

4X4 POLYNOMIAL MATRIX MULTIPLICATION ON FPGA

R.DHANA PRIYA¹, D.DEVI²
PG SCHOLAR, ASSISTANT PROFESSOR
DEPARTMENT OF ECE

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY^{1,2}
Priyar473@gmail.com¹, devi@skcet.ac.in²

ABSTRACT

In this paper, the polynomial matrix multiplication (PMM) of polynomial vectors and/or polynomial matrices has been introduced. This method provides an improvement of the fast convolution technique to multiple-input multiple-output systems (MIMO). It is devoted to the hardware implementation of PMM. Hardware implementation of this method is achieved via partly systolic, field-programmable gate array (FPGA) with highly pipelined architecture. The architecture, which is scalable in terms of the order of the input polynomial matrices, Xilinx system generator tool has been used for designing. The application to sensor array signal processing strong decorrelation is highlighted. The results are presented to verify the capability and accuracy of the architecture. The result proved that the proposed solution gives low execution times and the number FPGA resource is less.

Index Terms: Field-programmable gate array (FPGA), SBR2P, polynomial matrix multiplication (PMM), polynomial matrix computations, Xilinx system generator tool.

1. INTRODUCTION

Polynomial matrices have been used for many years in the area of control. They play an important role in the realization of multivariable transfer functions associated with multiple-input multiple-output (MIMO) systems. Few years back they have become more widely used in the context of digital signal processing (DSP) and communications [21]. Broadband subspace decomposition [12], Typical areas of application include broadband adaptive sensor array processing [22], [23], MIMO communication channels [12] [25], and digital filter banks for sub band coding [24] or data compression [23].

A polynomial matrix is simply a matrix whose elements are polynomials. It may be viewed

equivalently, as a polynomial with matrix coefficients. In this paper, we will use the term polynomial to include Laurent polynomials which can include negative powers of the indeterminate variable. We denote a polynomial matrix in the indeterminate variable.

Numerical procedures have previously been developed for a range of polynomial matrix factorization and reduction operations such as the Smith–McMillan decomposition [22]. To date, however, very little attention seems to have been devoted to polynomial matrix techniques equivalent to the eigenvalue decomposition (EVD) or singular value decomposition (SVD) for conventional matrices with scalar elements [11]. The development and implementation of such a technique is the subject of this paper. The Eigen value decomposition of conventional Hermitian matrices plays a major role in DSP. For example, it is at the heart of the Karhunen–Loeve transform for optimal data compaction.

II. FAST FOURIER TRANSFORM

A fast Fourier transform (FFT) algorithm computes the discrete Fourier transform (DFT) of a sequence, or its inverse. Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. As a result, it manages to reduce the complexity of computing the DFT from $O(n^2)$, which arises if one simply applies the definition of DFT, to $O(n \log n)$, where n is the data size. Fast Fourier transforms are widely used for many applications in engineering, science, and mathematics.

A. Cooley–Tukey algorithm

The best known use of the Cooley–Tukey algorithm is to divide the transform into two pieces of size $N/2$ at each step, and is therefore limited to power-of-two sizes, but any factorization can be used in general (as was known to both Gauss and Cooley/Tukey). These are called the radix-

2 and mixed radix cases, respectively (and other variants such as the split-radix FFT have their own names as well). Although the basic idea is recursive, most traditional implementations rearrange the algorithm to avoid explicit recursion. Also, because the Cooley–Tukey algorithm breaks the DFT into smaller DFTs, it can be combined arbitrarily with any other algorithm for the DFT.

B. COMPUTATIONAL ISSUES

1. BOUNDS ON COMPLEXITY AND OPERATION COUNTS

The interest is to prove lower bounds on the complexity and exact operation counts of fast Fourier transforms, and many open problems remain. It is not even rigorously proved whether DFTs truly require $\Omega(N \log(N))$ (i.e., order $N \log(N)$ or greater) operations, even for the simple case of power of two sizes. In particular, the count of arithmetic operations is usually the focus of such questions, although actual performance on modern-day computers is determined by many other factors such as cache or CPU pipeline optimization. Christo Ananth et al. [9] proposed a system, Low Voltage Differential Signaling (LVDS) is a way to communicate data using a very low voltage swing (about 350mV) differentially over two PCB traces. It deals about the analysis and design of a low power, low noise and high speed comparator for a high performance Low Voltage Differential Signaling (LVDS) Receiver. The circuit of a Conventional Double Tail Latch Type Comparator is modified for the purpose of low-power and low noise operation even in small supply voltages. The circuit is simulated with 2V DC supply voltage, 350mV 500MHz sinusoidal input and 1GHz clock frequency. LVDS Receiver using comparator as its second stage is designed and simulated in Cadence Virtuoso Analog Design Environment using GPDK 180nm. By this design, the power dissipation, delay and noise can be reduced.

2. ACCURACY

Even the "exact" FFT algorithms have errors when finite-precision floating-point arithmetic is used, but these errors are typically quite small; most FFT algorithms, e.g. Cooley–Tukey, have excellent numerical properties as a consequence of the pair wise structure of the algorithms. The upper bound on the relative error for the Cooley–Tukey algorithm is $O(\epsilon \log N)$, compared to $O(\epsilon N^{3/2})$ for the naïve DFT formula, where ϵ is the machine floating-point relative precision. In fact, the root

mean square (rms) errors are much better than these upper bounds, being only $O(\epsilon \sqrt{\log N})$ for Cooley–Tukey and $O(\epsilon \sqrt{N})$ for the naïve DFT (Schatzman, 1996). In fixed-point arithmetic, the finite-precision errors accumulated by FFT algorithms are worse, with RMS errors growing as $O(\sqrt{N})$ for the Cooley–Tukey algorithm (Welch, 1969). Moreover, even achieving this accuracy requires careful attention to scaling to minimize loss of precision, and fixed-point FFT algorithms involve rescaling at each intermediate stage of decompositions like Cooley–Tukey.

3. MULTIDIMENSIONAL FFTs

As defined in the multidimensional DFT article, the multidimensional DFT

$$X_K = \sum_{n=0}^{N-1} e^{-2\pi i K(n/N)x_n} \quad (1)$$

transforms an array x_n with a d -dimensional vector of indices $n = (n_1, \dots, n_d)$ by a set of d nested summations (over $n_j = 0 \dots N_j - 1$ for each j), where the division n/N , defined as $n/N = (n_1/N_1, \dots, n_d/N_d)$, is performed element-wise. Equivalently, it is the composition of a sequence of d sets of one dimensional DFTs, performed along one dimension at a time (in any order). This compositional viewpoint immediately provides the simplest and most common multidimensional DFT algorithm, known as the row-column algorithm (after the two-dimensional case, below). That is, one simply performs a sequence of one dimensional FFTs (by any of the above algorithms) first you transform along the n_1 dimension, then along the n_2 dimension, and so on. This method is easily shown to have the usual $O(M \log(N))$ complexity, where $N = N_1 \cdot N_2 \dots N_d$ is the total number of data points transformed. In particular, there are N/N_1 transforms of size N_1 , etcetera, so the complexity of the sequence of FFTs is

$$\frac{N}{N_1} O(N_1 \log N_1) + \dots + \frac{N}{N_d} O(N_d \log N_d) = O(N [\log N_1 + \dots + \log N_d]) = O(N \log N) \quad (2)$$

In two dimensions, the x_k can be viewed as an $n_1 \times n_2$ matrix, and this algorithm corresponds to first performing the FFT of all the rows (resp. columns), grouping the resulting transformed rows (resp. columns) together as another $n_1 \times n_2$ matrix, and then performing the FFT on each of the columns (resp. rows) of this second matrix, and similarly grouping the results into the final result matrix.

In more than two dimensions, it is often advantageous for cache locality to group the dimensions recursively. For example, a three-dimensional FFT might first perform two-dimensional FFTs of each planar "slice" for each fixed n_1 , and then perform the one-dimensional FFTs along the n_1 direction. More generally, an asymptotically optimal cache-oblivious algorithm consists of recursively dividing the dimensions into two groups ($n_1 \dots n_d/2$) and ($n_d/2+1 \dots n_d$) that are transformed recursively (rounding if d is not even) (see Frigo and Johnson, 2005).

Still, this remains a straightforward variation of the row-column algorithm that ultimately requires only a one-dimensional FFT algorithm as the base case, and still has $O(N \log(N))$ complexity.

C.A SUMMARY OF SOME BASIC FFT CONCEPTS

1. REAL DATA PROCESSED BY A REAL FFT

If real data is being processed by a real fft algorithm then the complex output spectrum extends from $n = 0$ to $n = N/2$ (so, $N/2 + 1$ data pairs are present, in total, of real/imaginary numbers). No data exists beyond this point (there are no frequency components with a negative frequency to deal with in this case).

2.REAL DATA PROCESSED BY A COMPLEX FFT

When a complex fft algorithm is being used, the complex frequency data may fill the whole range from $n = 0$ to $n = N - 1$. As with the case for real data, the frequency bins for $n=0$ to $n= N/2$ are for positive frequency content, but now negative frequency data fills the bins from $n = N/2$ to $N-1$. Because of symmetry in the frequency data, half of it is superfluous and can be discarded with no overall loss of information. This may be advantageous in that subsequent calculations are simplified. However, an inverse complex FFT cannot be used on the remaining frequency data, in order to recover a time domain waveform. Either a real IFFT has to be used for this purpose, or the missing data has to be reconstituted first.

III. ALGORITHM FOR PMM

This section provides an algorithm for multiplication of polynomial matrices of the MIMO convolution technique, for which FPGA architecture

is described. The algorithm proceeds by taking the FFT as an input polynomial matrices or vectors, and proceeds with the conventional matrix multiplication of FFT matrices (or vectors). Finally the IFFT of matrix products is taken. The proposed Matrix-Matrix PMM algorithm shows computing the FFT of the input polynomial matrices, the matrix sequences produce of their real and imaginary components. The process ends with the IFFT of the matrix sequence resulting the output polynomial matrix. Note that the algorithm can also be used to polynomial matrix-vector product computations, since a polynomial vector is a single-column or single-row polynomial matrix.

IV. PROPOSED ARCHITECTURE

A. HARDWARE IMPLEMENTATION OF POLYNOMIAL MATRIX

In this section, we describe the hardware implementation of the fast MIMO convolution technique of PMM in the following steps.

- 1) Initially, the FFT of the $p \times p$ two input polynomial matrices is computed. Row-by-row input for the multiplier polynomial matrix, and on a column by column input for the multiplicand. The resulting matrix sequences, with real and imaginary components, are stored in similar sets of memory blocks allocated to each transformed polynomial matrix. Each memory block consists of two dual-port memories for imaginary and real components. All calculations of this step are done by two FFT blocks operating in parallel.
- 2) The next step provides conventional multiplication of the similar elements of the two matrixes obtained, is explained in this section later. In this step, it takes two phases to complete during which an exclusive equation is produced for the computation of the complex matrix products. The results are thus stored into a second set of similar memory blocks; each consists of 2 dual-port memories. For this step, the first dual-port memory within each memory block is used. These computations are produced by a 2-D systolic array [24] comprised of $p \times p$ processing elements (PEs) especially designed for this purpose.
- 3) The third step is similar to step 2; however, during this time, the similar elements of the resulting two matrix sequences from the step1 are

multiplied to obtain the imaginary components of the matrix product. Therefore, in this step, two new exclusive equations are calculated respectively. Further, the results of this step are stored in the same memory block set as in step 2, but the second dual-port memories are utilized in this case. The implementations of this step are also accomplished by the $p \times p$ systolic array as mentioned above after the completion of step 2.

Finally, the algorithm ends with the IFFT of the contents of the memory blocks produced by the outcomes of steps 2 and 3 to produce the output polynomial matrix product. The IFFT is applied to two memory blocks at a time in a specific order. These computations are performed by the two parallel FFT blocks as in step 1; however, this time, they are computed in inverse mode. The above four steps of the proposed hardware algorithm are sequentially calculated in a pipelined manner for the input polynomial matrices to obtain the maximum throughput value.

B. POLYNOMIAL MATRIX ARCHITECTURE

In this section, the highly pipelined partly systolic architecture implementing the hardware algorithm detailed in Section IV-A for 4×4 polynomial matrices, (i.e., $p=4$). The FPGA architecture for PMM has been designed using the Xilinx system generator tool. It consists of the Data Path block that computes data processing operations with a control unit (i.e., Main_FSM block), incorporating with a finite-state machine (FSM), which regulates interaction between the Data Path block and the data itself. The Data Path block, shown in Fig. 2, consists of two sets of memory blocks and two main blocks, namely FFT_IFFT and Systolic Array blocks, sequentially implementing the four major steps of the fast MIMO convolution technique. Outputs of the FFT_IFFT block are stored in the first set of memory blocks such as MemI1, MemI2, MemI3, MemI4, MemI5, MemI6, MemI7, and MemI8.

On the other hand, results of the Systolic Array block are stored into the second set of memory blocks such as MemO1, MemO2, MemO3, and MemO4. As mentioned above, each memory block consists of two dual-port memories for storing imaginary and real values. An important aspect of the proposed architecture is it does not have any high dependency on the order of the input polynomial matrices. Only some specific parameter values as well as size of the dual-port memories, throughout the architecture need to be modified when the order of the input matrices is changed. Further, Xilinx uses fixed-point data type throughout

our architecture, where rounding and saturation have been chosen as overflow and quantization options respectively, for all arithmetical units. e.g., multipliers and adders, to improve the accuracy of the design.

1) FFT_IFFT BLOCK: The inner structure of the FFT_IFFT block is shown in Fig. 3. It mainly consists of two FFT blocks operating simultaneously, (i.e., FFT 7.1 modules [25]), which can be operated in forward or inverse mode depending on the *fwd-inv* signal. The block also consists of 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 provide it accordingly to the *xn_re* or *xn_im ports* of the ft blocks, which are for inputting real and imaginary values, respectively.

The FFT_IFFT block has two modes of operation. In the first mode, the FFT blocks perform the FFT on each polynomial element of multiplier and multiplicand matrices in parallel, respectively. In effect, the transformation is along the 3rd dimension of the polynomial matrix. Complex row vector sequences resulting from the FFT of each polynomial element multiplier matrix are stored into the MemI1, MemI2, MemI3, and MemI4 memory blocks respectively. Whereas complex column vector sequences resulting from the FFT of each polynomial element column for the multiplicand matrix are kept in the MemI5, MemI6, MemI7, and MemI7 blocks respectively in the same way.

In the second mode, an FFT block performs the IFFT of the data of MemO1, MemO2, MemO3, and MemO4 provided by the Systolic Array block, to obtain the architecture output: polynomial matrix product. The IFFT process is sequentially applied to the read complex data from the MemO1 and MemO2 blocks from the respective FFT blocks in parallel, and then the complex data from the blocks MemO3 and MemO4 are, processed by the FFT blocks respectively, at the same time. Note that multiplexers within the FFT_IFFT block architecture the data from the memory blocks to the FFT blocks are directed by multipliers, accordingly. The operations within this block are controlled by the Main FSM block.

2) SYSTOLIC ARRAY BLOCK: The Systolic Array block is at the center of our data path architecture (Fig. 1), containing a 2-D systolic array composed of 4×4 identical PEs, namely PE00, PE01, PE33, as shown in Fig. 4. The work of the block is to compute the matrix product of two (complex) polynomial matrices. This is split into two operations: computation of imaginary component

(imC_{ft}) and real component (reC_{ft}) of the matrix product in Fourier domain. In each operation, the two matching elements of the matrix, from the FFT_IFFT block, are multiplied. These two operations are computed independently in each PE within two consecutive cycles. The process in this block starts by accessing blocks MemI1, MemI2, MemI3, MemI4, MemI5, MemI6, MemI7, and MemI8 to obtain the stored transformed matrix sequences.

The complex data thus obtained are then shifted into the systolic array in a sequential manner. By adopting the following scheme: the polynomial elements on each row of the multiplier matrix are shifted into the corresponding row of the systolic array simultaneously, whereas transformed sequences for the polynomial elements on each column of the multiplicand matrix are shifted into the corresponding column of the systolic array simultaneously. Depending on the operating cycle, either the real or imaginary components of the multiplicand and multiplier matrices are transferred into the systolic array. During the 1st phase of the first cycle, real components are shifted in, whereas imaginary components are shifted in during the second phase, where the multiplier matrix are multiplied by (-1); the 1st phase of the second cycle requires the shifting of real and imaginary components of the multiplier and multiplicand matrices, respectively, whereas, the second phase of the second cycle, the inverse operation is performed.

These PEs work independent of each other, and works parallel as the data stream enters the PE during each cycle. At the end of the two consecutive cycles, each PE_{ij} respectively will contain the corresponding set of $imC_{ft}[i, j, t]$, (i.e., imaginary matrix product components) and $reC_{ft}[i, j, t]$, (i.e., real matrix product components) in one of the local memories for $t = 1, 2, \dots, N$. These components are shifted out and stored in two dual-port memories named as MemO1, MemO2, MemO3, and MemO4 correspondingly.

The four sets of PEs are used to facilitate the shifting process. The first set includes only PE00; the second set includes PE01, PE10, and PE11; the third set includes PE02, PE12, PE21, PE22, and PE20; and the last and largest set includes PE03, PE13, PE23, PE31, PE32, PE33, and PE30. Each of these four sets independently used for shifting the data within the linked PEs, at last shifts into the designated memory blocks mentioned above. This happens when each PE constituting the set ends within the cycle. Note that the first dual port memory within the memory blocks is filled with the real values produced during the first cycle in the systolic array, whereas the second one is utilized for utilizing the imaginary values computed during the 2nd Cycle.

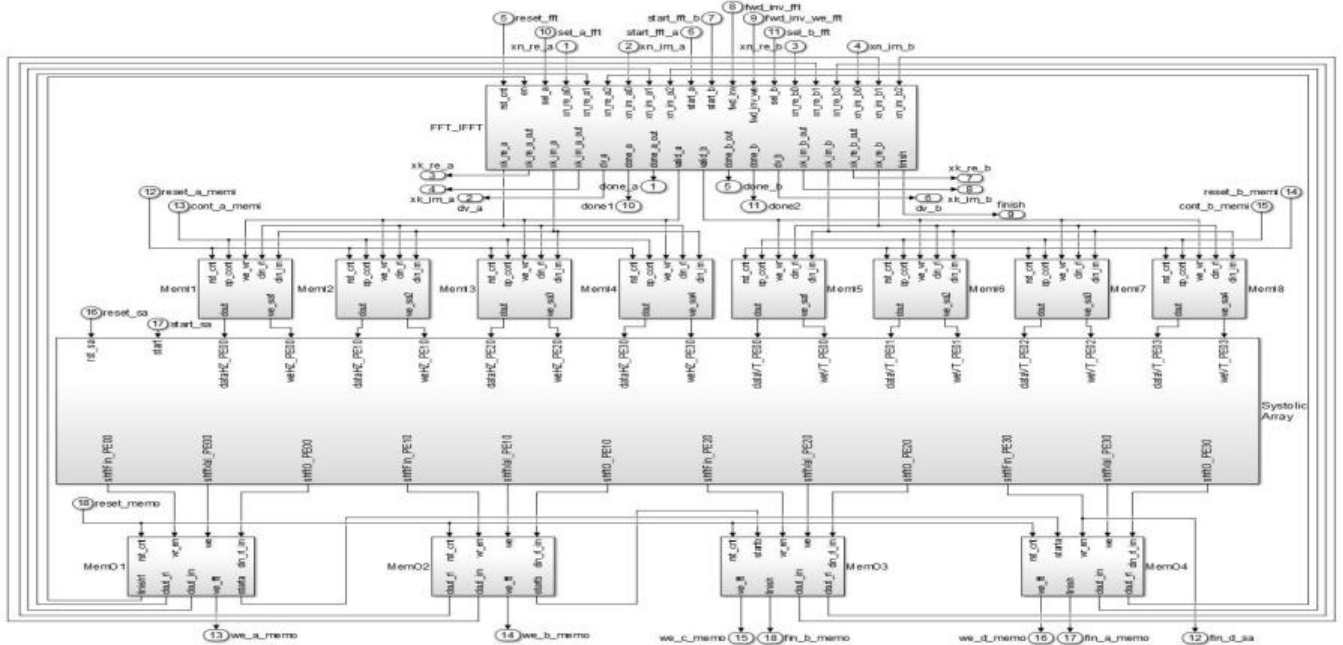


FIG.1.ARCHITECTURE OF THE PROPOSED METHOD

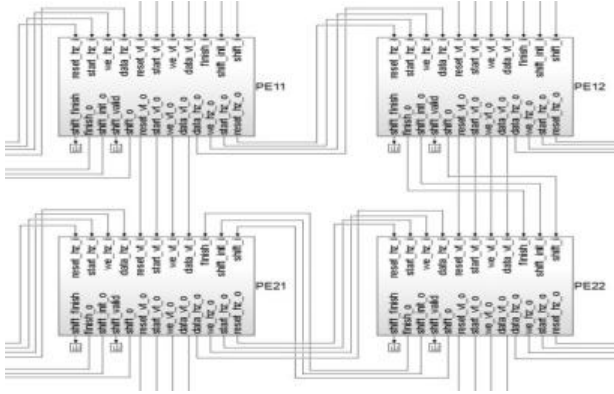


FIG.2. SYSTOLIC ARRAY BLOCK

3) PES: In Fig. 5 the inner structure of identical PEs is shown. It mostly consists of an addressable shift register, an adder, a multiplier, a multiplexer, two FIFO blocks, (i.e., FIFOa and FIFOb), and Control Unit block within the specific PE. PEs also contains a number of pipeline registers, simple logic gates, and other miscellaneous blocks. FIFOa and FIFOb within PEs used to transmit either real or imaginary components of the multiplier and multiplicand matrices respectively within current operation cycle, the buffers are simultaneously accessed, and then the read data are multiply and accumulated in accordance with the respective equations of

$$\mathbf{reCf f t [i, j, t] = reCf f t [i, j, t] + reHf f t [i, k, t] \times reRf f t [k, j, t]} \quad (15)$$

$$\mathbf{reCf f t [i, j, t] = reCf f t [i, j, t] - imHf f t [i, k, t] \times imRf f t [k, j, t]} \quad (16)$$

For $t = 1, 2, \dots, N$ and $k = 1, 2, \dots, P$, by the multiplier and adder (Fig. 5); meanwhile the addressable shift register is 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 the matrix products, (i.e., $\mathbf{reCf f t}$) for the particular indices i, j indicated by the position of the PE within the systolic array blocks. The 2nd cycle of operation is similar to the 1st cycle; however, the equations of

$$\mathbf{imCf f t [i, j, t] = imCf f t [i, j, t] + reHf f t [i, k, t] \times imRf f t [k, j, t]} \quad (17)$$

$$\mathbf{imCf f t [i, j, t] = imCf f t [i, j, t] + imHf f t [i, k, t] \times reRf f t [k, j, t]} \quad (18)$$

Are implemented for $t = 1, 2, \dots, N$ and $k = 1, 2, \dots, P$. During the 1st and 2nd phases, respectively, to compute the set of imaginary components of the matrix products, (i.e., $\mathbf{imCf f t}$) of the same index point. The addressable shift registers in the same PE set gets connected to each other through the Shafto cycle. This enables the out shifting of data contained within the addressable shift registers of PEs

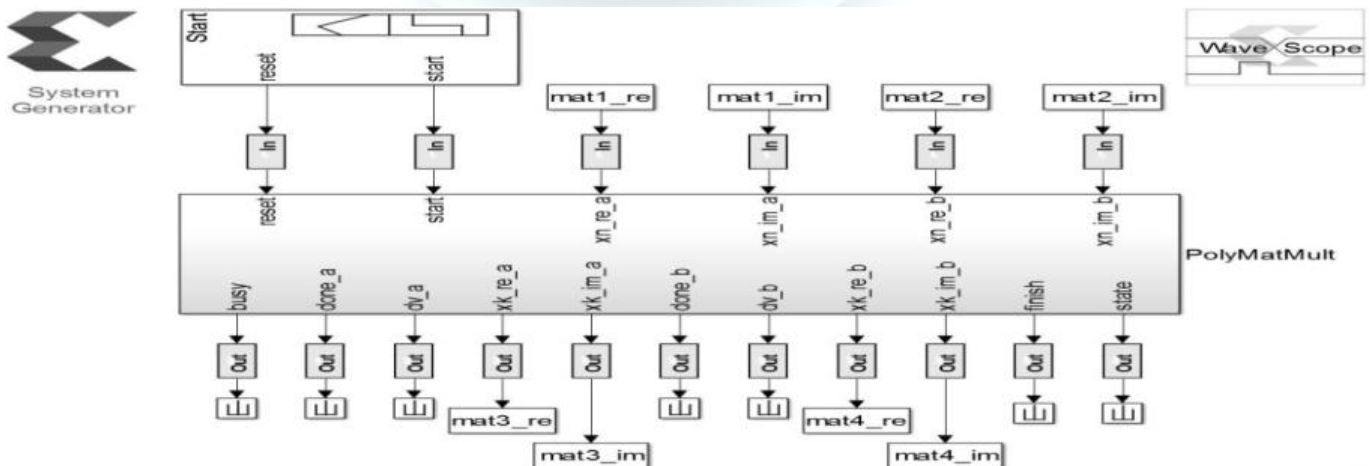


FIG.3.TOP-LEVEL BLOCK DIAGRAM FOR PMM

V. RESULTS

The architecture for performing the PMM, whose block diagram is shown in Fig. 3, was implemented on the Virtex-5 Open SPARC platform [26], which is a general-purpose development board with XC5VLX110T Xilinx Virtex-5 FPGA chip [27]. As described before, our architecture was developed using the Xilinx system generator tool, which was then synthesized, placed, mapped and routed by the Xilinx ISE 14.5 tool. FPGA-in-the-loop hardware co simulation method was used to verify the correct operation of the FPGA design. To show the performance of the proposed PMM algorithm implementation, we represent the results from the simulation defined in Section V-A. Simulations on the hardware accuracy of the utilized fast MIMO convolution technique are explained, whereas the amount of resources consumed in the FPGA chip and timing performance of the hardware design are presented.

A. HARDWARE TIMING PERFORMANCE AND RESOURCE UTILIZATION

To analyze the complexity of calculating the PMMs, the ensemble was simulated on a PC with 8-GB RAM running at 64-bit operating system and 2.67 GHz Intel core i5 processor. We show the average CPU access time taken by our proposed algorithm running in MATLAB versus FFT length for matrix-matrix PMM, and the MIMO filtering. Also included are the MATLAB execution times required by the equivalent time-domain MIMO convolution routines for comparison. As expected, the proposed PMM algorithm for matrix-matrix is faster than the time-domain counterpart. The result is the upward trend of the curves, for $N = 64$. However, there is, on average of six fold drop in this speedup, for $N = 512$. Note that, the CPU access time by our algorithm for matrix-vector PMM is greater than for time-domain MIMO convolution for large N . The reduced run-times achieved by our algorithm on PC shows its applicability to real-time problems.

In Table I, we demonstrated the fixed number of clock cycles required by our FPGA design, for example $N \in \{64, 128, 256, 512\}$, for the computation of the polynomial matrix product. It also shows the effective number of clock cycles for each N , which is the number of cycles after which a new set of polynomial matrices input populated into the architecture. As observed, the number of clock cycles required by our FPGA architecture increases as N is doubled. In the Tables II and III the access time consumed by the FPGA design versus N is given,

which is calculated based on the total number of clock cycles required at an operating frequency of 100 MHz the speed of the proposed FPGA design is compared with equivalent MATLAB code execution time on PC. It also shows the average CPU time execution, for all N . The speed up performance by our architecture over MATLAB execution is more than an order of magnitude, which is suitable for real-time applications. The maximum throughput achieved by our design for all N values, based on the effective number of required clock cycles, in mega word per second, in both imaginary and real component of the polynomial matrix products shown in Table VI. Notice that changes in N have a little effect on the throughput, which is due to the pipelined nature of the proposed architecture.

N	No of cycle	No of effective cycles
64	3215	3006
128	6245	5897
256	12273	11663
512	24327	23194

TABLE I
NUMBER OF CLOCK CYCLES REQUIRED BY OUR FPGA DESIGN

TABLE II
FPGA VERSUS CPU TIME REQUIREMENTS FOR THE POLYNOMIAL MATRIX-VECTOR MULTIPLICATION WITH FAST MIMO CONVOLUTION

N	CPU	FPGA	SPEED
64	430.32	32.15	13.4
128	701.83	62.45	11.2
256	1674.7	122.73	13.7
512	3169.3	243.27	13.0

TABLE III
FPGA VERSUS CPU TIME REQUIREMENTS FOR THE POLYNOMIAL MATRIX-MATRIX MULTIPLICATION WITH FAST MIMO CONVOLUTION

N	CPU	FPGA	SPEED-UP
64	927.59	32.15	28.9

128	1584.1	62.45	22.3
512	2740.2	243.27	25.4
256	5245.4	122.73	21.6

TABLE IV
LOGICAL COMPARISON

PARAMETERS	CMOS	PASS TRANSISTOR LOGIC	GATE ARRAY LOGIC
Number of Transistor	Most	More	Less
Area	Maximum	Medium	Minimum
Power	Most	More	Less
Delay	Most	More	Less
Speed	Less	Medium	High

TABLE V
FPGA RESOURCE UTILIZATIONS FOR THE PROPOSED ARCHITECTURE

	USED		AVAILABLE	UTILIZATION	
	N=64	N=128		N=64	N=128
NO.SLICE REGISTERS	19,349	25,847	69,120	27%	37%
NO.SLICE LUTS	20,847	27,907	69,120	30%	40%
NO USED AS LOGIC	16,814	22,191	69,120	24%	32%
NO.USED AS SHIFTREG	3,676	4,988	17,920	20%	27%
NO.OCCUPIED SLICES	7,226	9,323	17,920	41%	53%
NO.BLOCK RAM/FIFO	64	65	148	42%	43%
NO.USING BLOCK RAM ONLY	31	33	63,64	49%	51%
NO.USING FIFO ONLY	32	32	63,64	51%	49%
TOTAL MEMORY USED	2,160	2,304	5,328	40%	43%

TABLE VI
THROUGHPUT ATTAINABLE BY OUR ARCHITECTURE

N	THROUGHPUT(MW/s)
64	69.63

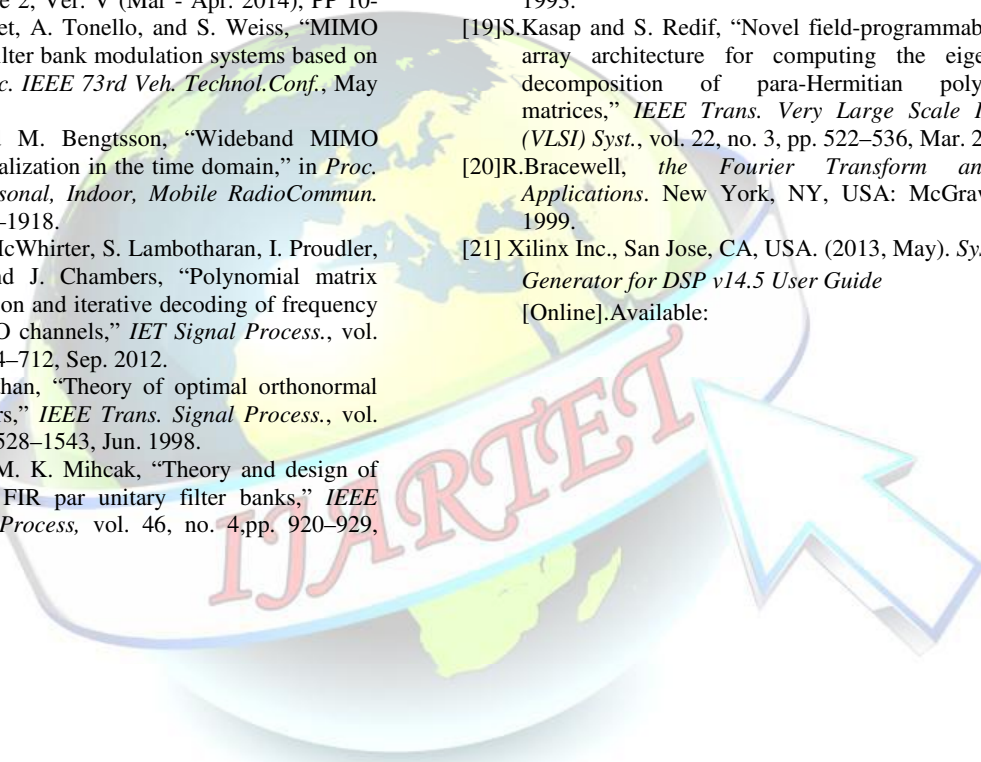
128	69.63
256	70.21
512	70.62

VI.CONCLUSION

In this paper, we have demonstrated FPGAarchitecture for the implementation of PMM. The proposed architecture is based on the application of the fast convolution technique to MIMO systems, which exploits the FFT. A main contribution of this paper is the introduction of a partly systolic highly pipelined architecture for the realization of fast MIMO convolution. The proposed architecture consumes limited FPGA resources and has a little dependency on the input polynomial matrices order, which makes for a scalable design. Specifically, built-in FIFOs and the sizes of the block RAMs as well as the transform length of the FFT blocks, should be modified for the input matrix order. Further, the FFT blocks transform length is, enhancing both the scalability and adaptability runtime configurable. The data widths used in the current implementation are reduced by the limited number of FPGA chip. However, for larger FPGA specialized for DSP implementations are employed, the data widths are increased to achieve higher accuracy. In this case, detailed investigations in determining the optimum data width versus accuracy in multiplication

REFERENCES

- [1] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD, USA: John Hopkins Univ. Press, 1996.
- [2] P. A. Regalia and P. Loubaton, "Rational subspace estimation using adaptive lossless filters," *IEEE Trans. Signal Process.*, vol. 40, no. 10, pp. 2392–2405, Oct. 1992.
- [3] R. H. Lambert, M. Joho, and H. Mathis, "Polynomial singular values For number of wideband source estimation and principal components Analysis," in *Proc. Int. Conf. Independ. Compon. Anal.*, 2001, pp. 379–383.
- [4] J. G. McWhirter, P. D. Baxter, T. Cooper, S. Redif, and J. Foster, "An EVD algorithm for para-Hermitian polynomial matrices," *IEEE Trans.Signal Process.* vol. 55, no. 5, pp. 2158–2169, May 2007.
- [5] S. Redif, J. G. McWhirter, P. Baxter, and T. Cooper "Robust broadband adaptive beam forming via polynomial eigenvalues," in *Proc. IEEE Ocean. Conf.*, Jun. 2006, pp. 1–6.

- 
- [6] S. Weiss, M. Almah, S. Lambbotharan, J.G. McWhirter, and M. Kaveh, "Broadband angle of arrival estimation methods in a polynomial matrix Decomposition framework," in *Proc. 5th IEEE Int. Workshop Comput. Adv. Multi-Sensor Adaptive Process.* Dec. 2013, pp. 1–3.
- [7] S. Redif and U. Fahrioglu, "Foetal ECG extraction using broadband signal subspace decomposition," in *Proc. IEEE Medit. Microw. Symp.* Guzelyurt, Cyprus, Jun. 2010, pp. 381–384.
- [8] S. Y. Kung, Y. Wu, and X. Zhang, "Bezout space-time precoders and equalizers for MIMO channels," *IEEE Trans. Signal Process.*, vol. 50, no. 10, pp. 2499–2541, Oct. 2002.
- [9] Christo Ananth, Bincy P Chacko, "Analysis and Design of Low Voltage Low Noise LVDS Receiver", IOSR Journal of Computer Engineering (IOSR-JCE), Volume 9, Issue 2, Ver. V (Mar - Apr. 2014), PP 10-18
- [10] N. Moret, A. Tonello, and S. Weiss, "MIMO precoding for filter bank modulation systems based on PSVD," in *Proc. IEEE 73rd Veh. Technol. Conf.*, May 2011, pp. 1–95.
- [11] R. Brandt and M. Bengtsson, "Wideband MIMO channel diagonalization in the time domain," in *Proc. Int. Symp. Personal, Indoor, Mobile Radio Commun.* 2011, pp. 1914–1918.
- [12] J. Foster, J. G. McWhirter, S. Lambbotharan, I. Proudler, M. Davies, and J. Chambers, "Polynomial matrix QRdecomposition and iterative decoding of frequency selective MIMO channels," *IET Signal Process.*, vol. 6, no. 7, pp. 704–712, Sep. 2012.
- [13] P. P. Vaidyanathan, "Theory of optimal orthonormal sub band coders," *IEEE Trans. Signal Process.*, vol. 46, no. 6, pp. 1528–1543, Jun. 1998.
- [14] P. Moulin and M. K. Mihcak, "Theory and design of signal-adapted FIR par unitary filter banks," *IEEE Trans. Signal Process.*, vol. 46, no. 4, pp. 920–929, Apr. 1998.
- [15] A. Tkachenko, "Approximate eigenvalue decomposition of para-hermitian systems through successive FIR par unitary transformations," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.* Mar. 2010, pp. 4074–4077.
- [16] S. Redif, S. Weiss, and J. G. McWhirter, "An approximate polynomial matrix eigenvalue decomposition algorithm for para-hermitian matrices," in *Proc. IEEE Int. Symp. Signal Process. Inf. Technol.*, Dec. 2011, pp. 421–425.
- [17] S. Redif, S. Weiss, and J. G. McWhirter, "Design of FIR paraunitary filter banks for subband coding using a polynomial eigenvalue decomposition," *IEEE Trans. Signal Process.*, vol. 59, no. 11, pp. 5253–5264, Nov. 2011.
- [18] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Upper Saddle River, NJ, USA: Prentice-Hall, 1993.
- [19] S. Kasap and S. Redif, "Novel field-programmable gate array architecture for computing the eigenvalue decomposition of para-Hermitian polynomial matrices," *IEEE Trans. Very Large Scale Integer. (VLSI) Syst.*, vol. 22, no. 3, pp. 522–536, Mar. 2013.
- [20] R. Bracewell, *the Fourier Transform and Its Applications*. New York, NY, USA: McGraw-Hill, 1999.
- [21] Xilinx Inc., San Jose, CA, USA. (2013, May). *System Generator for DSP v14.5 User Guide* [Online]. Available: