists of the DataPath block that performs data processing operations along with a control unit (i.e., Main_FSM block), incorporating a finite-state machine (FSM), which largely regulates interaction between the DataPath block and the data itself. The DataPath block, shown in Fig. 2, is comprised of two sets of memory blocks and two main blocks, namely FFT_IFT and SystolicArray blocks, sequentially implementing the four major steps of the fast MIMO convolution technique. Outputs of the FFT_IFT block are stored into the first set of memory blocks: MemI1, MemI2, MemI3, MemI4, MemI5, MemI6, MemI7, and MemI8.
On the other hand, outputs of the SystolicArray block are stored into the second set of memory blocks: MemO1, MemO2, MemO3, and MemO4. As mentioned above, each memory block incorporates two dual-port memories for storing real and imaginary values.

An important aspect of the proposed architecture is that it does not have high dependency on the order of the input polynomial matrices. Only the size of the dual-port RAM memories throughout the architecture as well as some specific parameter values, explained later in this section, need to be modified when the order of the input matrices is changed. Furthermore, Xilinx fixed-point data type is used throughout our architecture, where rounding and saturation have been chosen as quantization and overflow options, respectively, for all arithmetical units, e.g., multipliers and adders, to improve the accuracy of the computations.

1) FFT_IFFT Block: The inner structure of the FFT_IFFT block is shown in Fig. 3. As can be seen, it mainly consists of two simultaneously operating FFT blocks, (i.e., FFT 7.1 modules [25]), which can be configured in forward or inverse mode depending on the fwd_inv signal. The block also contains four multiplexers (as well as a number of other miscellaneous blocks) to select one input signal from the corresponding set of three data signals and direct it accordingly to the xn_re or xn_im ports of the FFT blocks, which are for inputting real and imaginary values, respectively.

The FFT_IFFT block has two modes of operation. In mode 1, the FFT blocks perform the FFT on each polynomial element of multiplier and multiplicand matrices in parallel, respectively, in a row-by-row and column-by-column sequential processing manner. In effect, the transformation is along the third dimension of the polynomial matrix. Complex row vector sequences resulting from the FFT of each polynomial element row for the multiplier matrix are, respectively, stored into the MemI1, MemI2, MemI3, and MemI4 memory blocks where sequences for the polynomial elements are individually placed within the memory, whereas complex column vector sequences resulting from the transformation of each polynomial element column for the multiplicand matrix are, respectively, kept in the MemI5, MemI6, MemI7, and MemI7 blocks in the same way.

In mode 2, FFT blocks compute the IFFT of the contents of MemO1, MemO2, MemO3, and MemO4, populated by the SystolicArray block, to obtain the architecture output: polynomial matrix product. The IFFT process is initially applied to the sequentially read complex data from the MemO1 and MemO2 blocks by the respective FFT blocks in parallel, and then complex data from the MemO3 and MemO4 blocks are, respectively, processed likewise by the FFT blocks at the same time. Note that multiplexers within the FFT_IFFT block architecture are utilized to direct the data from the memory blocks to the FFT blocks, accordingly. The operations within this block are controlled by the Main_FSM block.

2) SystolicArray Block: The SystolicArray block is at the core of our datapath architecture for PMM (Fig. 2), containing a 2-D systolic array composed of $4 \times 4$ identical PEs, namely PE00, PE01, ..., PE33, as partly shown in Fig. 4. The task of the block is to compute the matrix product of two (complex) polynomial matrices. This task is split into two operations: computation of the real component ($reC_{fft}$)
and the imaginary component \((imC_{fft})\) of the matrix product in the Fourier domain. In each operation, the two matching elements of the matrix, from the sequence of FFT transformed matrices (from the FFT/IFFT block), are multiplied together. These two operations are executed independently in each PE within two consecutive cycles.

The process in this block is initiated by simultaneously accessing blocks MemI1, MemI2, MemI3, MemI4, MemI5, MemI6, MemI7, and MemI8 to retrieve the stored transformed matrix sequences. The acquired complex data are then shifted into the systolic array in a sequential manner, while adopting the following scheme: transformed sequences for the polynomial elements on each row of the multiplier matrix are simultaneously shifted into the corresponding row of the systolic array, whereas transformed sequences for the polynomial elements on each column of the multiplicand matrix are simultaneously shifted into the corresponding column of the systolic array. Depending on the current phase of the operating cycle, either real or imaginary components of these sequences for the multiplier and multiplicand matrices are shifted into the systolic array. During the first phase of the first cycle, real components of all sequences are shifted in, whereas imaginary components of all sequences are shifted in during the second phase, where the multiplier matrix sequences are multiplied by \(-1\); the first phase of the second cycle requires the shifting of real and imaginary components of the sequences for the multiplier and multiplicand matrices, respectively, whereas, in the second phase of the second cycle, the opposite is performed.

Each PE\(_{ij}\) operating in parallel within the systolic array is designed to recursively evaluate the same sets of equations for the corresponding \(i\) and \(j\) index values over the nested loops in Algorithm 1. These PEs work independently from each other, and work in parallel as the data stream enters the PE during each cycle. Referring to Algorithm 1, at the end of the two consecutive cycles, each PE\(_{ij}\) will contain, respectively, the corresponding set of \(reC_{fft}[i, j, t]\), (i.e., real matrix-product components) and \(imC_{fft}[i, j, t]\), (i.e., imaginary matrix-product components) in one of its local memories for \(t = 1, 2, \ldots, N\). These components are then shifted out and correspondingly stored into two dual-port memories included in blocks MemO1, MemO2, MemO3, and MemO4, as explained in the following.

A number of PEs coupled with each other form four sets of PEs to facilitate the shifting out process. The first set includes only PE00; the second set includes PE01, PE11, and PE10; the third set includes PE02, PE12, PE22, PE21, and PE20; and the last and largest set includes PE03, PE13, PE23, PE33, PE32, PE31, and PE30. Each of these four sets independently initiate the shifting of data contained within the linked PEs, as a whole, into one of the designated memory blocks mentioned above. This happens when the cycle ends within each PE constituting the set. Note that the first dual-port memory within the memory blocks is populated with the real values computed during the first cycle in the systolic array, whereas the second one is utilized for storing the imaginary values computed during the second cycle. Furthermore, interactions within the systolic array are not centrally regulated by an external control unit, but distributively by independent FSMs present in each PE of the array. The inner structure and functionality of the PEs are detailed in the following.

3) PEs: The inner structure of identical PEs is shown in Fig. 5. It is mainly composed of a addressable shift register, a multiplier, an adder, a multiplexer, and two FIFO blocks, (i.e. FIFOa and FIFOb), as well as a block for controlling the operations within the specific PE (i.e., ControlUnit block), which contains an FSM designed for the purpose. PEs also contain a number of pipeline registers, simple logic gates, and other miscellaneous blocks.

FIFOa and FIFOb within PEs buffer either real or imaginary components of the transformed sequences for the multiplier and multiplicand matrices, respectively, contingent on the current phase of the current operation cycle. During the first and second phases of the first cycle, these buffers are
This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Fig. 5. PE structure.

simultaneously accessed, and then the read data are multiply accumulated in accordance with the respective equations (refer to Algorithm 1) of

\[
\text{reCfft}[i, j, t] = \text{reCfft}[i, j, t] + \text{reHfft}[i, k, t] \\
\times \text{reRfft}[k, j, t]
\]  

(15)

and

\[
\text{reCfft}[i, j, t] = \text{reCfft}[i, j, t] - \text{imHfft}[i, k, t] \\
\times \text{imRfft}[k, j, t]
\]  

(16)

for \( t = 1, 2, \ldots, N \) and \( k = 1, 2, \ldots, p \), by the multiplier and adder (Fig. 5); the addressable shift register is meanwhile used as an accumulator that continuously shifts the output of the adder in through the multiplexer and shifts it out into one input port of the adder through a delay element. Eventually, the addressable shift register will have, at the end of the cycle, the set of real components of the matrix products, (i.e., \( \text{reCfft} \)) at the same index point. The addressable shift registers in the same PE set gets connected to each other through the \( \text{shift}_o \) and \( \text{shift}_i \) ports via multiplexers at the termination of a cycle. This enables the outshifting of data contained within the addressable shift registers of PEs.

V. RESULTS

The architecture for computing the PMM, whose top-level block diagram is shown in Fig. 6, was implemented on the Virtex-5 OpenSPARC evaluation platform [26], which is a versatile general-purpose development board hosting the Xilinx Virtex-5 XC5VLX110T FPGA chip [27]. As mentioned before, our design was developed using the Xilinx system generator tool, which was then synthesized, mapped, placed, and routed by the Xilinx ISE 14.5 tool. FPGA-in-the-loop hardware cosimulation method was utilized to verify the correct operation of our FPGA design.

To demonstrate the performance of the proposed PMM algorithm implementation, we present the results from the simulation scenario defined in Section V-A. Simulations on the hardware accuracy of the utilized fast MIMO convolution technique are elaborated in Section V-B, whereas the timing performance of the hardware design and the amount of resources consumed in the FPGA chip are presented in Section V-C.

A. Application to Signal Processing

In [19], a hardware implementation of the SBR2 algorithm was successfully applied to the problem of diagonalizing para-Hermitian polynomial matrices in the context of
broadband SASP. In this paper, we consider using the proposed FPGA architecture with the SBR2P algorithm from [19] to realize the polynomial matrix diagonalization in (7) and the filtering operation in (9).

Consider that \( q \) statistically independent broadband source signals are received by an array of \( p \) sensors, where \( (p \geq q) \) over a convolutive channel, corrupted by additive white Gaussian noise. Assuming the received signals \( x(z) \) have zero mean, the corresponding CSD matrix, \( \mathbf{R}(z) \), is given by (2). As a result of the mixing process, the received signals will generally be correlated over multiple time lags, and so \( \mathbf{R}(z) \) will not be diagonal [4]. In practice, an estimate of the CSD matrix, namely \( \hat{\mathbf{R}}(z) \), is obtained using a sufficient number of samples of the received data [4], [19].

As reported in [19], the SBR2P algorithm can be used to generate a diagonalizing PU polynomial matrix \( \mathbf{H}(z) \) for the estimate \( \hat{\mathbf{R}}(z) \). In the following, we use this PU matrix \( \mathbf{H}(z) \) to perform the factorization in (7). Since we are working with an estimate of \( \mathbf{R}(z) \), the transformation produces an approximately diagonal CSD matrix, \( \hat{\mathbf{S}}(z) \). The same \( \mathbf{H}(z) \) is also applied to the (strongly correlated) received signals \( x(z) \), according to (9), to produce the transformed sequence \( \hat{v}(z) \). The signals \( \hat{v}(z) \) are a good approximation to \( v(z) \) in (9), and are strongly decorrelated, to a good approximation [4].

In the following experiments, we consider that there are \( p = 4 \) received signals, a number of samples from which were used to estimate the \( 4 \times 4 \) CSD matrix \( \mathbf{R}(z) \). The performance of our FPGA-implemented fast MIMO convolution algorithm is evaluated for different FFT lengths \( N \in \{64, 128, 256, 512\} \). The SBR2P algorithm was set up as described in [19] and applied to \( \hat{\mathbf{R}}(z) \), which generated \( \mathbf{H}(z) \). The received signals are modeled as the output of innovation filters excited by uncorrelated noise. An ensemble of 150 moving average systems of order five is utilized, whereby the coefficients of sample systems are drawn from independent and identically distributed Gaussian processes of unit variance and zero mean.

The excitation of the innovation filters is formed by uncorrelated zero mean and unit variance quaternary phase shift keying sequences.

### B. Hardware Accuracy

In this section, we investigate the validity of the proposed FPGA architecture for PMM in terms of its accuracy, as compared with a double-precision version of time-domain MIMO convolution running in MATLAB. To this end, we use the mean squared error (MSE) to quantify the difference between the result yielded by our architecture and that obtained from MATLAB. The MSE is the best suited metric for capturing the variance of the error for the types of processes used in our experiments [28]. Let \( S_h w(z) \) and \( S^w(z) \) denote the approximately diagonalized CSD matrix, \( \hat{\mathbf{S}}(z) \), obtained from our hardware solution and the MATLAB software solution, respectively. The MSE in the result produced by our FPGA architecture is

\[
\eta_M = \frac{1}{Np^2} \sum_{i=1}^{p} \sum_{k=1}^{p} \sum_{r=1}^{p} |s^w_{ik} - s^{hw}_{ik}|^2
\]

(19)

where \( s^w_{ik} \) and \( s^{hw}_{ik} \) are elements from \( S^w(z) \) and \( S^{hw}(z) \), respectively. The error in the result produced by our architecture is

\[
\eta_V = \frac{1}{Np^2} \sum_{i=1}^{p} \sum_{r=1}^{p} |v_i^w - v_i^{hw}|^2
\]

(20)

where \( v_i^w \) and \( v_i^{hw} \) are elements from \( v^w(z) \) and \( v^{hw}(z) \), respectively.

The ensemble average of \( \eta_M \) and \( \eta_V \) as a function of FFT length is shown in Fig. 7. As can be seen, the error in the hardware solution reduces as the FFT length increases.
This trend is, however, negligible considering that the absolute errors are very low, of the order of $10^{-13}$. Notice that the error for matrix–matrix PMM ($\eta_{\text{MM}}$), denoted by the dashed curve, is greater than that for matrix–vector PMM ($\eta_{\text{MV}}$). This is ultimately due to hardware-related small roundoff errors, which amplify the sample noise present in the estimated space–time covariance matrix. The error, however, subsides markedly as the number of data samples, (i.e., FFT length $N$) increases.

C. Hardware Timing Performance and Resource Utilization

To assess the complexity of calculating the PMMs, the ensemble was simulated on a PC with 2.67-GHz Intel core i5 processor and 8-GB RAM running a 64-bit operating system. In Fig. 8, we show the average CPU time taken by the proposed algorithm running in MATLAB versus FFT length for matrix–matrix PMM (solid curve), as in (14), and the MIMO filtering (dashed curve), as in (13). Also included are the MATLAB execution times required by the equivalent time-domain MIMO convolution routines for comparison. As expected, the proposed algorithm for matrix–matrix PMM is faster than the time-domain counterpart. A striking result is the upward trend of the curves for the proposed algorithm. For example, the proposed method is 15 times faster at computing matrix–matrix PMM than time-domain MIMO convolution, for $N = 64$. However, there is, on average, a sixfold drop in this speedup, for $N = 512$. Notice that, for large $N$, the CPU time taken by our algorithm for matrix–vector PMM is greater than that for time-domain MIMO convolution, the reason for which is discussed in Section II-D. The short run-times achieved by our algorithm on PC demonstrate its applicability to real-time practical problems.

In Table I, we give the fixed total number of clock cycles required by our FPGA architecture, for $N \in \{64, 128, 256, 512\}$, to complete the computation of the polynomial matrix product. Table I also presents the effective number of cycles for each $N$, which are the number of cycles after which a new pair of input polynomial matrices can be fed into the architecture while the process for the previous pair is ongoing. As can be observed, the number of clock cycles required by our FPGA architecture proportionally increases as $N$ is doubled.

The fixed time consumed by the FPGA architecture versus $N$ is given in Tables II and III, which is calculated based on the total number of clock cycles required and an achievable operating frequency of 100 MHz. The speed of the proposed FPGA architecture is compared with the equivalent MATLAB code execution on PC. Tables II and III also show the average CPU times of the MATLAB executions, extracted from Fig. 8, for all settings of $N$. The speedup achieved by our architecture over MATLAB execution is more than an order of magnitude, which again demonstrates the suitability of our architecture for real-time applications.

Table V presents the maximum throughput attainable by our architecture for all $N$ values, based on the effective number
TABLE IV
FPGA RESOURCE UTILIZATIONS FOR THE PROPOSED ARCHITECTURE

<table>
<thead>
<tr>
<th></th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N = 64</td>
<td>N = 128</td>
<td></td>
</tr>
<tr>
<td>No. Slice Registers</td>
<td>19,349</td>
<td>25,847</td>
<td>27%</td>
</tr>
<tr>
<td>No. Slice LUTs</td>
<td>20,863</td>
<td>27,906</td>
<td>30%</td>
</tr>
<tr>
<td>No. Used as Logic</td>
<td>16,814</td>
<td>22,191</td>
<td>24%</td>
</tr>
<tr>
<td>No. Used as Shift Reg.</td>
<td>3,676</td>
<td>4,988</td>
<td>20%</td>
</tr>
<tr>
<td>No. Occupied Slices</td>
<td>7,226</td>
<td>9,323</td>
<td>41%</td>
</tr>
<tr>
<td>No. BlockRAM/FIFO</td>
<td>63</td>
<td>65</td>
<td>42%</td>
</tr>
<tr>
<td>No. Using BlockRAM Only</td>
<td>31</td>
<td>33</td>
<td>49%</td>
</tr>
<tr>
<td>No. Using FIFO Only</td>
<td>32</td>
<td>32</td>
<td>51%</td>
</tr>
<tr>
<td>Total Memory Used</td>
<td>2,160</td>
<td>2,304</td>
<td>40%</td>
</tr>
<tr>
<td>No. DSP48E</td>
<td>64</td>
<td>64</td>
<td>100%</td>
</tr>
</tbody>
</table>

TABLE V
THROUGHPUT ATTAINABLE BY OUR ARCHITECTURE

<table>
<thead>
<tr>
<th>N</th>
<th>Throughput (MW/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>68.20</td>
</tr>
<tr>
<td>128</td>
<td>69.63</td>
</tr>
<tr>
<td>256</td>
<td>70.21</td>
</tr>
<tr>
<td>512</td>
<td>70.62</td>
</tr>
</tbody>
</table>

of required clock cycles, in megaword per second, counting in both real and imaginary component words of the polynomial matrix products. Notice that changes in N have a little effect on the throughput, which is due to the highly pipelined nature of the proposed architecture.

Finally, Table IV shows the utilization of resources for the proposed architecture in Fig. 6, implemented on a Xilinx Virtex-5 XC5VLX110T FPGA chip, for the cases of N = 64 and N = 128. We see that resource utilization is less than 55%, except for the DSP48E slices that are fully utilized to implement the multipliers in each PE (Fig. 5) of the systolic array within the design. Note that DSP48E usage was chosen for architectural speed optimization. It is also interesting to observe that an increase in N reasonably affects the slice register and lookup table counts due to the expansion of FFT blocks, whereas BlockRAM and FIFOs show a little change in utilization rates. Conversely, an increase in data bit width results in an increased number of DSP48 slices, which is therefore not possible with the currently utilized FPGA chip.

VI. CONCLUSION

In this paper, we have presented a novel FPGA architecture for the implementation of an algorithm for computing PMM. The proposed architecture, which is the first hardware solution to this type of problem, is based on the application of the fast convolution technique to MIMO systems, which exploits the FFT. A major contribution of this paper is the introduction of a highly pipelined partly systolic architecture for hardware realization of fast MIMO convolution for PMM. The proposed architecture uses limited FPGA resources and has a little dependency on the order of the input polynomial matrices, which makes for a scalable design. Specifically, only the sizes of the block RAMs and built-in FIFOs, as well as the transform length of the FFT blocks, need to be modified to account for changes in the input matrix order. Furthermore, the FFT blocks’ transform length is run-time configurable, enhancing both the scalability and adaptability.

The data widths used in the current implementation are constrained by the limited number of DSP48E slices available in the utilized FPGA chip. However, if a larger FPGA specialized for DSP implementations is employed, the data widths can be increased to achieve higher accuracy. In this case, a thorough investigation into determining the optimal data width versus multiplication accuracy would be required, taking into account the potential increase in the utilization of resources as well.

This paper has focused on the relevance of the proposed architecture to signal processing in the context of strong decorrelation. However, considering the applications reviewed in Section I, the advantages of the new architecture, in conjunction with a PEVD architecture, is expected to make a wider-ranging impact. For example, when employing subspace-based methods, such as [5]–[7], PMM is an essential stage in the separation of the broadband subspaces. It is also implicit in the efficient calculation and application of subspace projections, such as the generator and parity check polynomial matrices in [9], for channel coding. All these strongly suggest that our design can be applied to solve a plethora of real problems.

Currently, our design is able to support up to four signals for broadband SASP. However, with the goal of supporting any number of signals using the same core of systolic array with $4 \times 4$ PEs, we envisage that one can devise an effective and efficient algorithm, which will enable the repetitive operation of the array on a different set of sub-matrices stripped from the input polynomial matrices in each iteration, allowing the architecture to perform PMM of any size matrices in a certain number of iterations. This approach would slightly, if at all, increase the FPGA resource consumption, whereas the achievable speed-up figures will possibly degrade.

ACKNOWLEDGMENT

The authors would like to thank Prof. J. G. McWhirter of Cardiff University and Dr. S. Weiss of the University of Strathclyde for a number of very helpful discussions. The
Paderborn Center for Parallel Computing (PC²) is gratefully acknowledged for the partial financial support. They would also like to thank the anonymous reviewers for their careful reading and insightful comments.

REFERENCES


Soydan Redif (M’09–SM’12) received the B.Eng. (first class Hons.) degree in electronic engineering from Middlesex University, London, U.K., and the Ph.D. degree in electrical and electronic engineering from the University of Southampton, Southampton, U.K., in 1998 and 2006, respectively.

He joined Defense Evaluation and Research Agency, U.K., involved initially in airborne SHF SATCOM systems in 1999. From 2000 to 2007, he was with the Malven Signal Processing Group, QinetiQ, Malvern, U.K., where his research focused on adaptive signal processing algorithms. From 2008 to 2011, he was an Assistant Professor with Near East University, Nicosia, Cyprus. In 2011, he was a Visiting Lecturer with the European University of Lefke (EUL), Gemikonagi, Cyprus. He has been at EUL since 2011, and is currently the Head of the Department of Electrical and Electronics Engineering. His current research interests include adaptive signal processing, broadband sensor arrays, convolutive blind signal separation, polynomial matrix techniques, and power systems.

Dr. Redif is a U.K. Chartered Engineer and a member of the IET. He was the recipient of the IEE Award for Outstanding Academic Achievement in 1998.

Server Kasap (M’10) received the B.Sc. (Hons.) degree in electrical and electronic engineering from Middle East Technical University, Ankara, Turkey, and the M.Sc.(Distinction) and Ph.D. degrees in electronic engineering from the System Level Integration Research Group, University of Edinburgh, Edinburgh, U.K., in 2006, 2007, and 2010, respectively.

He joined the Computer Engineering Research Group at the University of Paderborn, Paderborn, Germany, as a Post-Doctoral Research Associate in 2013. He is also affiliated with the Paderborn Center for Parallel Computing. From 2011 to 2013, he was an Assistant Professor with the Electrical and Electronics Engineering (EEE) Department, European University of Lefke, Gemikonagi, Cyprus. In 2011, he was a Visiting Lecturer with the EEE Department, Cyprus International University, Nicosia, Cyprus. His current research interests include reconfigurable dataflow computing for finite-difference time-domain simulations, FPGA hardware design and implementation for digital signal processing applications, and high-performance scientific computing in general.