

A Systolic Array Architecture for Computing Time-varying Higher-order Cumulants

Mohammed AlOqeely and Abdullah Al-Shoshan

*Department of Computer Engineering, College of Computer & Information Sciences
King Saud University, P.O. Box 51178 Riyadh 11543, Saudi Arabia*

(Received 6 November 2001; accepted for publication 18 May 2003)

Abstract. For non-stationary processes, the conventional power spectrum does not reflect the time variation of the process characteristics. Higher-order cumulants or bispectrum has been applied in the analysis of non-minimum phase linear, time-invariant (LTI) systems under stationarity assumptions and restricting the signal to have non-symmetric distribution. It is clearly of interest to examine what type of analysis is available for those cases where the assumption of stationarity becomes untrue or unrealistic. In many practical applications such as seismic surveying, detection, system identification, sonar, communications, speech, and medical instrumentation, the signal analysts are faced with the task of processing signals whose spectral characteristics are time-dependent [1] and [2]. In this paper, a simple hardware architecture for computing the time-varying cumulants of non-stationary signals is proposed using systolic arrays.

1. Introduction

Due to a large number of applications in diverse fields, the problem of designing hardware implementations to deal with non-stationary signals is of utmost important. A common and well-known procedure for dealing with stationary stochastic processes is based on the application of the power spectrum. Although this procedure has a wide range of applications, it has certain limitations. The fact that the power spectrum carries only the magnitude information and no phase information implies that minimum-phase (or maximum-phase) characteristics had to be assumed a-priori. Since non-minimum phase systems are systems that have zeros inside and outside the unit circle, the analysis of non-minimum phase LTI systems cannot be done using the power spectral analysis. Furthermore, for non-stationary processes (processes which have at least one time-varying parameter), the conventional power spectrum does not reflect the time variation of the process characteristics. Higher-order cumulants or bispectrum has been applied in

the analysis of non-minimum phase LTI systems under stationarity assumptions and restricting the signal to have non-symmetric distribution [3], [4], [5].

System analysis when the input/output are non-stationary requires new techniques and a great deal of work is needed in this area. It is clearly of interest to examine what type of analysis is available for those cases where the assumption of stationarity becomes untrue or unrealistic. In many practical applications such as seismic surveying, detection, system identification, sonar, communications, speech, and medical instrumentation, the signal analysts [1], [2] are faced with the task of processing signals whose spectral characteristics are time-dependent. Therefore, non-stationary signals need special mathematical treatment when estimating their spectrum or when identifying systems with non-stationary input. An accurate spectral analysis of these signals cannot be accomplished by the simple use of classical time-domain or frequency-domain representation. To deal with time-dependent spectrum, the concept of time-frequency distributions has been introduced. These methods represent an attempt to provide a general solution to the problem of representing non-stationary signals. In order to develop a useful theory we need to replace stationarity by a more general notion which still allows us to carry out meaningful statistical analysis and to develop a form of time-dependent spectral analysis which shares many of the features of the spectral analysis of stationary processes [6], [7].

2. Spectrum and Polyspectrum

Let $x(n)$ be a stationary random process up to order k . Then, its k^{th} order cumulant is defined as [3]

$$c_k^x(x^{r_1}(i_1), x^{r_2}(i_2), \dots, x^{r_n}(i_n)) = (-j)^k \frac{\partial^k \ln \varphi_n(u_1, u_2, \dots, u_n)}{\partial u_1^{r_1} \partial u_2^{r_2} \dots \partial u_n^{r_n}} \Big|_{u_1=u_2=\dots=u_n=0}$$

where $k=r_1+r_2+\dots+r_n$ and

$$\varphi(u_1, u_2, \dots, u_n) = E\{e^{j\sum_{m=1}^n u_m x(i_m)}\}$$

which is called the cumulant generating function, and $E\{.\}$ is the expected value operator. The first, second, and third cumulants of $x(n)$ as functions of its moments are given by

$$c_1^x(n) = E\{x(n)\}$$

$$c_2^x(m_1) = E\{x(n)x(n+m_1)\} - (E\{x(n)\})^2$$

$$c_3^x(m_1, m_2) = E\{x(n)x(n+m_1)x(n+m_2)\} - E\{x(n)\}E\{x(n)x(n+m_1)\} \\ + E\{x(n)x(n+m_1)\} + E\{x(n+m_1)x(n+m_2)\} + 2(E\{x(n)\})^3$$

If $x(n)$ is a zero-mean stationary process, then its first, second, and third cumulants reduce to the first, second, and third order moments of $x(n)$, respectively.

The $(k+1)$ st order spectrum $S_{k+1}(w_1, w_2, \dots, w_k)$ of the process $x(n)$ is defined as the Fourier transform of its $(k+1)$ st order cumulant $c_{k+1}(m_1, m_2, \dots, m_k)$. That is

$$S_{k+1}(w_1, w_2, \dots, w_k) = \sum_{m_1=-\infty}^{\infty} \dots \sum_{m_k=-\infty}^{\infty} c_{k+1}(m_1, \dots, m_k) e^{-j(w_1 m_1 + \dots + w_k m_k)}$$

In general, $S_{k+1}(w_1, w_2, \dots, w_k)$ is complex and a sufficient condition for its existence is that $c_{k+1}(m_1, m_2, \dots, m_k)$ be absolutely summable.

Particular cases of the $(k+1)$ st order spectrums are the power spectrum and the bispectrum defined as follows:

Power Spectrum ($k=1$)

$$S_2(w) = \sum_{m=-\infty}^{\infty} c_2(m) e^{-jwm} \quad , |w| \leq \pi$$

Bispectrum ($k=2$)

$$S_3(w_1, w_2) = \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} c_3(m_1, m_2) e^{-j(w_1 m_1 + w_2 m_2)} \quad , |w_1| \leq \pi, |w_2| \leq \pi, |w_1 + w_2| \leq \pi$$

Let $H(z)$ be the transfer function of an LTI exponentially stable system, (i.e., its impulse response decays exponentially with time), whose input is an independent, identically distributed (i.i.d.) non-Gaussian process with non-zero $(k+1)$ order cumulant $\beta_{k+1}(m_1, \dots, m_k)$. The $(k+1)$ st order cumulant of the output process is related to the system impulse response $h(n)$ by

$$c_{k+1}(m_1, m_2, \dots, m_k) = \beta_{k+1} \sum_n h(n)h(n+m_1)\dots h(n+m_k) \quad (1)$$

In the frequency domain, the $(k+1)$ st order spectrum of the output process is related to the system transfer function $H(w)$ by

$$S_{k+1}(w_1, w_2, \dots, w_k) = \beta_{k+1} H(w_1)H(w_2)\dots H(w_k)H(-w_1 - w_2 - \dots - w_k) \quad (2)$$

The above two equations are of fundamental importance because they serve as the bases for most non-minimum phase system identification methods.

Most of the standard parameter estimation and linear system identification algorithms available in the literature estimate only a spectrally equivalent minimum phase system because these techniques exploit only the second-order statistics which suppress all phase information of the underlying process; thus they are incapable of identifying the non-minimum phase structure of the system. Furthermore, for non-stationary processes, the conventional power spectrum does not reflect the time variation of the process characteristics. With the recent introduction of the bispectrum in digital signal processing, a new approach to the solution of non-minimum phase system identification problem has been devised. This approach exploits the fact that the bispectrum contains information regarding both the phase and the magnitude of the system. Although the bispectrum has been applied in the identification of non-minimum phase LTI systems, it requires the assumption of stationarity and restricts the process to have non-symmetric probability density. Therefore, when the input/output of the system are non-stationary or when they have symmetric probability densities, system analysis requires a new approach. In [8] a new method based on the evolutionary spectrum was proposed to solve some of these problems. In this paper, we propose a hardware implementation for evaluating the time-varying cumulants of a non-stationary signal.

3. Normal Equations for Time-varying Autoregressive Moving Average Systems Using the Time-varying Cumulants

Consider the time-varying autoregressive moving average (TVARMA) equation

$$x(n) + \sum_{i=1}^p a_i(n) x(n-i) = e(n) + \sum_{i=1}^q b_i(n) e(n-i) \quad (3)$$

where the residuals $e(n)$ are a set of mutually independent stationary random variables, identically distributed with zero-mean and unit-variance. Then multiplying both sides of equation (3) by $x(n+m)x(n)$ and taking the expected value of both sides and assuming that $E\{e(n)e(n+m)e(n+k)\} = \gamma_n^e \delta(m,k)$, we have

$$c_n^x(m,0) + \sum_{k=1}^p a_k(n) c_n^x(m,-k) = 0 \quad , \quad m=q+1, q+2, \dots, q+p \quad (4)$$

where $c_n^x(m,k) = E\{x(n)x(n+m)x(n+k)\}$ is the time-varying third-order cumulant of $x(n)$. Equation (4) can be written in a matrix form as

$$C_n^x \mathbf{a}_n = -c_n^x \quad (5)$$

where for $k=1,2,\dots,p$ and $m=q+k=k$ when a time-varying AR model is considered, then we have

$$C_n^x = \begin{pmatrix} c_n^x(1,-1) & c_n^x(1,-2) & \cdots & c_n^x(1,-p) \\ c_n^x(2,-1) & c_n^x(2,-2) & \ddots & c_n^x(2,-p) \\ \vdots & \ddots & \ddots & \vdots \\ c_n^x(p,-1) & \cdots & \cdots & c_n^x(p,-p) \end{pmatrix} \quad (6)$$

$\mathbf{a}_n = [a_1(n) \ a_2(n) \ \dots \ a_p(n)]^T$, and $c_n^x = [c_n^x(1,0) \ c_n^x(2,0) \ \dots \ c_n^x(p,0)]^T$, where T stands for matrix transpose. The above equations constitute a set of linear equations at each instant of time n .

3.1 Estimation of the time-varying cumulants

The time-varying cumulants (TVC) can be estimated from the evolutionary bispectrum [7], [8] as follow:

$$\hat{c}_n(\mu\mu\ell) = \left(\frac{N}{2\pi M}\right)^2 \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \hat{S}(n, w_1, w_2) e^{jw_1\mu} e^{jw_2\ell} dw_1 dw_2 \quad (7)$$

where N is the data sequence length and M is an arbitrary constant depends on signal non-stationarity. Using the evolutionary bispectrum defined as

$$\hat{S}_x(n, w_1, w_2) = \hat{H}(n, w_1) \hat{H}(n, w_2) \hat{H}(n, -w_1 - w_2) \quad (8)$$

where

$$\hat{H}(n, w) = \sum_{m=0}^{N-1} w_n(m) x(m) e^{-jwm} \quad (9)$$

and $w_n(m)$ is a time-varying window. From equations (8) and (9), equation (7) becomes

$$\hat{c}_n(\mu\mu\ell) = \left(\frac{N}{M}\right)^2 \sum_{k=-\mu}^{N-1-\mu} w_n(k) x(k) w_n(k+\ell) x(k+\ell) w_n(k+\mu) x(k+\mu) \quad (10)$$

which is an estimate of the third-order time-varying cumulant of $x(n)$. In the case that $x(n)$ is stationary, equation (10) reduces to

$$\hat{c}_n(\mu\mu\ell) = \frac{1}{N} \sum_{k=0}^{N-1} x(k) x(k+\ell) x(k+\mu)$$

which is an estimate of the third-order cumulant of $x(n)$. Modifying the algorithm proposed by [9] to the time-varying signals, let C_n be a symmetric matrix defined as

$$C_n = \begin{pmatrix} c_n(p, -p) & c_n(p, -p+1) & \cdots & c_n(p, -1) \\ c_n(p-1, -p) & c_n(p-1, -p+1) & \ddots & c_n(p-1, -1) \\ \vdots & \ddots & \ddots & \vdots \\ c_n(1, -p) & c_n(1, -p+1) & \cdots & c_n(1, -1) \end{pmatrix}$$

then, by defining the upper triangular matrix U_n to be

$$U_n = \begin{pmatrix} w_n(N-1)x(N-1) & w_n(N-2)x(N-2) & \cdots & w_n(0)x(0) \\ 0 & w_n(N-1)x(N-1) & \ddots & w_n(1)x(1) \\ \vdots & 0 & \ddots & \vdots \\ 0 & \cdots & 0 & w_n(N-1)x(N-1) \end{pmatrix}$$

and D_n as a diagonal matrix whose elements are

$$d_n(I, I) = w_n(I)x(I), \quad I=0, 1, \dots, N-1$$

then C_n can be written in a matrix form as

$$C_n = U_n^t D_n U_n \quad (11)$$

In the following section, a hardware architecture for implementing the time-varying cumulants defined in equation (11) is proposed using systolic arrays.

4. Parallel Architecture for Computing the Time-frequency Spectrum

In this paper we present a linear (one dimensional) systolic system for computing TVC. We exploit the formulation of the problem, as a series of matrix multiplication operations rather than the direct application of the original definition of the problem [10, 11]. The main advantage of doing so is to be able to "re-use" old work on systolic and processor-array design for matrix multiplication. Systolic array design for matrix multiplication has received a significant attention from researchers during the last two decades [12 - 17]. Using such well known and established work as a building block in our proposed system makes it modular, easy to understand, easy to test and easy to enhance or modify. The other approach (which we will not address in this paper) is to

design a systolic array starting from the original definition of the problem using standard systematic mapping techniques e.g., [18]. However, the systematic design techniques, often, yield complex Processing Elements (PEs) designs which are difficult to understand, test or modify. Our main goal in this paper is to map the computations onto a linear systolic array. The basic PEs used are extremely simple and are very suitable for VLSI implementations. We focus on linear array implementations since they have several advantages over two dimensional arrays. Specifically, they are more suitable for VLSI implementation and require lower I/O bandwidth; usually $O(1)$ compared to two dimensional arrays which require $O(N)$ bandwidth where N is the number of data points. Therefore, as the problem size becomes larger, linear arrays become more attractive [19, 20]. Furthermore, linear arrays make it easier to incorporate fault tolerance capabilities into the system. In this paper we propose a linear system for computing TVC in $O(N^3)$ to $O(N^2)$ time using from $O(N^2)$ to $O(N)$ processing elements depending on the size of PEs local storage and choice of configuration. The proposed system uses, in part, a linear array for matrix multiplication presented in [19 - 22]. The rest of this section covers the proposed architecture in detail.

4.1 Linear array design for TVC

As mentioned earlier, we are dealing with a non-stationary process so the matrix C_n in equation (11) has to be computed at each instances of time (i.e., for $n=0, 1, \dots, N-1$). Nevertheless we will focus first on designing a system capable of computing a single C_n and later we will show how to compute all instances of C_n on the same system. Equation (11) is the core of our proposed architecture. By ignoring the scaling constant, we notice that the problem is redefined as a series of matrix multiplication operations of $N \times N$ matrices. All the elements of these matrices, except for D_n , could be pre-computed (for a fixed value of w). Matrix D_n is a diagonal matrix and its diagonal elements can be obtained "on the fly". Therefore, what is left is to carry out the multiplication operations. By re-examining (11) one can see that there are two types of matrix multiplication operations. Basically, the first type multiplies two regular matrices together. We call it non-trivial matrix multiplication. The second type multiplies a regular matrix with a diagonal matrix, which we call trivial matrix multiplication. Trivial matrix multiplication, which is multiplying a matrix A with a diagonal matrix B , is equivalent to multiplying each row of A with the corresponding diagonal element in B . Since no addition is involved, only multipliers are needed. If the data is fed serially, then a single multiplier (denoted as *S-MUL* module) will suffice as shown in the complete system design later. For non-trivial matrix multiplication, a full-fledged systolic array for matrix multiplication (denoted as *M-MUL* module) adopted from [19] is used and incorporated into the system. The over-all design consists of two building blocks of type *S-MUL* and *M-MUL* connected together in a serial fashion to form a linear system which implements the computations of equation (11).

The proposed architecture consists of two parts *S-MUL* and *M-MUL* that can perform the matrix multiplication of (11) as depicted in Fig. 1. *S-MUL* performs the first

multiplication and feeds its results to $M\text{-}MUL$ which is basically a matrix multiplication array based on the VMF design in [19], [22]. $M\text{-}MUL$ multiplies $U_n^T D_n$ by U_n to form C_n

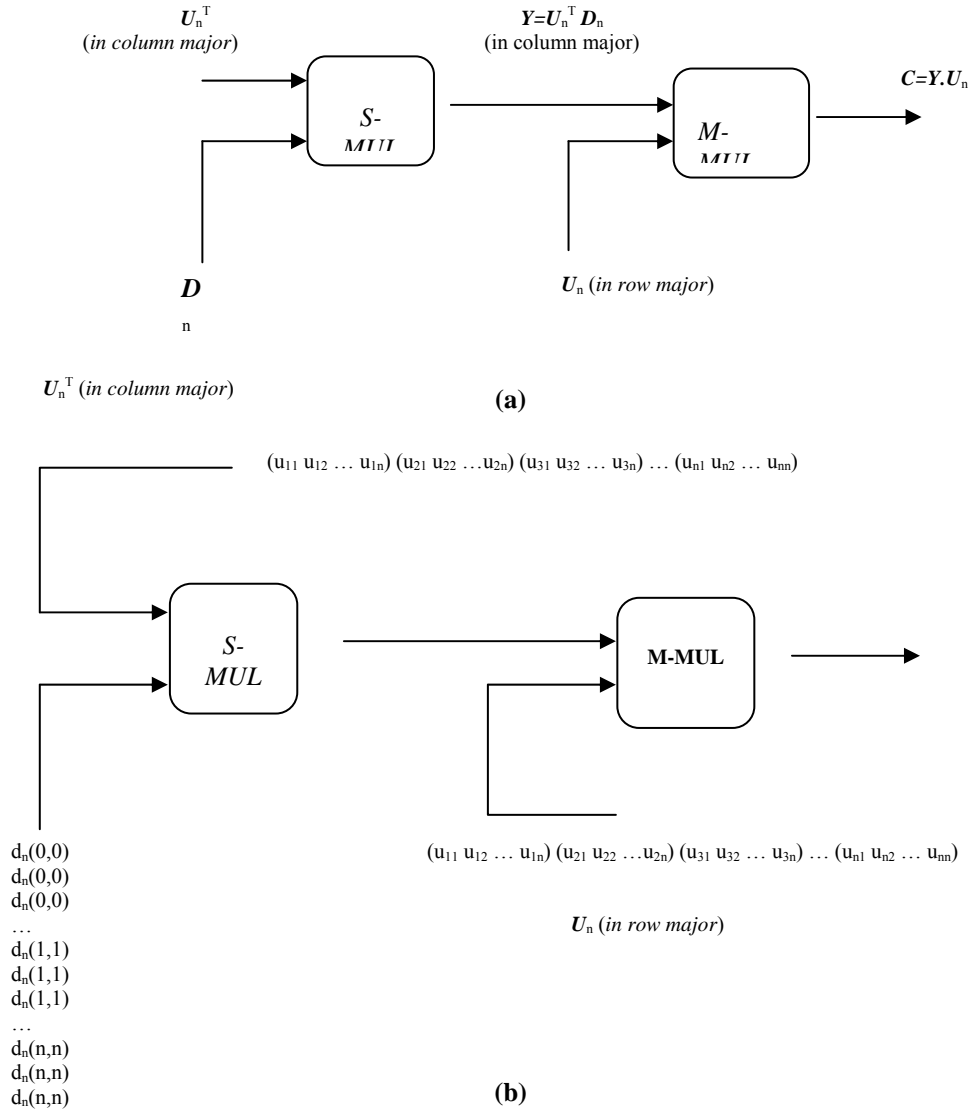


Fig. 1. The proposed architecture:
a) Block diagram,

b) Data flow into the system.

Care was taken to make sure that the output data of *S-MUL* is in the format required by *M-MUL* (i.e., Column major Row major as specified in [19]). The matrix \mathbf{D}_n is a diagonal matrix with non-zero entries $d_n(I,I)=w_n(I)x(I)$. Multiplying the matrix U_n^t by the diagonal matrix \mathbf{D}_n is equivalent to multiplying the first diagonal element $x(\cdot)$ of \mathbf{D}_n by the entries of the first column of U_n^2 , the second diagonal element $x(\cdot)$ by the non-zero entries of the second column, and so on. As mentioned earlier, since data is fed serially, it was surprisingly sufficient to put *S-MUL* as a single multiplier to multiply the two input sequences that correspond to the elements of U_n^t and \mathbf{D}_n matrices.

In order for the reader to understand how the whole system works, we will explain how the module *M-MUL* works since it is the most complex component. Let us consider the *VMF* systolic array presented in [19], [22] (assuming that the parameter S equals 1, where S is the internal storage capacity of Pes [19]). *M-MUL* is a linear array that multiplies two $m \times m$ matrices \mathbf{A} and \mathbf{B} to obtain \mathbf{C} . The linear array is obtained by mapping a 2-dimensional array onto a 1-dimensional array. As a simple example, assume that \mathbf{A} , \mathbf{B} and \mathbf{C} are 4×4 arrays. The flow of input data in the original 2-dimensional array is shown in Fig. 2(a). Each processor performs one multiply-and-accumulate operation and passes the \mathbf{A} and \mathbf{B} elements to its right and bottom neighbor respectively. All the operations for computing C_{ij} are performed by $PE_{(i-1)m+j}$. This particular 2-dimensional array is mapped onto 1-dimensional (linear) array by stretching each row in row major order and eliminating vertical links of the array. Therefore, the resultant linear array will have m^2 (in this case = 4^2) Pes as shown in Figure 2(b). In the resultant one-dimensional array, the data flows into the array through the left most PE. The elements of the \mathbf{A} matrix are fed in column major while those of the \mathbf{B} matrix are fed in row major order, which correspond to the \mathbf{A} and \mathbf{B} bands in Figure 2. Each element $b_{1j}(1 \leq j \leq 4)$ in the \mathbf{B} band has to meet with the element $a_{1,1}$ in the \mathbf{A} band. Since all data is fed serially at the leftmost PE, in order for $a_{1,1}$ to meet the elements of the \mathbf{B} band, the \mathbf{B} band data must travel faster than the \mathbf{A} band data. This alignment of operands is achieved using two speed data channels. The general methodology used to design a linear array for matrix multiplication given in [19] assuming that we will partition the 2-D array into m -rows is as follows:

- The 2-dimensional array is partitioned into m rows: $CROW_1, CROW_2, \dots, CROW_m$.
- The linear array consists of m blocks each block having m Pes. The computation performed by the PE's in block _{i} is the computations of Pes in $CROW_i, (1 \leq i \leq m)$.
- The Pes are selectively activated to perform a step of the matrix multiplication algorithm.

- Within each block, the elements of B matrix are saved in a slow channel that will be used by the PE's in the next block. At the end of each block, the B matrix data is switched from slow to fast channels so that they can commute with the elements of A matrix within the next block.

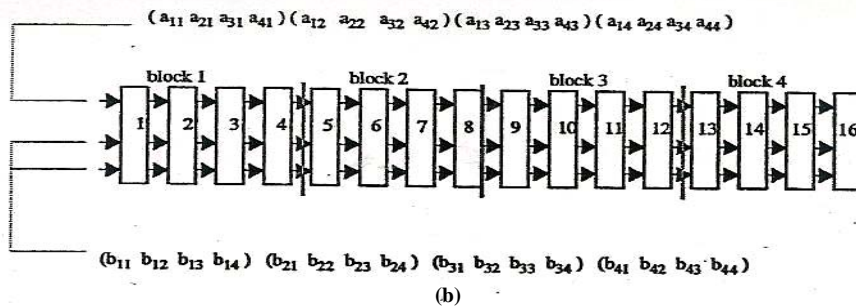
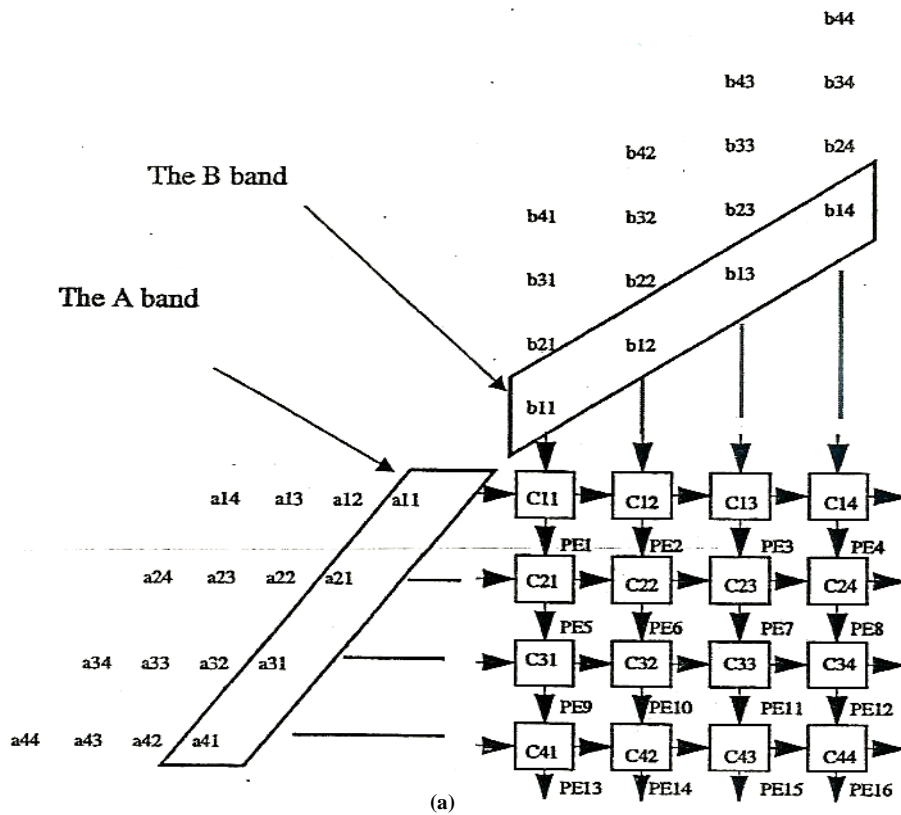


Fig. 2. The linear array with continuous data streams used in M-MUL (Courtesy of [19]):

- a) Original 2-dimensional array.
- b) Linear array.

The array is built using the basic PE shown in Fig. 3 that contains only one Multiply and Accumulate (MAC) unit. Data travels across the array through chains of registers (or channels). There are three major channels for moving data: one fast channel (BF) which carries the elements of B within one block, and two slow channels AS and BS which are used to carry the elements of A and B , respectively, as shown in Figs. 3 and 4. Control is accomplished using, one bit wide control lines I , J and ACT which connect neighboring PEs. Rows of the B matrix are moved from $CROW_i$ to $CROW_{i+1}$ (i.e., from block _{i} to the block _{$i+1$}) using the slow channel BS. Moreover, the fast channel is used to move the data (of the B matrix) within a block of PEs. At the end of each block, data in the fast channel are discarded. Fig. 3 shows hardware details inside PEs to perform the channel switching at the end of each block by using multiplexes and the control signal ϕ .

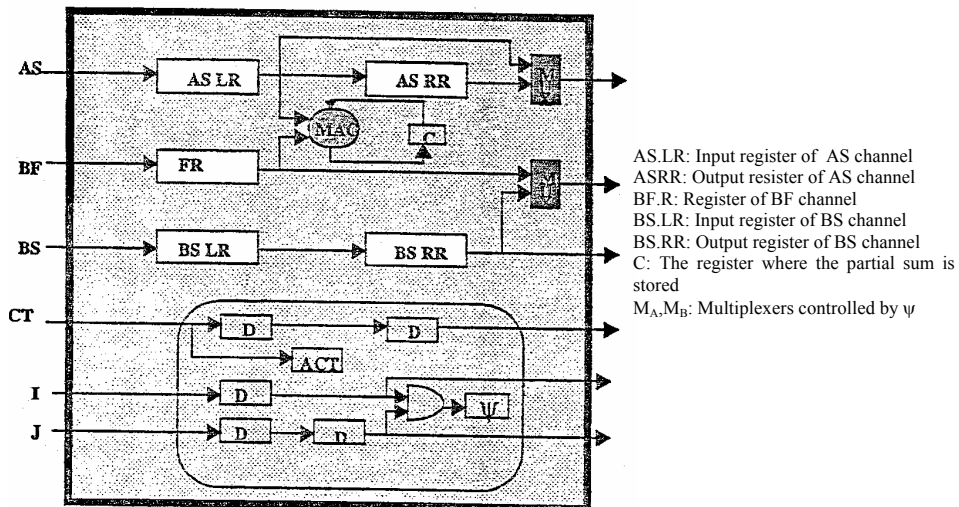


Fig. 3. Block diagram of the basic PE used in M-MUL (Courtesy of [19]).

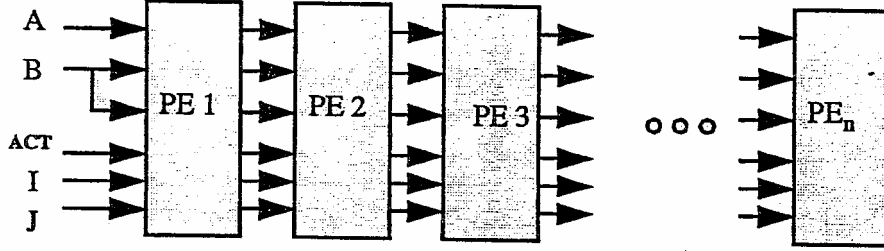


Fig. 4. Organization of the PEs in M-MUL.

The control signal ACT flag is used to trigger a partial product computation and the flag ϕ is used to control data multiplexing. If ($\phi=1$) then multiplexer M_A selects data from AS.LR and M_B selects data from BS.RR; otherwise M_A selects data from AS.LR and M_B selects data from BS.RR. A PE is activated to perform a multiply-and-accumulate operation $C_{ij} = C_{ij} + a_{ik} * b_{kj}$ (a_{ik} in the AS.LR of the slow channel is multiplied with b_{kj} in the register BF.R of the fast channel (see Fig. 3)) when the signal $ACT=1$. At the end of a block, the current element of matrix A is deactivated and the next element of A is activated. Concurrently with feeding data, ϕ is set in $PE_m * i$ ($1 \leq i \leq m$) using the I and J control signals. The data are input in every clock period continuously. Also, the control input ACT is set to 1 every time a_{ik} ($1 \leq k \leq m$) is inserted, into the array. For more details on the operation of the array see [22]. This array performs the multiplication of two $m \times m$ matrices using m^2 PEs in $3m^2 - m - 1$ clock cycles [22] (assuming that $S=1$). Recall that in our system, m equals N (the size of the matrices is $N \times N$).

Now that we have a linear array that can compute C_n for a single value on n , what is left is to replicate the process to compute all C_n for all possible values of n . In general one can replicate the architecture N times and let each linear system compute a C_n matrix. This will guarantee that all computations will be performed in parallel so the whole system will have an $O(N^2)$ computation time which is the time needed to compute an instance of C_n . On the other hand, it is also possible to compute all C_n matrices (i.e. for $n=0, 1, \dots, N-1$) using the single linear system showing in Figure 1 sequentially and thus no replication of hardware is needed. The tradeoff between the two options will be discussed in the next section.

4.2 Performance analysis

As mentioned earlier, $M-MUL$ has a computation time of $N^2 + 2N(N/S) - (N/S) + 1$ clock cycles, where S is the size of PEs local storage. Therefore, the over-all computation time involved in computing one matrix C_n is given by:

$$\begin{aligned} \text{Computation Time} &= T_{M-MUL} + T_{S-MUL} \\ &= [N^2 + 2N(N/S) - (N/S) + 1] + 1 \end{aligned}$$

which is $O(N^2)$; remarkably faster than the non-parallel computation time that is of $O(N^3)$. On the other hand the number of PEs in the system equals the sum of PEs in the building blocks which is given by

$$\text{Number of PEs} = N(N/S) + 1$$

which ranges from $O(N^2)$ to $O(N)$ depending on the PEs local storage S . No previous work on the same subject is available for comparison. Table 1, however, summarizes the main features of the proposed architecture for computing one C_n matrix.

As for the over-all design for computing all instances of C_n , the first option discussed in the previous section (that is replicating the system for each instance of C_n) is N times faster than the second option (that is using a single linear system to compute all instances of C_n sequentially) but this performance boost comes at a cost. The second option will require N times the hardware and N times the I/O requirement of the first option which could make it undesirable if the system is to be realized in VLSI. It is up to the designer to judge according to the needs and constraints of his/her particular application and constraining environment.

Table 1. Main features of the proposed architecture for computing one C_n matrix

# of PEs	$N [N/S]$ for $M-MUL$ Plus 1 for $S-MUL$
# of MACs/PE	1
Computation Time	$\{N^2+2N [N/S]-[N/S] +1\} +1$
Max Speedup over Serial Comp.	$O(N)$

5. Conclusion

In this paper, a new approach for linear systolic array realization of time-varying cumulants (TVCs) is proposed. The underlying matrix multiplication formulation obviously simplifies the design process considerably. A very simple architecture that uses an existing matrix multiplication systolic array as a building block has been developed. The new design has the advantage of simplicity, flexibility and the suitability for VLSI implementation. An obvious advantage of the development presented in this paper is putting the problem in a formulation that can benefit from past (and possibly future) research in systolic architectures for matrix multiplication. A specific example from the literature [19] has been adopted in this paper because of its simplicity and efficiency, but other techniques (see [12]) can be considered as well. The decision of which approach for matrix multiplication to choose would depend on the overall design objectives and constraints.

References

- [1] Haykin, S. *Advances in Spectrum Analysis and Array Processing*. Englewood Cliffs, NJ:Prentice-Hall: 1991.
- [2] Kay, S. M. *Modern Spectral Estimation: Theory and Application*. Englewood Cliffs, NJ:Prentice-Hall: 1988.
- [3] Nikias, C. L. and Raghuvver, M. R. "Bispectrum Estimation: A Digital Signal Processing Framework." *IEEE Proc.*, 75, No. 7, (July 1987), 869-891.
- [4] Rosenblatt, M. "Linear Processes and Bispectrum." *J. Appl. Prob.*, 17 (1980), 265-270.
- [5] Brillinger, D. R. and Krishnaiah, P. R. (Eds.) *Handbook of Statistics*. Vol. 3. "Cumulants and Cumulant Spectra, by M. Rosenblatt." Elsevier Science Publisher B. V. (1983), 369-382.
- [6] Priestley, M. B. *Non-linear and Non-stationary Time Series Analysis*. New York: Academic Press, 1988.
- [7] Priestley, M.B. and Gabr, M.M. "Bispectral Analysis of Non-stationary Processes." In: Rao, C. R. (Ed.). *Multivariate Analysis: Future Directions*. Amsterdam: North-Holand, 1993.
- [8] Al-Shoshan, A. I. "System Identification and Modeling of Nonstationary Signals." Ph.D. Dissertation, University of Pittsburgh, PA, 1995.
- [9] AlOqeely, M. A. and Alshebeili, S. A. "A New Approach for the Design of Systolic Arrays for Computing Third and Fourth-order Moments." *Can. J. Elect. & Comp. Eng.*, 23, No. 3 (1998), 127-132.
- [10] Evans, D.J. "A Systolic Array for the Parallel Solution of Block Tridiagonal Linear Systems." *International Journal of Computer Mathematics*, 71 (1999), 57-70.
- [11] Wan, C.R. and Evans, D.J. "Nineteen Ways of Systolic Matrix Multiplication." *International Journal of Computer Mathematics*, 68 (1998), 39-69.
- [12] Risset, T. "Linear Systolic Arrays for Matrix Multiplication: Comparison of Existing Synthesis Methods and New Results." *Algorithms and Parallel VLSI Architectures II*, P. Quinton and Y. Robert (Eds.), Elsevier Publishers B.V., 1992.
- [13] Chang, Y.T. and Wang, C.L. "A New Fast DCT Algorithm and Its Systolic VLSI Implementation." *IEEE Transactions on Circuits and Systems II-analog and Digital Signal Processing*, 44 (1997), 959-962.
- [14] Diamantaras, K.I. and Kung, S.Y. "A Linear Systolic Array for Real-time Morphological Image Processing." *Journal of VLSI Signal Processing Systems For Signal Image and Video Technology*, 17 (1997), 43-55.
- [15] Megson, G.M. and Bland, I.M. "Generic Systolic Array for Genetic Algorithms." *IEE Proceedings-Computers and Digital Techniques*, 144 (1997), 107-119.
- [16] Oksa, G. and Vajtersic, M. "Systolic Block-jacobi SVD Algorithm for Processor Meshes." In: *Highly Parallel Computations: Algorithms and Applications*, Bekakos, M.P. (Ed.), Southampton, U.K. WIT Press, Wessex Institute of Technology, (2001), 211-235.
- [17] Oksa, G. and Vajtersic, M. "A Systolic SVD Solver for Toroidal Processor Meshes." In: *Proc. Int. Workshop "Parallel Numerics 2000"*, Bratislava (Slovakia), 2000, 21-38.
- [18] Rao, S. and Kailath, T. "Regular Iterative Algorithms and Their Implementation on Processor Arrays." *IEEE Proc.*, 3 (1988), 259-269.
- [19] Kumar, V.K.P. and Tsai, Y. "On Synthesizing Optimal Family of Linear Systolic Arrays for Matrix Multiplication." *IEEE Trans. Computer*, 40, No. 6 (1991), 770-774.
- [20] Verman, P. J. and Ramakrishnan, I. V. "Synthesis of an Optimal Family of Matrix Multiplication Algorithms on Linear Arrays." *IEEE Trans. on Computer*, 35, No.11 (1986).
- [21] Oksa, G. and Evans, D.J. "Triangular Systolic Arrays for QZ Matrix Decomposition." *Neural, Parallel and Scientific Comp.* 6 (1998), 1-10.
- [22] Kumar, V. P. and Tsai, Y. "On Mapping Algorithms to Linear and Fault-tolerant Systolic Arrays." *IEEE Trans. on Computers*, 38, No.3 (1989), 470-478.

(/ / / /)