TECHNICAL
RESEARCH
REPORT

SYSTEMS
RESEARCH
CENTER

*Supported by the
National Science Foundation
Engineering Research Center
Program (NSFD CD 8803012),
the University of Maryland,
Harvard University,
and Industry*

Parallel and Fully-Pipelined Instantaneous Optimal Weight Extraction for Adaptive Beamforming Using Systolic Arrays

by C.F.T. Tang, K.J.R. Liu and S.A. Tretter

TR 91-79

Parallel and Fully-Pipelined Instantaneous Optimal Weight Extraction for Adaptive Beamforming Using Systolic Arrays

C.F. T. Tang, K.J. R. Liu, and S. A. Tretter

Systems Research Center

Electrical Engineering Department

University of Maryland

College Park, MD 20742

Abstract

In this paper we present systolic algorithms and architectures for parallel and fully-pipelined instantaneous optimal weight extraction for multiple sidelobe canceller (MSC) and minimum variance distortionless response (MVDR) beamformer. The proposed systolic parallelogram array processors are parallel and fully pipelined, and they can extract the optimal weights instantaneously without the need for forward or backward substitution. We also show that the square-root-free Givens method can be easily incorporated to improve the throughput rate and speed up the system. As a result, these MSC and MVDR systolic array weight extraction systems are suitable for real-time VLSI implementation in practical radar/sonar systems.

1 Introduction

The problem of weight extraction for systolic adaptive beamforming systems has been the subject of intense research since the first well-known work of Gentleman and Kung on recursive least squares (RLS) systolic arrays [1, 2]. Their approach is based on the QR decomposition (QRD) which is numerically stable. Although the QRD updates proposed in [1] are pipelined on a triangular array, their fully pipelined weight extraction by using the RLS systolic array consists of the two separate steps of QR-updates and backward substitution and has been shown to be unrealizable [3].

A major challenge in implementing the RLS algorithm for the multiple sidelobe cancellation (MSC) adaptive beamforming system using systolic array processors is to design a single fully pipelined structure. A critical obstruction appears because the process of the QR-updates runs from the upper-left corner to lower-right corner of the array, while the process of the backward substitution runs in exactly the opposite direction as pointed out in [3]. Much research has been done on this subject recently [4, 5, 6]. In [4], Hudson and Shepherd have proposed a Kalman closed-loop system for the RLS parallel weight extraction problem that consists of a systolic QRD post-processor to compute the least squares weighting vector. However, the parallel RLS weight extraction system proposed is not efficient for VLSI hardware implementation. The major hurdles are that this system, which requires two modes to update the data and to freeze the updated data for computing the weight vector error at the same time, is inefficient in obtaining the instant weight vector recursively, and that the feedback configuration may create serious routing problems in VLSI implementation and roundoff noise accumulation [4].

In [5], McWhirter introduced a fixed parallelogram structure for the parallel weight extraction in the RLS problem. However, McWhirter's RLS weight extraction system does not efficiently update the weight vector since the array must be frozen at each snapshot to compute the weights.

In a recent paper [7], a numerically stable and computationally efficient algorithm for weight extraction for a constrained recursive least squares (CRLS) problem has been described by Schreiber. Although the algorithm shown in [7] has robust numerical properties, it is difficult to arrange the whole algorithm into a single fully pipelined structure as pointed out in [8, 9]. The difficulty, which is the same as that in the RLS case, arises because the CRLS algorithm consists of several steps particularly involving the backward substitution step. Many MVDR adaptive beamformers with CRLS systolic array structures proposed in [8, 9, 10] are designed to avoid the extra backward substitution processor for computing the residual. Unfortunately, for the problem of parallel/pipelined weight extraction, very little has been done in implementing the CRLS algorithm into a single fully pipelined systolic array structure without requiring the backward substitution processors. Recently, Owsley developed an adaptive MVDR beamformer with a systolic array implementation using Schreiber's CRLS algorithm for weight extraction [11, 12]. Nevertheless, Owsley's CRLS systolic array structure which consists of several block processors including the forward and backward substitution processors has been shown to be unpipelinable. Subsequently, Tang, Liu, and Tretter [6, 13] presented systolic architectures for MSC and MVDR adaptive array systems. During the preparation of this paper, the authors discovered a paper by Shepherd et al. [14] in which they independently proposed a similar concept for RLS and CRLS algorithms.

In this paper, fully parallel/pipelined systolic arrays for the MSC and MVDR adaptive weight extraction systems without the need for forward or backward substitution are described. The proposed MSC and MVDR systolic adaptive array systems have five advantages: (1) they are simple, modular, and expandable so as to be very suitable for VLSI hardware implementation, (2) the square root free Givens method can be easily incorporated into the architectures, (3) they are fully pipelined since the backward substitution is avoided, (4) they are open-loop systems without any feedback arrangement, and (5) they function recursively to update the instantaneous optimal weight vector since only one mode is required in the recursive updating.

This paper is organized as follows. In Section 2, two types of adaptive beamformers are introduced. They are the multiple sidelobe canceller (MSC) formulated as the recursive least squares (RLS) problem and the minimum variance distortionless response (MVDR) beamformer formulated as the constrained recursive least squares (CRLS) problem. In Section 3, the background and the new techniques to replace the forward and backward substitutions by the parallel multiplication and accumulation operation using systolic arrays are considered. In Section 4, the QR-based RLS algorithm for the MSC adaptive beamforming system without the need for backward substitution is described and its parallel/pipelined MSC weight extraction system with systolic array processor is also presented. In Section 5, the QR-based CRLS algorithm for the MVDR adaptive beamforming without the need for forward and backward substitutions is described and the parallel/pipelined MVDR weight extraction system with systolic array processor implementation is also considered. In Section 6 the fast, square-root free, Givens method is employed for the MSC and MVDR adaptive array systems.

2 Adaptive Beamforming Systems

In this section, two popular adaptive beamforming systems are described. We first describe the multiple sidelobe canceller (MSC) which consists of a main antenna and several auxiliary antennas. The multiple sidelobe cancellation technique is employed to suppress the sidelobe interferences and noises. The interferences and noises are estimated by multiplying the observed input data received through the auxiliary antennas by the adaptive weights and then summing them together. The interferences and noises are suppressed from the main radar channel by subtracting the estimates from radar main channel output. Second, a minimum variance distortionless response (MVDR) beamformer is considered. The MVDR beamforming technique is used to suppress the interferences and noises by constraining the response of the beamformer to specific directions and then minimizing the output power subject to the response constraints.

2.1 Multiple Sidelobe Canceller (MSC)

The block diagram of an multiple sidelobe canceller (MSC) is shown in Figure 1. It is easy to see that the output at the *ith* snapshot can be expressed as

$$y(t_i) = \sum_{l=1}^{N} x_l(t_i) w_l - z(t_i). \tag{1}$$

A set of n successive snapshots can be represented in the vector form

$$\underline{y}(n) = X(n)\underline{w}(n) - \underline{z}(n), \tag{2}$$

where $\underline{y}(n)$ is an n by 1 output vector matrix

$$\underline{y}(n) = \begin{bmatrix} y(t_1) \\ y(t_2) \\ \vdots \\ y(t_n) \end{bmatrix}, \tag{3}$$

$\underline{z}(n)$ is an n by 1 desired input data vector matrix

$$\underline{z}(n) = \begin{bmatrix} z(t_1) \\ z(t_2) \\ \vdots \\ z(t_n) \end{bmatrix}, \tag{4}$$

$X(n)$ is an n by N observed input data matrix

$$X(n) = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \cdots & x_N(t_1) \\ x_1(t_2) & x_2(t_2) & \cdots & x_N(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_n) & x_2(t_n) & \cdots & x_N(t_n) \end{bmatrix}, \tag{5}$$

and $\underline{w}(n)$ is an N by 1 weight vector

$$\underline{w}(n) = \begin{bmatrix} w_1(t_n) \\ w_2(t_n) \\ \vdots \\ w_N(t_n) \end{bmatrix}. \tag{6}$$

The aim of the MSC system is to minimize the output residual of the total interferences and noises; i.e. to minimize the sum of the squares of the elements of the n by 1 output residual vector $\underline{y}(n)$. This leads to a maximization of output residual signal to noise (including interferences and receiver noises) ratio. Therefore, we have the least-squares problem

$$\min_{\underline{w}} \|\underline{y}(n)\|_2 = \min_{\underline{w}} \|X(n)\underline{w}(n) - \underline{z}(n)\|_2. \tag{7}$$

The solution to this minimization problem is [15]

$$\underline{w}_{LS}(n) = (X^H(n)X(n))^{-1}X^H(n)\underline{z}(n) = M^{-1}(n)X^H(n)\underline{z}(n), \tag{8}$$

where the superscript H denotes Hermitian.

This direct solution, generally known as the sample matrix inversion (SMI) method in adaptive array and multichannel applications, is often difficult to compute numerically since $M(n)$ is frequently ill conditioned.

2.2 Minimum Variance Distortionless Response (MVDR) Beamformer

The minimum variance distortionless response (MVDR) adaptive beamforming system can be formulated as a constrained recursive least squares (CRLS) problem. The block diagram of the MVDR adaptive beamformer is shown in Figure 2. The output y at the kth snapshot of the adaptive beamformer is

$$y(t_k) = \sum_{l=1}^{N} x_l(t_k) w_l. \tag{9}$$

If n snapshots of input data are available, the output can be written in the vector form

$$\underline{y}(n) = X(n)\underline{w}(n) \tag{10}$$

where $\underline{y}(n), X(n)$, and $\underline{w}(n)$ are as defined in (3), (5), and (6), respectively.

A signal received from the direction of interest θ_i is called a beamformer response r^i and is given by

$$r^i = \underline{c}^{i^H} \underline{w}, \tag{11}$$

where $\underline{c}^{i^H}$ is the N by 1 mainbeam steering direction vector given by

$$\underline{c}^{i^H} = \begin{bmatrix} 1 & e^{j\frac{2\pi}{\lambda} Dsin(\theta_i)} & e^{j\frac{2\pi}{\lambda}2\, Dsin(\theta_i)} & \ldots & e^{j\frac{2\pi}{\lambda}(N-1)\, Dsin(\theta_i)} \end{bmatrix}$$

with D being the distance between two sensors in the array, and λ being the wavelength. Taking expectation of $\underline{y}^H(n)\underline{y}(n)$ yields

$$E[\underline{y}^H(n)\underline{y}(n)] = E[\underline{w}^H(n)X^H(n)X(n)\underline{w}(n)] = \underline{w}^H(n)M(n)\underline{w}(n), \tag{12}$$

where $M(n)$ is the n by n covariance matrix of $X(n)$.

In general, designing a MVDR beamformer requires solving a constrained least squares optimization problem. The problem is

$$\min_{\underline{w}} \underline{w}^H M \underline{w}$$

$$\text{subject to } \underline{c}^{i^H} \underline{w} = r^i, \quad for \;\; i = 1, 2, \cdots, K,$$

where K is the number of look directions. Using the method of Lagrange multipliers, the optimal weight vector, $\underline{w}^i_{opt}$, is found to be [15]

$$\underline{w}^i_{opt}(n) = \frac{r^i M^{-1}(n)\underline{c}^i}{\underline{c}^{i^H} M^{-1}(n)\underline{c}^i} \quad for \;\; i = 1, \cdots, K. \tag{13}$$

In practice, $M(n)$ used in Equation 8 and 13 is the sample covariance matrix of the observed data $X(n)$,

$$M(n) = X^H(n)X(n) \tag{14}$$

rather than the covariance matrix of $X(n)$ used in Equation 12.

3 Systolic Array Linear Algebra Processing

In the emerging field of *algorithmic engineering* introduced by McWhirter [5], the hybrid disciplines of designing numerically stable parallel algorithms suitable for parallel computation and mapping them onto VLSI systolic architectures to achieve high throughput rates and VLSI hardware implementation are demanded for sophisticated, high performance real-time modern signal processing. In real-time modern signal processing applications, numerically reliable and computationally efficient algorithms and architectures for techniques such as recursive least squares estimation (RLS), constrained recursive least squares estimation (CRLS), solving linear systems, and performing singular value decomposition are required. Furthermore, in these applications it is also necessary to design a highly parallel/pipelined structure for the use in parallel supercomputers and for implementation by using VLSI systolic processors.

In this section some key processors are developed as the basic tools for designing sophisticated adaptive array systems. Moreover, the parallel/pipelined techniques considered here make it possible to design more advanced adaptive array systems such as the MSC and MVDR adaptive beamforming systems studied in this paper. A brief description of the key linear-algebra-based parallel algorithms with systolic array processors is provided in the following subsections.

3.1 Systolic Array for Preventing Forward Substitution

The computation of $R^H F = X$ is usually called the forward substitution where F is $n \times m$ matrix, R is $n \times n$ upper triangular matrix, and X is $n \times m$ matrix. In [16] it is carried out in two steps to obtain F by the Comon-Robert's algorithm when the matrices R and X are given. In the first step, the matrix R^{-1} is computed when the matrix R is fed into the systolic array. In the second step, the vector X^H is given as input to compute $X^H R^{-1}$. As a result, the complex conjugate of $X^H R^{-1}$ is taken to obtain F. Similar to [8, 17], the systolic algorithm shown in Table 1 requires only one step to generate F with the matrix R prestored in the systolic array and the matrix X as input fed into the array.

The systolic parallelogram array processor designed to obtain F without computing forward substitution is illustrated in Figure 3. The systolic parallelogram array processor is operated by sending X into the upper triangular systolic processor in parallel to obtain F [6].

3.2 Systolic Array for Preventing Backward Substitution

Backward substitution is required in many adaptive array algorithms. Assume the vector $\underline{z}$ and the lower triangular matrix R^{-H} are the given data. The question now is how to design a systolic structure to compute $\underline{b} = R^{-1}\underline{z}$, by using the given vector $\underline{z}$ and lower triangular matrix R^{-H}. Fortunately, this can be easily carried out on a systolic array efficiently. The technique introduced in this subsection shows that instead of solving $R\underline{b} = \underline{z}$, the backward substitution, the parallel multiplication and

```
for i = 1 to n
  begin
  $y_{i1} = \frac{1}{r^*_{11}} x_{i1}$
  in parallel for j = 2 to m
    begin
    $z_{ij} = x_{ij}$
    in parallel for k = 1 to j - 1
      $z_{ij} = z_{ij} - y_{ik} r^*_{kj}$
      $y_{ij} = \frac{z_{ij}}{r^*_{jj}}$
    end
  end
```

Table 1: The Parallel/Pipelined Algorithm for Preventing Forward Substitution

```
in parallel for $i = 1$ to $m$, $j = 1$ to $m$
  begin
  $temp(i,j) = z(j) * x^*(i,j)$
end in parallel
in parallel for $i = 1$ to $m$
  begin
  $w(i) = 0$
  in parallel for $j = 1$ to $m$
    begin
    $w(i) = w(i) + temp(i,j)$
  end in parallel
end in parallel
```

Table 2: The Parallel/Pipelined Algorithm for Preventing Backward Substitution

accumulation of $\underline{z}$ and matrix R^{-H} is employed.

In Table 2, the systolic algorithm for the parallel multiplication and accumulation operation of a given vector $\underline{z}$ and matrix R^{-H} to prevent computing backward substitution is described. The systolic array shown in Figure 4 is designed by concurrently sending each element of the vector $\underline{z}$ to multiply the complex conjugate of each element of the matrix R^{-H} and then by summing them together to obtain the vector $\underline{b}$. It is clear that the vector $\underline{b}$ can be obtained by the parallel multiplication and accumulation operation of the data stored in the lower triangular part of the systolic array processor without performing the backward substitution [6].

4 QRD-MSC Adaptive Beamformer

Pipelined data-parallel algorithms that can be implemented on systolic array processors (SAPs) are called *systolic algorithms* (SAs). The SAs designed in this paper not only can be implemented by SAPs but also are numerically stable, an important property that the SMI method does not possess. The SAPs have additional nice properties such as simplicity, modularity, and expandability which are very suitable for implementations of adaptive beamforming systems onto VLSI. Conventional SMI methods generally lead to some undesirable numerical properties. In order to alleviate this difficulty, the QRD can then be used.

The QR based MSC adaptive beamforming system is described in this section while the QR based MVDR adaptive beamformer will be considered in the next section. In this section we introduce a single and fully pipelined systolic parallelogram array processor for optimal weight extraction in the MSC adaptive beamforming system. The adaptive beamforming system requires two modes for initialization and only one mode for recursive updating to obtain the optimal weights. By using a QRD, we avoid computing the sample covariance matrix inversion as needed in Equation 8 and Equation 13.

4.1 QR Based RLS Algorithm for MSC

The N-snapshot input data matrix $X(N)$ is received during the initialization period $0 \leq n \leq N$. Applying the QRD to the data matrix $X(N)$, we have

$$R(N) = Q(N)X(N), \tag{15}$$

where $R(N)$ is an $N \times N$ upper triangular matrix and $Q(N)$ is a unitary matrix with $Q^H(N)Q(N) = I$. Then by applying the unitary matrix $Q(N)$ to the desired data

$\underline{z}(N)$, the orthogonalized desired vector $\underline{u}(N)$ is given by

$$\underline{u}(N) = Q(N)\underline{z}(N), \tag{16}$$

where $\underline{u}(N)$ is an $N \times 1$ vector.

Combining Equations 15 and 16, that use the QRD, we have

$$\left[\begin{array}{cc} R(N) & \underline{u}(N) \end{array} \right] = Q(N) \left[\begin{array}{cc} X(N) & \underline{z}(N) \end{array} \right]. \tag{17}$$

Let us call this the mode 1 operation. Finally, using systolic parallelogram array processor described in Subsection 3.1, the initial lower triangular matrix $R^{-H}(N)$ can be generated as in Table 1 and Figure 3. This operation is called the mode 2 operation.

For n greater than N, it is known [18] that, in recursive updating, the unitary matrix $Q(n)$ consists of two factors given by

$$Q(n) = \check{Q}(n)\overline{Q}(n-1), \tag{18}$$

where $\check{Q}(n)$ is a n by n unitary matrix obtained from a sequence of Givens rotations that updates the new data row $\underline{x}^T(n)$, and $\overline{Q}(n-1)$ is a n by n unitary matrix given by

$$\overline{Q}(n-1) = \left[\begin{array}{ccc} Q(n-1) & \vdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \vdots & 1 \end{array} \right].$$

Thus, when the unitary matrix $Q(n)$ is applied to update the current data matrix $X(n)$, we have

$$Q(n)X(n) \;=\; Q(n) \left[\begin{array}{c} X(n-1) \\ \cdots \\ \underline{x}^T(t_n) \end{array} \right]$$

$$= \tilde{Q}(n) \begin{bmatrix} \beta R(n-1) \\ 0 \\ \cdots \\ \underline{x}^T(t_n) \end{bmatrix} = \begin{bmatrix} R(n) \\ 0 \\ \cdots \\ 0 \end{bmatrix}. \tag{19}$$

where β, the forgetting factor, is defined as the weight factor applied to the previous data. Applying the same unitary matrix $Q(n)$ to the desired signal, we obtain

$$Q(n)\underline{z}(n) = Q(n) \begin{bmatrix} \underline{z}(n-1) \\ \cdots \\ z(t_n) \end{bmatrix}$$

$$= \tilde{Q}(n) \begin{bmatrix} \beta\underline{u}(n-1) \\ \beta\underline{v}(n-1) \\ \cdots \\ z(t_n) \end{bmatrix} = \begin{bmatrix} \underline{u}(n) \\ \beta\underline{v}(n-1) \\ \cdots \\ \# \end{bmatrix}, \tag{20}$$

where $\underline{u}(n-1)$ is an $N \times 1$ vector, $\underline{v}(n-1)$ is an $(n-N-1) \times 1$ vector, and # denotes an arbitrary value of no interest in mathematical and physical concept. Equations 19 and 20 update $R(n-1)$ and $\underline{u}(n-1)$ once the data row $\underline{x}^T(t_n)$ and new desired data $z(t_n)$ are available.

It is also necessary to update the lower triangular matrix $R^{-H}(n-1)$ for computing the instantaneous optimal RLS weighting vector. It can be seen that the same unitary matrix $\tilde{Q}(n)$ can also be used to update the lower triangular matrix $R^{-H}(n-1)$ as given by

$$\begin{aligned} I &= R^H(n)R^{-H}(n) \\ &= R^H(n-1)R^{-H}(n-1) \\ &= \begin{bmatrix} \beta R^H(n-1) & 0 & \underline{x}^*(t_n) \end{bmatrix} \tilde{Q}^H(n)\tilde{Q}(n) \begin{bmatrix} \frac{1}{\beta}R^{-H}(n-1) \\ \# \\ 0 \end{bmatrix}. \end{aligned} \tag{21}$$

Therefore, from Equation 21, we obtain,

$$\grave{Q}(n)\begin{bmatrix} \frac{1}{\beta}R^{-H}(n-1) \\ \# \\ 0 \end{bmatrix} = \begin{bmatrix} R^{-H}(n) \\ \# \\ \# \end{bmatrix}. \tag{22}$$

Combining the updatings for the upper triangular matrix $R(n-1)$, the orthogonalized desired vector $\underline{u}(n-1)$, and the lower triangular matrix $R^{-H}(n-1)$, we have

$$\begin{aligned} \tilde{Q}(n)&\begin{bmatrix} \beta R(n-1) & \vdots & \beta\underline{u}(n-1) & \vdots & \frac{1}{\beta}R^{-H}(n-1) \\ 0 & \vdots & \beta\underline{v}(n-1) & \vdots & \# \\ \underline{x}^T(t_n) & \vdots & z(t_n) & \vdots & 0 \end{bmatrix} \\ &= \begin{bmatrix} R(n) & \vdots & \underline{u}(n) & \vdots & R^{-H}(n) \\ 0 & \vdots & \beta\underline{v}(n-1) & \vdots & \# \\ 0 & \vdots & \# & \vdots & \# \end{bmatrix}. \end{aligned} \tag{23}$$

Thus, the optimal weight vector for the RLS problem can be obtained by substituting Equations 19, 20, and 23 into Equation 8 to give

$$\underline{w}_{LS}(n) = R^{-1}(n)\underline{u}(n). \tag{24}$$

Equation 24 shows how to compute the optimal weight vector using QRD. However, the problem of designing a fully pipelined processor due to the backward substitution still remains. Since $\underline{u}(n)$ and $R^{-H}(n)$ are available, the technique described in Subsection 3.2 can be used to realize the backward substitution. Equation 24 is then computed as follows:

$$\underline{w}_{LS}{}^T(n) = \underline{u}^T(n)R^{-H*}(n), \tag{25}$$

where $\underline{w}_{LS}{}^T(n)$ is a 1 by N vector and T denotes transpose.

1. Initialize Conditions at $n = 0$ by setting

   $R(0) = 0 \quad \underline{z}(0) = 0 \quad R^{-H}(0) = 0$

2. Initialization Procedure for $0 \leq n \leq N$:

   (a) $\left[\begin{array}{ccc} R(N) & \vdots & \underline{u}(N) \end{array} \right]$ is generated by using mode 1 operation, QR decomposition

   (b) $R^{-H}(N)$ is obtained by mode 2 operation, the parallel multiplication and accumulation operation

3. Recursive Procedure for $n > N$ (Mode 1 only):

   (a) $\tilde{Q}(n) \begin{bmatrix} \beta R(n-1) \\ 0 \\ \underline{x}^T(t_n) \end{bmatrix} = \begin{bmatrix} R(n) \\ 0 \\ 0 \end{bmatrix}$

   (b) $\tilde{Q}(n) \begin{bmatrix} \beta \underline{u}(n-1) \\ \beta \underline{v}(n-1) \\ z(t_n) \end{bmatrix} = \begin{bmatrix} \underline{u}(n) \\ \beta \underline{v} \\ \alpha \end{bmatrix}$

   (c) $\tilde{Q}(n) \begin{bmatrix} \frac{1}{\beta} R^{-H}(n-1) \\ \# \\ 0 \end{bmatrix} = \begin{bmatrix} R^{-H}(n) \\ \# \\ \# \end{bmatrix}$

   (d) $\underline{w}_{RLS}{}^T(n) = \underline{u}^T(n) R^{-H^*}(n)$

Table 3: Summary of Parallel/Pipelined QRD-RLS Algorithm

$PE1$	$PE2$	$PE3$	$PE4$
$d \leftarrow (\beta^2 r^2 + \lvert x\rvert^2)^{\frac{1}{2}}$	$y \leftarrow -s\beta r + cx$	$x_{out} \leftarrow -\frac{1}{\beta}sr + cx_{in}$	$x_{out} \leftarrow -\frac{1}{\beta}sr + cx_{in}$
$c \leftarrow \frac{\beta r}{d}$	$r \leftarrow c\beta r + s^* x$	$r \leftarrow \frac{1}{\beta}cr + s^* x_{in}$	$r \leftarrow \frac{1}{\beta}cr + s^* x_{in}$
$s \leftarrow \frac{x}{d}$		$y \leftarrow r$	$w_{out} \leftarrow yr^* + w_{in}$
$r \leftarrow d$			

Table 4: The Mode 1 Operation of MSC

4.2 Systolic MSC Weight Extraction System

The systolic RLS weight extraction algorithm is summarized in Table 3. The parallel/pipelined weight vector obtained by this algorithm is defined only during the recursive updating for $n \geq N$ when the observed data matrix is of full rank. We start with initializing the algorithm by setting the upper triangular matrix, the orthogonalized desired vector, and the lower triangular matrix to zero, i.e., $R(0) = 0$, $\underline{u}(0) = \underline{0}$, and $R^{-H}(0) = 0$. In the initialization, mode 1 and mode 2 are required while only mode 1 is needed in recursive updating. When the observed input data matrix $X(N)$ is available, where N is the number of sensors. Then, the initial upper triangular matrix $R(N)$ and the orthogonalized initial desired vector $\underline{u}(N)$ are generated by using the mode 1 operation, the QRD, and the lower triangular matrix $R^{-H}(N)$ are obtained by employing the mode 2 operation, the parallel multiplication and accumulation operation, as described in Subsection 3.1. Finally, for $n \geq N$, the optimal parallel weights are obtained by using only the mode 1 operation to update parameters and to carry out the parallel multiplication and accumulation instead of backward substitution.

In Figure 5, the MSC systolic parallelogram array processor is illustrated for the case of four sensors to receive the observed input data and the desired input data,

$PE1$	$PE2$	$PE3$	$PE4$
$s \leftarrow \frac{x}{r}$	$y \leftarrow cx - sr$	$y \leftarrow cx - sr$	$x_{out} \leftarrow x_{in}$
$c \leftarrow 1$			$if \;\; x_{in} \leftarrow 1$
			$then \;\; r \leftarrow s*$

Table 5: The Mode 2 Operation of MSC

and to instantaneously generate the optimal updated weights in parallel. The system needs two procedures which are the initialization and the recursive updating. The initialization is further divided into two parts. First, under the mode 1 operation, the 3×3 observed input data X, the 3×1 desired input data $\underline{z}$, and the 3×3 zero matrix are fed into the MSC systolic array to compute the 3×3 upper triangular matrix R and the 3×1 orthogonalized desired vector $\underline{u}$ stored in the upper left part of the systolic parallelogram array processor. Secondly, under the mode 2 operation, the 3×3 identity matrix, 1×3 matrix of 1's, and zeroes are sent into the processor to generate the 3×3 lower triangular matrix R^{-H} which is stored in the lower right triangular part of the array processor. The upper left triangular processors perform the parallel multiplication and accumulation operation instead of forward substitution to generate the lower triangular matrix R^{-H} when 3×3 identity matrix is received, and the lower right triangular processors function as the loading operation when 1×3 matrix of 1's is received. Finally, during recursive updating, the optimal weight vector is obtained in parallel under the mode 1 operation. When the 1×3 observed input data vector, the desired input data, and 1×3 zero vector are fed into the processor, the 1×3 updated optimal weight vector is obtained instantaneously at the bottom of the array.

The four processor elements of the systolic array are given in Figure 6. The mode 1 operation for the MSC systolic array for each processor element is described in

Table 4, and the mode 2 operation is presented in Table 5. The mode 1 operation is used to carry out the QRD and parallel multiplication in both initialization and recursive updating. Under mode 1 operation, the processor element 1 generates the rotation coefficients c and s when zeroing out the observed input data. The processor elements 2 and 3 perform the rotation of the received input data according to the rotation coefficients. The processor element 4 not only perform the rotation but also carries out the parallel multiplication and accumulation operation to compute the optimal weights. The mode 2 operation is employed to carry out the parallel multiplication and accumulation operation without computing the forward substitution to generate the lower triangular matrix $R^{-H}(n)$. The processor element 1 of the mode 2 operation is used to generate parameters while the processor 2 and 3 are employed to carry out the parallel multiplication and accumulation operation. The processor element 4 is simply used to store the lower triangular matrix. It is illustrated in Figure 5 that the two different modes described in Table 4 and 5 are used in the initialization. The recursive updating for parallel/pipelined weight extraction used for $n > N$ only requires mode 1 operation. Since the optimal weight vector obtained in the recursive updating requires only one mode, the parallel MSC systolic parallelogram array processor proposed is fully pipelined.

5 QRD-MVDR Adaptive Beamformer

Since weight extraction in the conventional QR-based constrained recursive least squares (QRD-CRLS) algorithms requires both recursive orthogonalization and backward substitution [7, 11, 12], it is impossible to update the whole system in a fully pipelined manner. The new QRD-CRLS technique proposed is able to obtain the optimal weights without performing backward substitution.

5.1 QR Based CRLS Algorithm for MVDR Beamformer

In this subsection, a parallel and fully pipelined algorithm is introduced for weight extraction in a systolic MVDR beamformer. When the N-snapshot observed input data matrix $X(N)$ is available, by applying the QRD to the observed data $X(N)$, the initial upper triangular matrix $R(N)$ is given by

$$R(N) = Q(N)X(N), \tag{26}$$

where $X(N)$ is an N by N observed data matrix consisting of N row vectors and each row vector is a snapshot of N sensors, and Q is an N by N unitary matrix. The initial parameter vector $\underline{s}^i(N)$ is defined by

$$\underline{s}^i(N) = R^{-H}(N)\underline{c}^i, \tag{27}$$

where $\underline{c}^i$ is the steering vector defined in Equation 11. The initial lower triangular matrix $R^{-H}(N)$ is obtained by replacing the steering vector in Equation 27 by an identity matrix. It is shown in Subsection 3.1 that the initial parameter vector $\underline{s}^i(N)$ and the initial lower triangular matrix $R^{-H}(N)$ can be computed by using the systolic parallelogram array processor described without computing the forward substitution.

When n is greater than N, the number of sensors, it has been shown in Subsection 4.1 that a QRD can be carried out recursively to update the optimal weights. The upper triangular matrix can be updated as before, i.e.

$$Q(n)X(n) = \check{Q}(n)\begin{bmatrix} \beta R(n-1) \\ 0 \\ \cdots \\ \underline{x}^T(t_n) \end{bmatrix} = \begin{bmatrix} R(n) \\ 0 \\ \cdots \\ 0 \end{bmatrix}. \tag{28}$$

To update a parameter vector $\underline{s}^i(n-1)$, notice that

$$\underline{c}^i \;=\; R^H(n)\underline{s}^i(n)$$

$$= R^H(n-1)\underline{s}^i(n-1)$$

$$= \begin{bmatrix} \beta R^H(n-1) & 0 & \underline{x}^*(t_n) \end{bmatrix} \tilde{Q}^H(n)\tilde{Q}(n) \begin{bmatrix} \frac{1}{\beta}\underline{s}^i(n-1) \\ \# \\ 0 \end{bmatrix}. \qquad (29)$$

Therefore,

$$\tilde{Q}(n) \begin{bmatrix} \frac{1}{\beta}\underline{s}^i(n-1) \\ \# \\ 0 \end{bmatrix} = \begin{bmatrix} \underline{s}^i(n) \\ \# \\ \# \end{bmatrix}. \qquad (30)$$

As described in the MSC system, the same unitary matrix used to update the upper triangular matrix $R(n-1)$ can also be employed to update the lower triangular matrix $R^{-H}(n-1)$, that is

$$\tilde{Q}(n) \begin{bmatrix} \frac{1}{\beta}R^{-H}(n-1) \\ \# \\ 0 \end{bmatrix} = \begin{bmatrix} R^{-H}(n) \\ \# \\ \# \end{bmatrix}. \qquad (31)$$

Updating for the upper triangular matrix $R(n-1)$, the parameter vector $\underline{s}^i(n-1)$, and the lower triangular matrix $R^{-H}(n-1)$ can be described by the combined equation

$$\tilde{Q}(n) \begin{bmatrix} \beta R(n-1) & \vdots & \frac{1}{\beta}\underline{s}^i(n-1) & \vdots & \frac{1}{\beta}R^{-H}(n-1) \\ 0 & \vdots & \# & \vdots & \# \\ \underline{x}^T(t_n) & \vdots & 0 & \vdots & 0 \end{bmatrix}$$
$$= \begin{bmatrix} R(n) & \vdots & \underline{s}^i(n) & \vdots & R^{-H}(n) \\ 0 & \vdots & \# & \vdots & \# \\ 0 & \vdots & \# & \vdots & \# \end{bmatrix}. \qquad (32)$$

For convenience, let

$$\underline{\tilde{w}}^{i^T}(n) = \underline{s}^{i^T}(n)R^{-H^*}(n). \qquad (33)$$

On substituting Equations 28, 29, and 33 into 13, the MVDR weight vector $\underline{w}^{iT}_{CRLS}(n)$ becomes

$$\underline{w}^{iT}_{CRLS}(n) = \frac{r^i}{|\underline{s}^i(n)|^2}\underline{\tilde{w}}^{iT}(n), \quad for \;\; i = 1, \cdots, K. \tag{34}$$

5.2 Systolic MVDR Weight Extraction System

A single fully pipelined structure is shown in Figure 7 for the case of three sensors with one constraint. Another single fully pipelined structure is shown in Figure 8 for three sensors with multiple constraints. There are five processor elements as shown in Figure 9 for our proposed fully pipelined structure. Both parallel weight extraction MVDR systems require two procedures: the initialization and recursive updating. The initialization can be further divided into two modes. Under the mode 1 operation, the 3×3 upper triangular matrix R is generated and stored in processor elements 1 and 2, when the 3×3 input observed matrix and the 4×3 zero matrix are received. Then, under the mode 2 operation described in Subsection 3.1 functioned as the parallel multiplication and accumulation operation instead of forward substitution, the 3×1 initial parameter vector $\underline{s}$ and the 3×3 lower triangular matrix R^{-H} are obtained and stored in processor elements 3 and 4, when a 3×1 steering vector and a 3×3 identity matrix on the left, and a 1×4 matrix of 1's and zeroes on the right are received. Finally, in recursive updating, the systolic parallelogram array processor performs updating, accumulating and adding, multiplying, and normalizing operations to compute the optimal weights when a 3×1 observed input data and zeros are received.

To demonstrate how the parallel/pipelined weight extraction system functions, the summary of the whole system to obtain the optimal weight vector for MVDR adaptive beamforming is described in Table 6. The mode 1 operation and the mode 2 operation for each processor element are given in Table 7 and Table 8. Under the

1. Initialize Conditions at $n = 0$ by setting

   $R(0) = 0 \quad \underline{s}(0) = 0 \quad R^{-H}(0) = 0$

2. Initialization Procedure for $0 \leq n \leq N$:

   (a) $R(N)$ is generated by using mode 1 operation, QR decomposition

   (b) $\underline{s}^i(N)$ and $R^{-H}(N)$ are obtained by mode 2 operation, the parallel multiplication and accumulation operation

3. Recursive Procedure for $n > N$ (Mode 1 only):

   (a) $\tilde{Q}(n)\begin{bmatrix} \beta R(n-1) \\ 0 \\ \underline{x}^T(t_n) \end{bmatrix} = \begin{bmatrix} R(n) \\ 0 \\ 0 \end{bmatrix}$

   (b) $\tilde{Q}(n)\begin{bmatrix} \frac{1}{\beta}\underline{s}^i(n-1) \\ \# \\ 0 \end{bmatrix} = \begin{bmatrix} \underline{s}^i(n) \\ \# \\ \# \end{bmatrix}$

   (c) $\tilde{Q}(n)\begin{bmatrix} \frac{1}{\beta}R^{-H}(n-1) \\ \# \\ 0 \end{bmatrix} = \begin{bmatrix} R^{-H}(n) \\ \# \\ \# \end{bmatrix}$

   (d) $\underline{\tilde{w}}^{i^T}(n) = \underline{s}^{i^T}(n)R^{-H^*}(n)$

   (e) $\underline{w}^T(n) = \frac{r^i}{|\underline{s}^i(n)|^2}\underline{\tilde{w}}^{i^T}(n)$

Table 6: Summary of Parallel/Pipelined QRD-CRLS Algorithm

$PE1$	$PE2$	$PE3$	$PE4$
$d \leftarrow (\beta^2 r^2 + \|x\|^2)^{\frac{1}{2}}$	$y \leftarrow -s\beta r + cx$	$x_{out} \leftarrow -\frac{1}{\beta} sr + cx_{in}$	$x_{out} \leftarrow -\frac{1}{\beta} sr + cx_{in}$
$c \leftarrow \frac{\beta r}{d}$	$r \leftarrow c\beta r + s^* x$	$r \leftarrow \frac{1}{\beta} cr + s^* x_{in}$	$r \leftarrow \frac{1}{\beta} cr + s^* x_{in}$
$s \leftarrow \frac{x}{d}$		$y \leftarrow r$	$w_{out} \leftarrow yr^* + w_{in}$
$r \leftarrow d$		$\eta_{out} \leftarrow \|r\|^2 + \eta_{in}$	
$PE5$		$w_{out} \leftarrow \frac{r w_{in}}{\beta^2 \eta}$	

Table 7: The Mode 1 Operation of MVDR

$PE1$	$PE2$	$PE3$	$PE4$	$PE5$
$s \leftarrow \frac{x}{r}$	$y \leftarrow cx - sr$	$x_{out} \leftarrow x_{in}$	$x_{out} \leftarrow x_{in}$	$w_{out} \leftarrow w_{in}$
$c \leftarrow 1$		$if \;\; x_{in} \leftarrow 1$	$if \;\; x_{in} \leftarrow 1$	
		$then \;\; r \leftarrow s*$	$then \;\; r \leftarrow s*$	

Table 8: The Mode 2 Operation of MVDR

mode 1 operation, the processor element 1 generates the rotation coefficients c and s, while processor elements 2, 3, and 4 rotate the received data. The processor elements 3 and 4 also performs the multiplication-and-accumulation operation to compute the normalization coefficient and optimal weight vector before normalization. The processor element 5 performs the normalization for the optimal weights. Under mode 2 operation, the processor elements 1 and 2 perform the parallel multiplication and accumulation operation without exactly computing the forward substitution while the remaining processor elements operate as temporary storage for the parameter vector and the lower triangular matrix. The system is started by setting the whole parallelogram array processor to zero, i.e., $R(0) = 0$, $\underline{s}(0) = 0$, and $R^{-H}(0) = 0$ at time $n = 0$. The initialization is then performed during time $0 \leq n \leq N$ to obtain the initial upper triangular matrix $R(N)$, the initial parameter vector $\underline{s}(N)$, and the initial lower triangular matrix $R^{-H}(N)$. Two different modes described are employed in the initialization. The upper triangular matrix $R(N)$ is generated by using mode 1 operation while the initial parameter vector $\underline{s}(N)$ and the initial lower triangular matrix $R^{-H}(N)$ are obtained by using mode 2 operation described in Subsection 3.1. In recursive updating, the systolic parallelogram array processor uses mode 1 operation to update and compute the optimal weights in parallel during time $n > N$.

6 Fast Givens Based Adaptive Beamforming Systems

It is important in implementing the QR decomposition by VLSI to avoid the square root [8, 18]. Since computation of the square root is complicated, it limits the throughput of the VLSI processors. To speed up the system, a square-root-free version of the Givens method is essential and is referred to as the *fast Givens method* by Gentleman

[19] and Hammarling [20].

6.1 Fast Givens Based RLS Algorithm for MSC

The upper triangular matrix $R(n)$ can be rewritten as

$$R(n) = D^{\frac{1}{2}}(n)\overline{R}(n) = Q(n)X(n) \tag{35}$$

where $D(n)$ is a N by N diagonal matrix with the form

$$\begin{bmatrix} d_1(t_n) & 0 & \cdots & 0 \\ 0 & d_2(t_n) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_N(t_n) \end{bmatrix},$$

and $\overline{R}(n)$ is an upper triangular matrix with the form

$$\begin{bmatrix} 1 & r_{12}(t_n) & r_{13}(t_n) & \cdots & r_{1N}(t_n) \\ 0 & 1 & r_{23}(t_n) & \cdots & r_{2N}(t_n) \\ 0 & 0 & 1 & \cdots & r_{3N}(t_n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

In the QRD, we can also factor the diagonal matrix $D^{\frac{1}{2}}$ out of the orthogonalized desired vector $\underline{u}$, and a lower triangular matrix $R^{-H}(n)$ as follows

$$\underline{u}(n) = D^{\frac{1}{2}}\overline{\underline{u}}(n), \tag{36}$$

and

$$R^{-H}(n) = D^{\frac{1}{2}}\overline{R}^{-H}(n). \tag{37}$$

Then, it can be readily seen that the LS weight vector can be computed without a square root as

$$\underline{w}(n) = \overline{R}^{-1}(n)\overline{\underline{u}}(n), \tag{38}$$

where $D^{\frac{1}{2}}$ is never explicitly computed.

In the initialization the upper triangular matrix $\overline{R}(N)$, and orthogonalized desired vector $\underline{\overline{u}}(N)$ can be generated without explicitly computing $D^{\frac{1}{2}}$

$$Q(N)\left[\begin{array}{ccc} X(N) & \vdots & \underline{y}(N) \end{array}\right] = D^{\frac{1}{2}}(N)\left[\begin{array}{ccc} \overline{R}(N) & \vdots & \underline{\overline{u}}(N) \end{array}\right]. \tag{39}$$

Similar to the last two sections, the initial lower triangular matrix $\overline{R}^{-H}(N)$ can also be obtained easily.

In order to update the optimal weight vector recursively, a vector $\underline{\overline{u}}(n-1)$ and a lower triangular matrix $\overline{R}^{-H}(n-1)$ must be updated at each new data sample. The following equation shows how to update all of the parameters together

$$\check{Q}(n)\left[\begin{array}{ccccc} \beta D^{\frac{1}{2}}(n-1)\overline{R}(n-1) & \vdots & \beta D^{\frac{1}{2}}(n-1)\underline{\overline{u}}(n-1) & \vdots & \frac{1}{\beta}D^{\frac{1}{2}}(n-1)\overline{R}^{-H}(n-1) \\ 0 & \vdots & \beta\underline{\overline{v}}(n-1) & \vdots & \# \\ \underline{x}^T(t_n) & \vdots & y(t_n) & \vdots & 0 \end{array}\right]$$
$$= \left[\begin{array}{ccccc} D^{\frac{1}{2}}(n)\overline{R}(n) & \vdots & D^{\frac{1}{2}}(n)\underline{\overline{u}}(n) & \vdots & D^{\frac{1}{2}}(n)\overline{R}^{-H}(n) \\ 0 & \vdots & \beta\underline{\overline{v}}(n-1) & \vdots & \# \\ 0 & \vdots & \# & \vdots & \# \end{array}\right], \tag{40}$$

where # denotes an arbitrary matrix or vector with no special interest. Finally, the optimal weight vector is then computed by

$$\underline{w}^T(n) = \underline{\overline{u}}^T(n)\overline{R}^{-H*}(n). \tag{41}$$

The square-root-free algorithm and its systolic array processor are described in Table 9 and Figure 10. The proposed fast MSC weight extraction system consists of four processor elements as given in Figure 11. In Table 10, the mode 1 operation performs the fast Givens rotations and parallel multiplication without computing backward substitution. In addition, under the mode 1 operation, the PE 1 is used to generate parameters without computing the square root, the PE 2 and 3 are operated

1. Initialize Conditions at $n = 0$ by setting

   item $D^{\frac{1}{2}}(0)\overline{R}(0) = 0 \quad \underline{\overline{u}}(0) = 0 \quad \overline{R}^{-H}(0) = 0$

2. Initialization Procedure for $0 \leq n \leq N$:

   (a) $Q(N)\left[\begin{array}{ccc} X(N) & \vdots & \underline{z}(N) \end{array}\right] = D^{\frac{1}{2}}(N)\left[\begin{array}{ccc} \overline{R}(N) & \vdots & \underline{\overline{u}}(N) \end{array}\right]$ (Mode 1)

   (b) $\overline{R}^{-H}(N)$ is generated by Mode 2 operation

3. Recursive Procedure for $n > N$ (Mode 1 only):

   (a) $$\check{Q}(n)\begin{bmatrix} \beta D^{\frac{1}{2}}(n-1)\overline{R}(n-1) \\ 0 \\ \underline{x}^T(t_n) \end{bmatrix} = \begin{bmatrix} D^{\frac{1}{2}}(n)\overline{R}(n) \\ 0 \\ 0 \end{bmatrix}$$

   (b) $$\check{Q}(n)\begin{bmatrix} \beta D^{\frac{1}{2}}(n-1)\underline{\overline{u}}(n-1) \\ \# \\ z(t_n) \end{bmatrix} = \begin{bmatrix} D^{\frac{1}{2}}(n)\underline{\overline{u}}(n) \\ \# \\ \# \end{bmatrix}$$

   (c) $$\tilde{Q}(n)\begin{bmatrix} \frac{1}{\beta}D^{\frac{1}{2}}(n-1)\overline{R}^{-H}(n-1) \\ \# \\ 0 \end{bmatrix} = \begin{bmatrix} D^{\frac{1}{2}}(n)\overline{R}^{-H}(n) \\ \# \\ \# \end{bmatrix}$$

   (d) $\underline{w}^T(n) = \underline{\overline{u}}^T(n)\overline{R}^{-H^*}(n)$

Table 9: Summary of Fast Givens-RLS Algorithm

$PE1$	$PE2$	$PE3$	$PE4$
$k \leftarrow \beta^2 r + \delta_{in}\|x\|^2$	$y \leftarrow -c\beta r + x$	$x_{out} \leftarrow -c\beta r + x_{in}$	$x_{out} \leftarrow -\frac{1}{\beta}cr + x_{in}$
$\delta_{out} \leftarrow \frac{\beta^2 r \delta_{in}}{k}$	$r \leftarrow \beta r + s^* y$	$r \leftarrow \beta r + s^* x_{out}$	$r \leftarrow \frac{1}{\beta} r + s^* x_{out}$
$c \leftarrow x$		$y \leftarrow \frac{r}{k}$	$w_{out} \leftarrow y r^* + w_{in}$
$s \leftarrow \frac{\delta_{in} x}{k}$			
$r \leftarrow k$			

Table 10: The Mode 1 Operation of Fast MSC

$PE1$	$PE2$	$PE3$	$PE4$
$s \leftarrow x$	$y \leftarrow cx - sr$	$x_{out} \leftarrow x_{in}$	$x_{out} \leftarrow x_{in}$
$c \leftarrow 1$			$if \;\; x_{in} \leftarrow 1$
			$then \;\; r \leftarrow s*$

Table 11: The Mode 2 Operation of Fast MSC

to transform the input data by those parameters, and the PE 4 is employed not only to transform input data but also to carry out the accumulation and addition operation for obtaining the optimal weights. In Table 11, the mode 2 operation functions as the parallel multiplication and accumulation operation without the need for forward substitution. The functions of the four processor elements in the fast MSC systolic array processor are performed in the same way as in the MSC systolic array processor described in Section 4.2.

6.2 Fast Givens Based CRLS Algorithm for MVDR

Similarly, for MVDR beamforming we can factor $D^{\frac{1}{2}}$ out as follows

$$R(n) = D^{\frac{1}{2}}(n)\overline{R}(n), \tag{42}$$

and

$$\underline{s}(n) = D^{\frac{1}{2}}(n)\underline{\overline{s}}(n), \tag{43}$$

and

$$R^{-H}(n) = D^{\frac{1}{2}}(n)\overline{R}^{-H}(n). \tag{44}$$

All the parameters must also be updated for computing the optimal weight vector. The following equation shows how all the parameters can be updated together

$$\tilde{Q}(n)\begin{bmatrix} \beta D^{\frac{1}{2}}(n-1)\overline{R}(n-1) & \vdots & \frac{1}{\beta}D^{\frac{1}{2}}(n-1)\underline{\overline{s}}(n-1) & \vdots & \frac{1}{\beta}D^{\frac{1}{2}}(n-1)\overline{R}^{-H}(n-1) \\ 0 & \vdots & \# & \vdots & \# \\ \underline{x}^T(t_n) & \vdots & 0 & \vdots & 0 \end{bmatrix}$$
$$= \begin{bmatrix} D^{\frac{1}{2}}(n)\overline{R}(n) & \vdots & D^{\frac{1}{2}}(n)\underline{\overline{s}}(n) & \vdots & D^{\frac{1}{2}}(n)\overline{R}^{-H}(n) \\ 0 & \vdots & \# & \vdots & \# \\ 0 & \vdots & \# & \vdots & \# \end{bmatrix}. \tag{45}$$

Finally,

$$\underline{w}(n) = \frac{r^i}{\underline{\overline{s}}(n)D^{-1}(n)\underline{\overline{s}}(n)}\underline{\tilde{w}}(n), \quad for\ i = 1, \cdots, K. \tag{46}$$

1. Initialize Conditions at $n = 0$ by setting

   item $D^{\frac{1}{2}}(0)\overline{R}(0) = 0 \quad \underline{\overline{s}}(0) = 0 \quad \overline{R}^{-H}(0) = 0$

2. Initialization Procedure for $0 \leq n \leq N$:

   (a) $Q(N)X(N) = D^{\frac{1}{2}}(N)\overline{R}(N)$ under mode 1 operation

   (b) $\underline{\overline{s}}(N)$ and $\overline{R}^{-H}(N)$ are obtained by sending the steering vector $\underline{c}$ and a unit matrix into systolic array under mode 2 operation

3. Recursive Procedure for $n > N$ (Mode 1 only):

   (a) $$\tilde{Q}(n)\begin{bmatrix} \beta D^{\frac{1}{2}}(n-1)\overline{R}(n-1) \\ 0 \\ \underline{x}^T(t_n) \end{bmatrix} = \begin{bmatrix} D^{\frac{1}{2}}(n)\overline{R}(n) \\ 0 \\ 0 \end{bmatrix}$$

   (b) $$\tilde{Q}(n)\begin{bmatrix} \frac{1}{\beta}D^{\frac{1}{2}}(n-1)\underline{\overline{s}}(n-1) \\ \# \\ 0 \end{bmatrix} = \begin{bmatrix} D^{\frac{1}{2}}(n)\underline{\overline{s}}(n) \\ \# \\ \# \end{bmatrix}$$

   (c) $$\tilde{Q}(n)\begin{bmatrix} \frac{1}{\beta}D^{\frac{1}{2}}(n-1)\overline{R}^{-H}(n-1) \\ \# \\ 0 \end{bmatrix} = \begin{bmatrix} D^{\frac{1}{2}}(n)\overline{R}^{-H}(n) \\ \# \\ \# \end{bmatrix}$$

   (d) $\underline{w}(n) = \frac{r}{\underline{\overline{s}}^{-H}(n)D^{-1}(n)\underline{\overline{s}}(n)}\overline{R}^{-1}(n)D^{-1}(n)\underline{\overline{s}}(n)$

Table 12: Summary of Fast Givens-CRLS Algorithm

$PE1$	$PE2$	$PE3$	$PE4$
$k \leftarrow \beta^2 r + \delta_{in}\lvert x\rvert^2$	$y \leftarrow -c\beta r + x$	$x_{out} \leftarrow -\frac{1}{\beta}cr + x_{in}$	$x_{out} \leftarrow -\frac{1}{\beta}cr + x_{in}$
$\delta_{out} \leftarrow \frac{\beta^2 r \delta_{in}}{k}$	$r \leftarrow \beta r + s^* y$	$r \leftarrow \frac{1}{\beta}r + s^* x_{out}$	$r \leftarrow \frac{1}{\beta}r + s^* x_{out}$
$c \leftarrow x$		$y \leftarrow r$	$w_{out} \leftarrow \frac{yr^*}{k} + w_{in}$
$s \leftarrow \frac{\delta_{in} x}{k}$		$\eta_{out} \leftarrow \frac{\lvert r\rvert^2}{k} + \eta_{in}$	
$r \leftarrow k$			
$PE5$		$w_{out} \leftarrow \frac{r w_{in}}{\beta^2 \eta}$	

Table 13: The Mode 1 Operation of Fast MVDR

$PE1$	$PE2$	$PE3$	$PE4$	$PE5$
$s \leftarrow x$	$y \leftarrow cx - sr$	$x_{out} \leftarrow x_{in}$	$x_{out} \leftarrow x_{in}$	$w_{out} \leftarrow w_{in}$
$c \leftarrow 1$		$if\ \ x_{in} \leftarrow 1$	$if\ \ x_{in} \leftarrow 1$	
		$then\ \ r \leftarrow \frac{s*}{k}$	$then\ \ r \leftarrow s*$	

Table 14: The Mode 2 Operation of Fast MVDR

The fast Givens-CRLS algorithm and systolic array processor are described in Table 12 and Figures 12 and 13. There are five processor elements in the systolic array. Processor elements are illustrated in Figure 14 and the functions of each processor element under the two modes are also described in Tables 13 and 14.

7 Conclusion

In this paper, we consider the QRD based recursive least squares (RLS) and constrained recursive least squares (CRLS) problems for MSC and MVDR weight extraction systems. Both weight extraction systems require two modes for initialization and one mode for recursive updating. Since the optimal weight vectors obtained for MSC and MVDR beamformers are defined solely in recursive updatings with only one mode, it is shown that the weight extraction systems proposed are fully pipelined to compute the optimal weights instantaneously. Compared to the conventional weight extraction system involving the forward and backward substitutions which may lead to significant obstruction in designing a fully pipelined systolic architecture to update the optimal weights instantaneously, our proposed systolic parallelogram structure is very promising for VLSI implementation. Furthermore, the fast Givens method without square root operation is employed to improve the throughput for parallel weight extraction systems and to increase the operational speed for MSC and MVDR adaptive array systems.

References

[1] W. M. Gentleman and H. T. Kung, "Matrix Triangularization by Systolic Array," *Proc. SPIE*, Real-Time Signal Processing IV, 1981, **298**, pp. 19-26.

[2] H. T. Kung, "Why systolic architectures?" *Computer*, **15**, 37, 1982, pp. 37-45.

[3] M. Moonen and J. Vandewalle, "Recursive Least Squares with Stabilized Inverse Factorization," *Signal Processing*, Vol. 21, 1990, **1**, pp. 1-15.

[4] J. E. Hudson and T. J. Shepherd, "Parallel Weight Extraction by a Systolic Least squares Algorithm," *Proc. SPIE*, Advanced Algorithms and Architectures for Signal Processing IV, 1989, **1152**, pp. 68-77.

[5] J. G. McWhirter, "Algorithmic Engineering - an Emerging Discipline," *Proc. SPIE*, Advanced Algorithms and Architectures for Signal Processing IV, 1989, **1152**, pp. 2-15.

[6] C.F. T. Tang, *Adaptive Array Systems Using QR-Based RLS and CRLS Techniques with Systolic Array Architectures*, Ph.D Thesis Report, Ph.D.91.5, Systems Research Center, University of Maryland, College Park, May, 1991.

[7] R. Schreiber, "Implementation of Adaptive Array Algorithms," *IEEE Trans. Acoust. Speech, Signal Processing*, vol. ASSP-34, NO. 5, Oct. 1986, pp. 1038-1045.

[8] J. G. McWhirter and T. J. Shepherd, "Systolic Array Processor for MVDR Beamforming," *IEE proceedings*, Vol 136, No. 2, April 1989, pp. 75-80.

[9] A. W. Bojanczyk and F. T. Luk, "A Novel MVDR Beamforming Algorithm,"*Proc. SPIE*, Advanced Algorithms and Architectures for Signal Processing II, 1987, **826**, pp. 12-16. .

[10] B. Yang and J.F. Bome, "Systolic Implementation of A General Adaptive Array Processing Algorithm," *Proc. IEEE ICASSP*, April 1988, pp. 2785-2788.

[11] N. L. Owsley, "Systolic Array Adaptive Beamforming," *NUSC* report 7971, 1987.

[12] N. L. Owsley, "Systolic Array Adaptive Beamforming" in Haykin ed., *Selected Topics in Signal Processing*, Pentice-Hall, 1989.

[13] C.F. T. Tang, K.J. R. Liu, and S. A. Tretter, "A VLSI Algorithm and Architecture of CRLS Adaptive Beamforming," in press, *Proc. of the Conf. on Information Sciences and Systems*, March, 1991, pp862-867.

[14] T. J. Shepherd, J. G. McWhirter and J. E. Hudson, "Parallel Weight Extraction from a Systolic Adaptive Beamforming," *Mathematics in Signal Processing II*, 1990.

[15] S. Haykin, *Adaptive Filter Theory*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[16] P. Comon and Y. Robert, "A Systolic Array for Computing BA^{-1}," *IEEE Trans. Acoust. Speech, Signal Processing*, vol. ASSP-35, No. 5 June. 1987, pp. 717-723.

[17] K.J. R. Liu and K. Yao, "Multi-Phase Systolic Algorithms for Spectral Decomposition," Accepted and to Appear in *IEEE Trans. Acoust. Speech, Signal Processing*.

[18] J. G. McWhirter, "Recursive Least-Squares Minimization Using a Systolic Array," *Proc. SPIE*, **431**, Real-Time Signal Processing **VI**, 2983, 1983, pp. 105-112.

[19] W. M. Gentleman, "Least Squares Computations by Givens Transformation without Square Roots," *J.Inst. Maths Applics*, **12**, pp329-336, 1973.

[20] S. Hammarling, "A Note on Modifications to the Givens Plane Rotation," *J. Inst. Maths Applics*, **13**, pp 215-218, 1974.

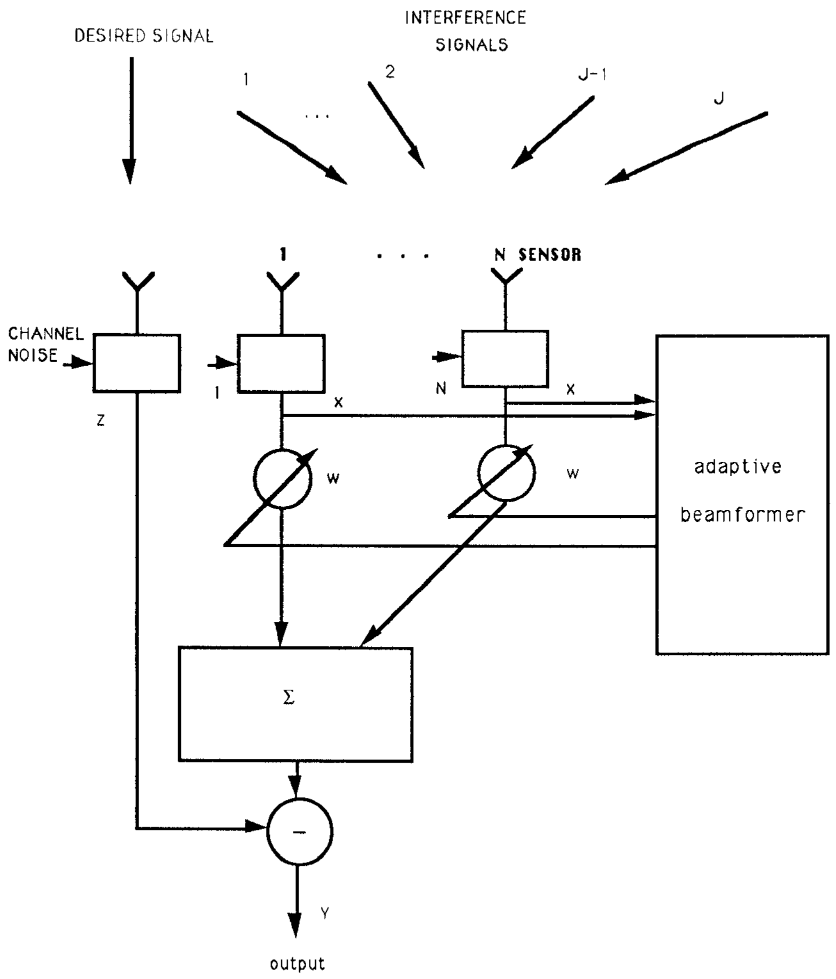

Figure 1: Multiple Sidelobe Canceller (MSC)

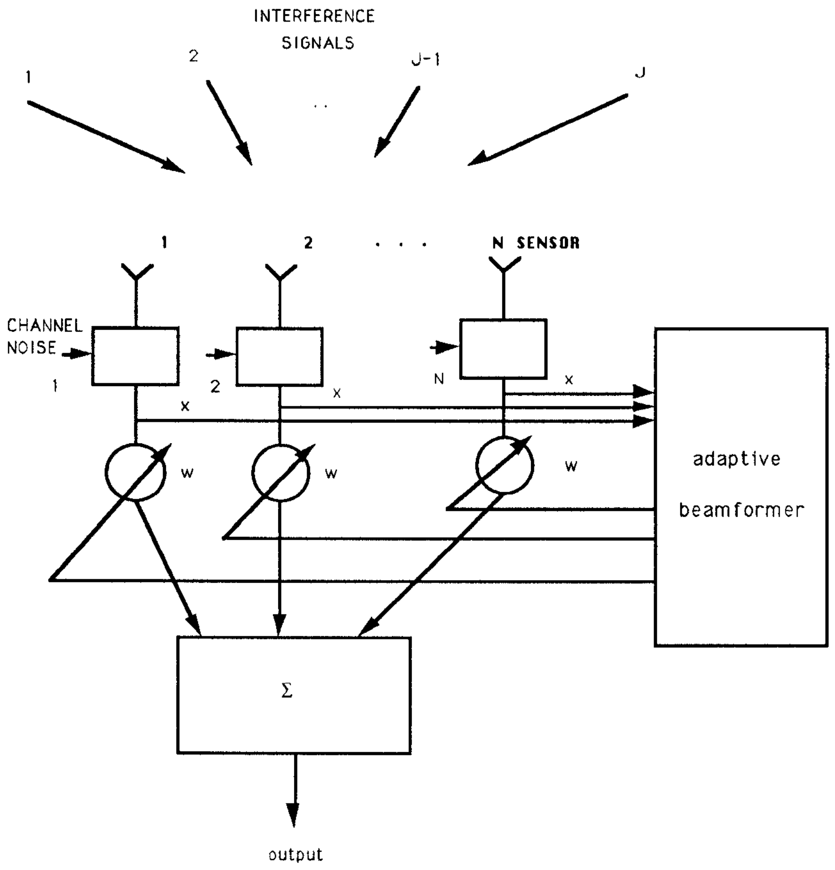

Figure 2: MVDR Adaptive Beamformer

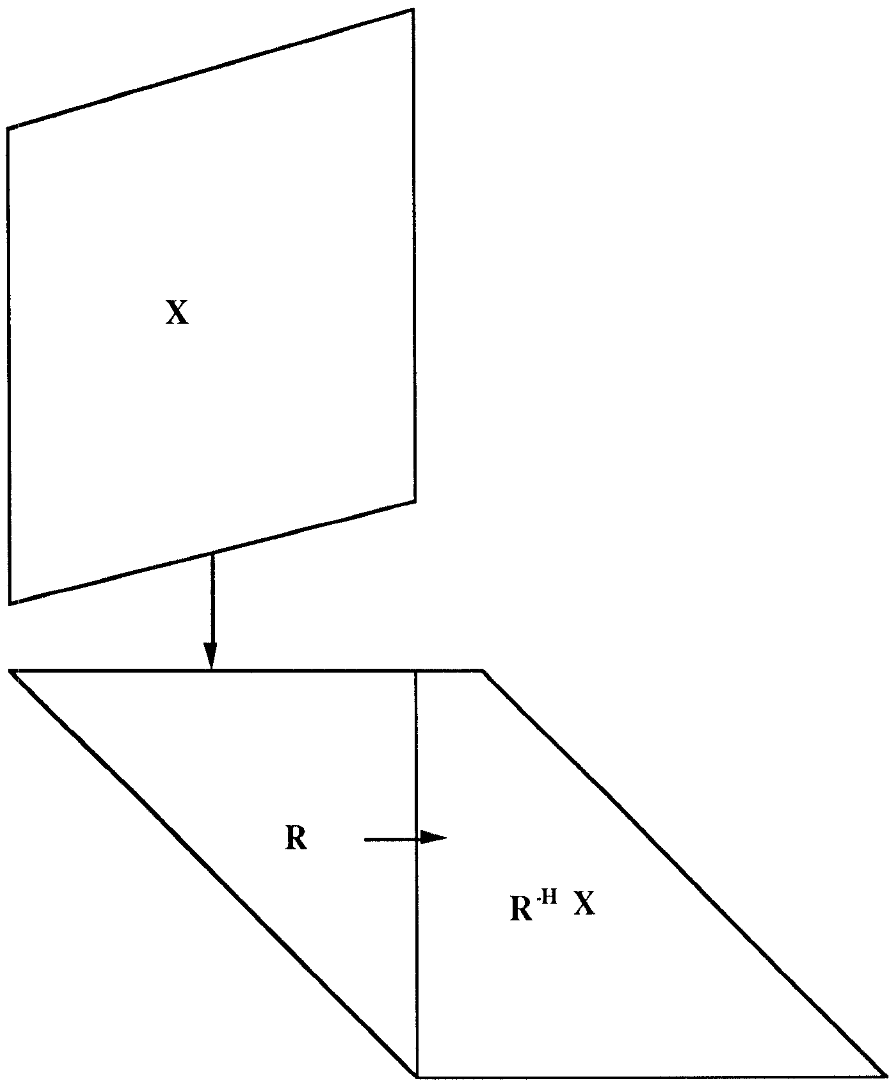

Figure 3: Systolic Array Processors for Forward Substitution

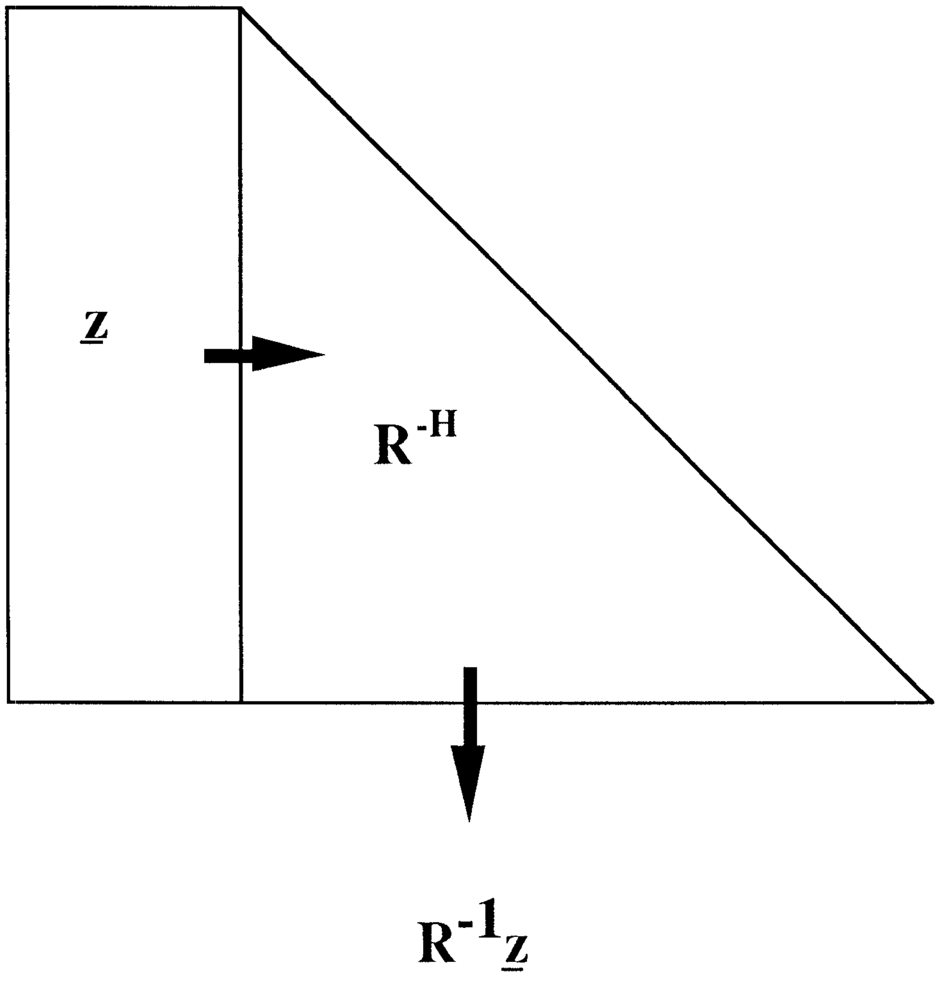

Figure 4: Systolic Array Processors for Backward Substitution

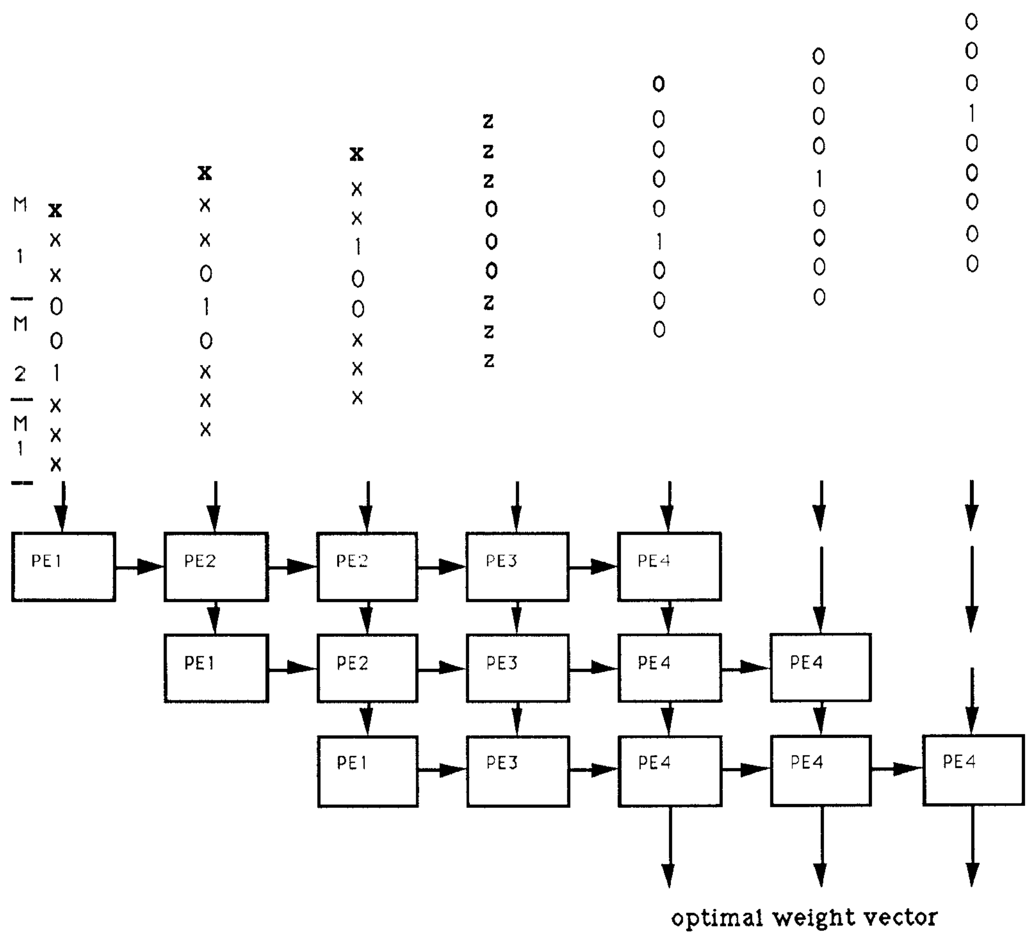

Figure 5: Parallel Weight Extraction for MSC

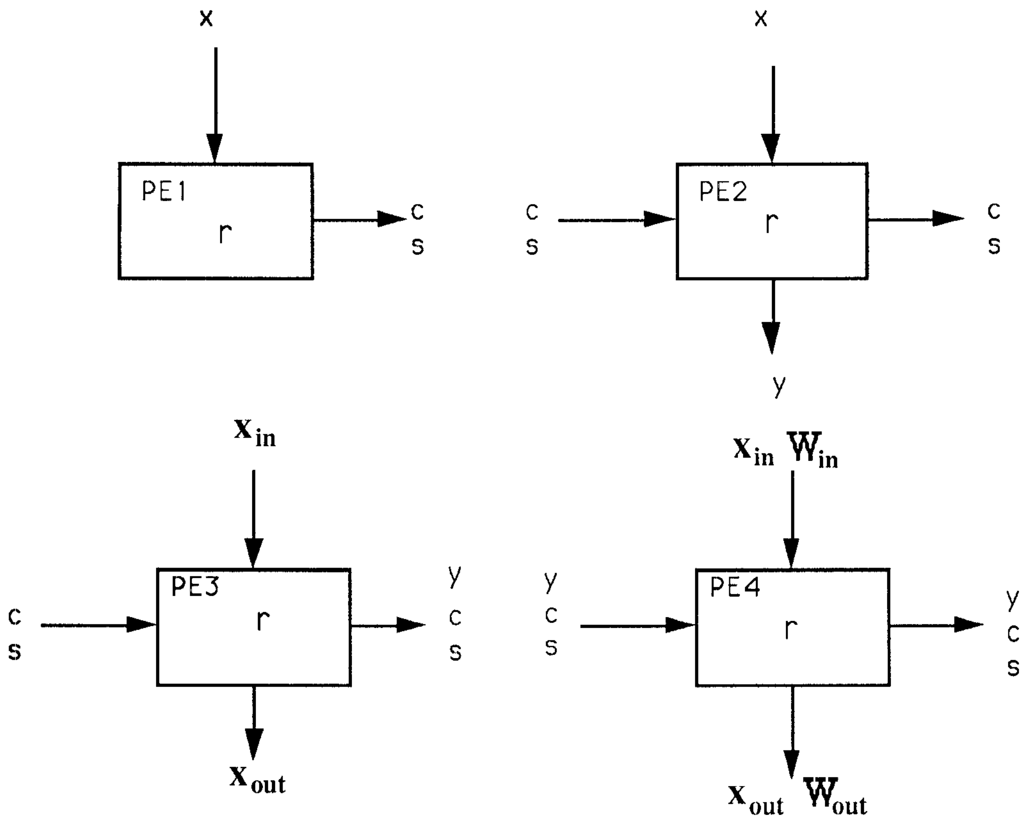

Figure 6: Processor Elements of MSC Systolic Array Processor

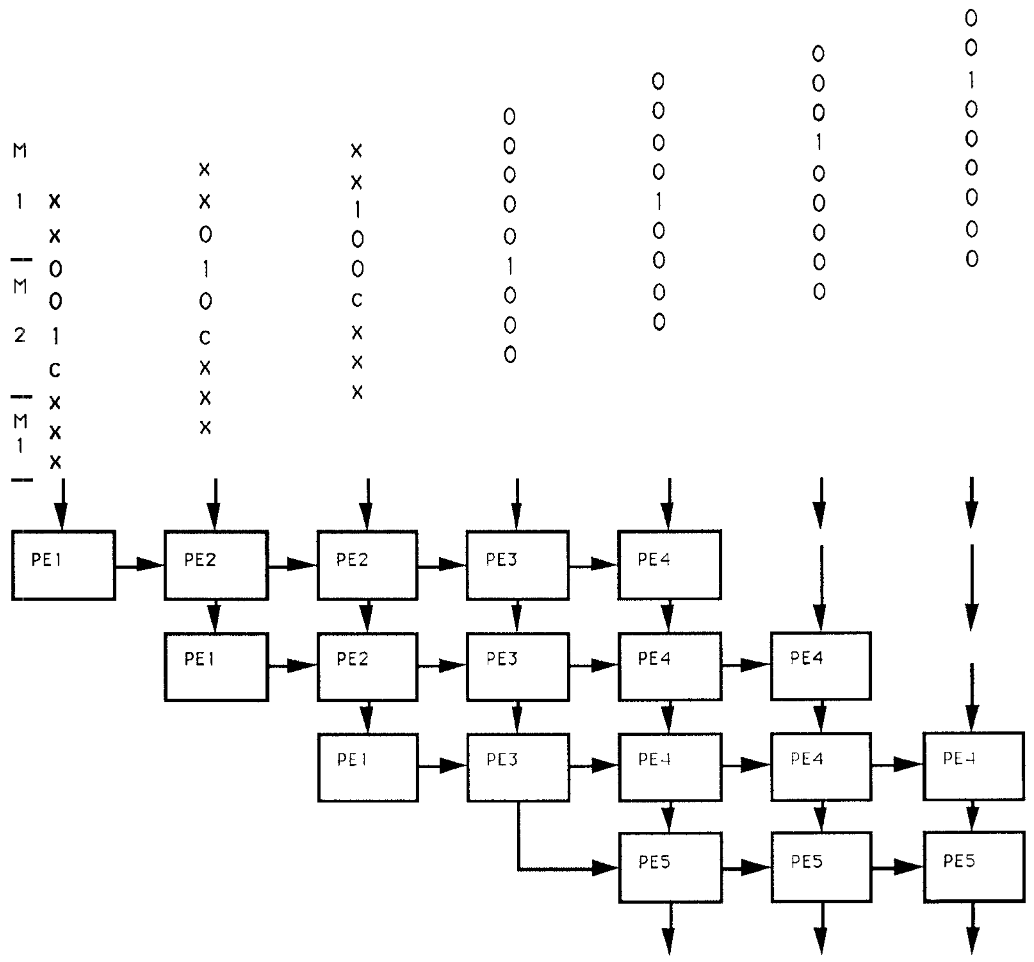

Figure 7: Parallel Weight Extraction for MVDR with One Constraint

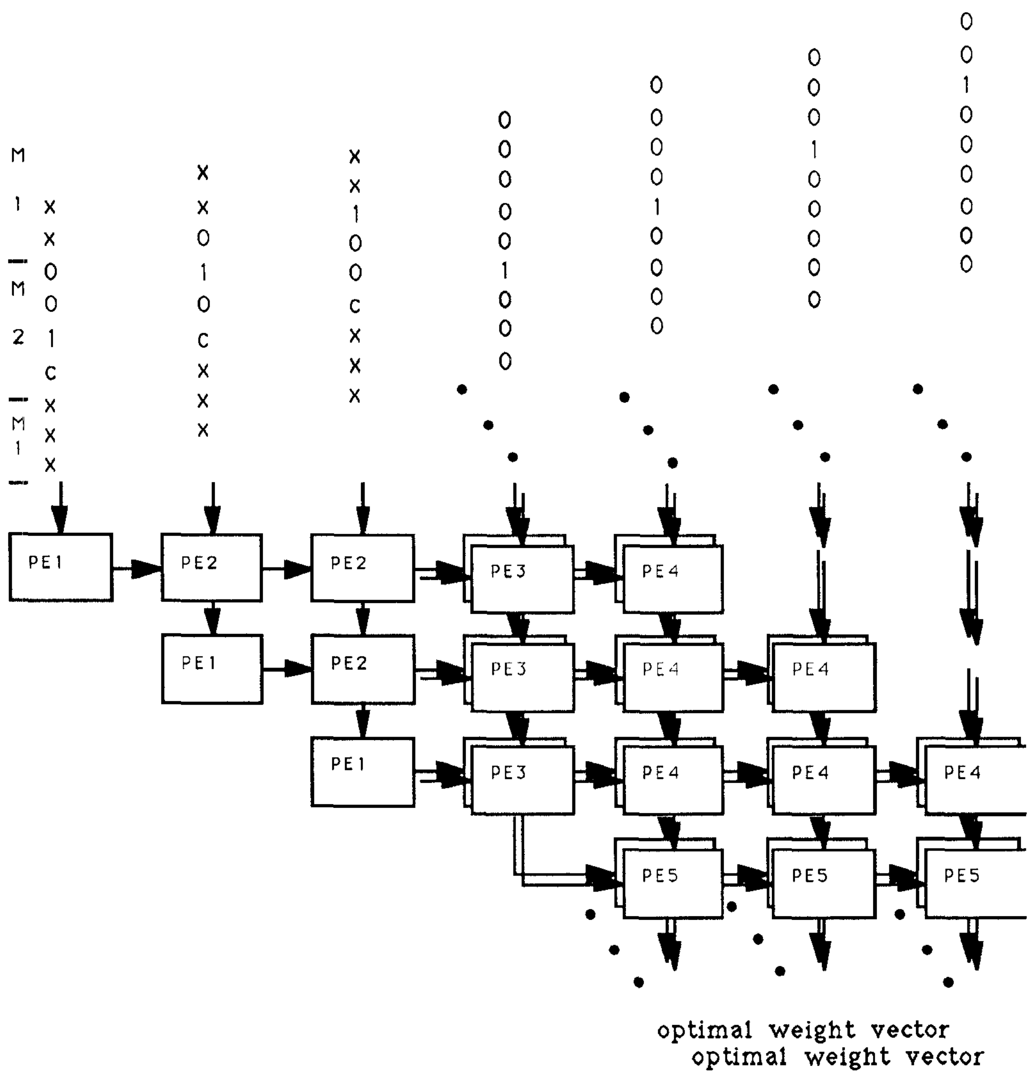

Figure 8: Parallel Weight Extraction for MVDR with Multiple Constraints

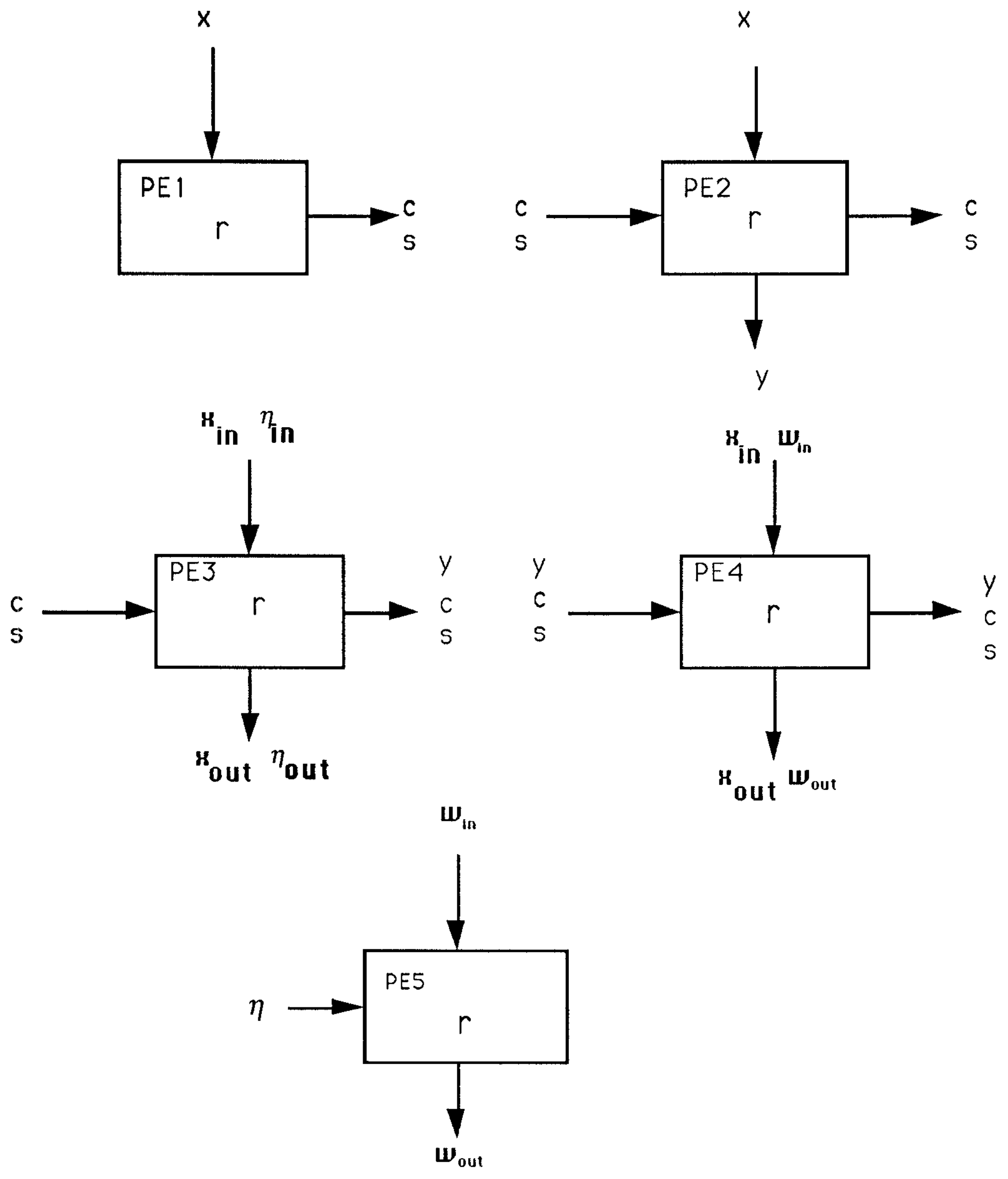

Figure 9: Processor Elements of MVDR Systolic Array Processor

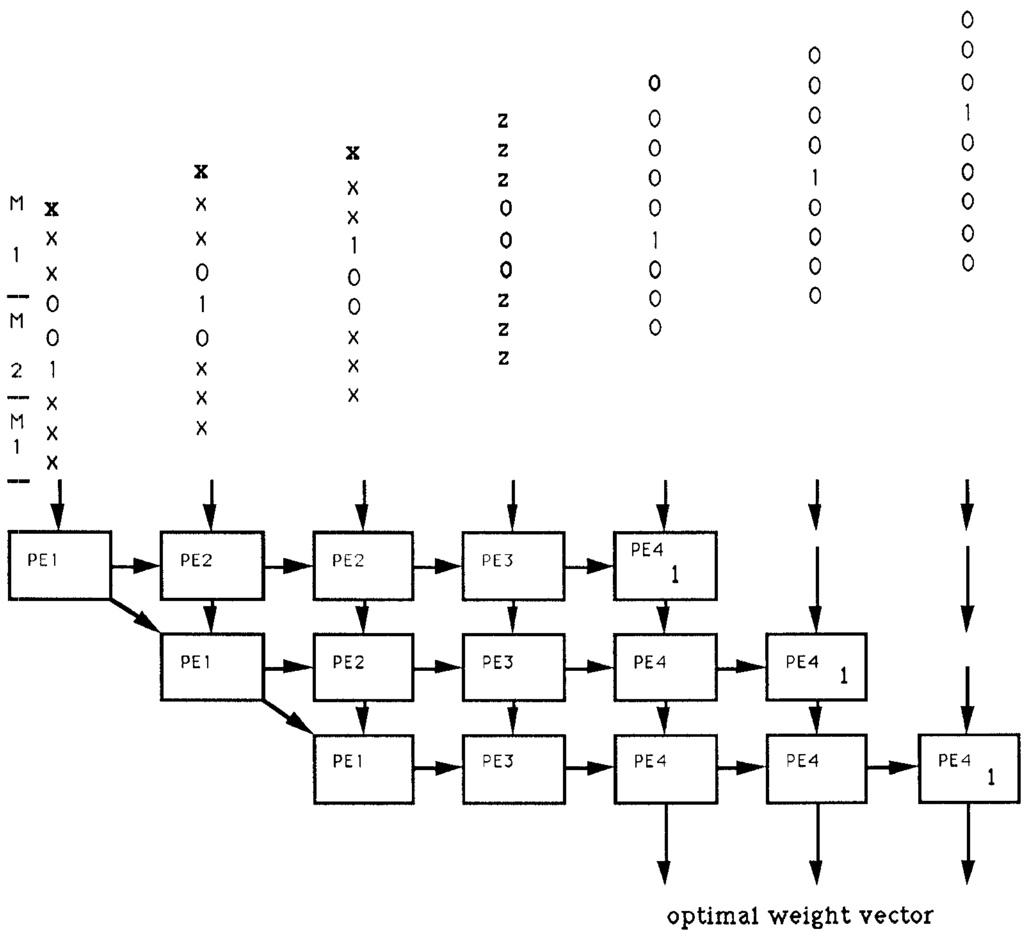

Figure 10: Fast MSC Systolic Array Processor

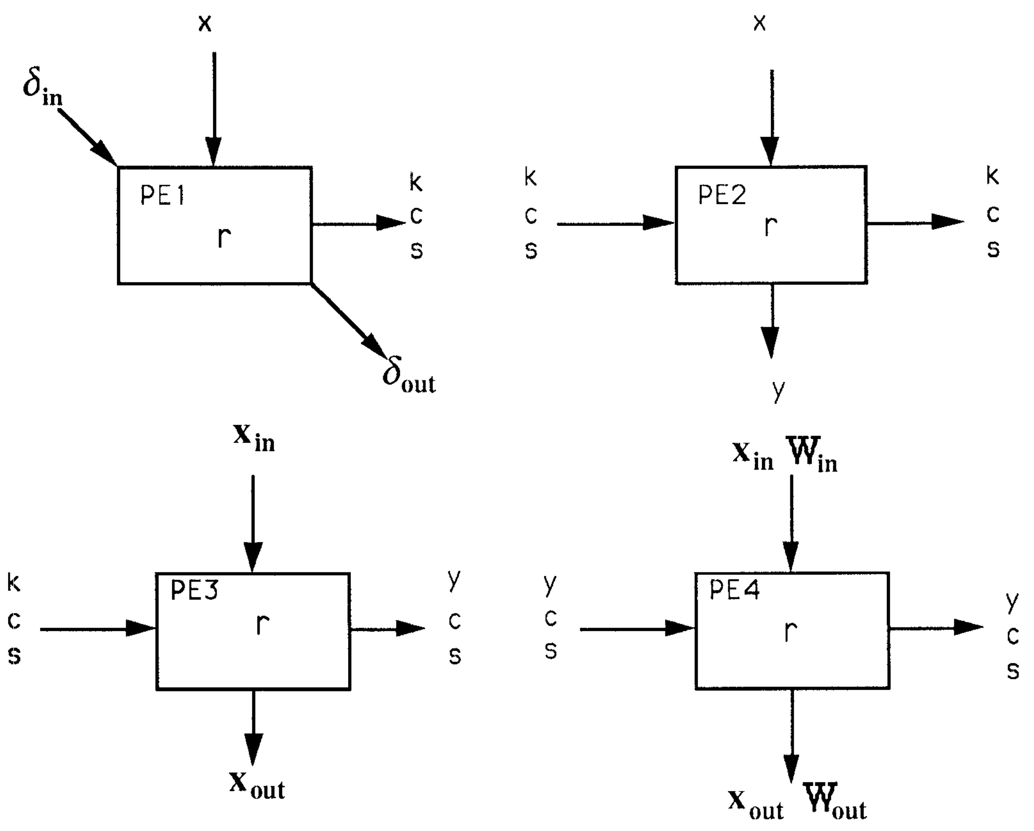

Figure 11: Processor Elements of Fast MSC Systolic Array Processor

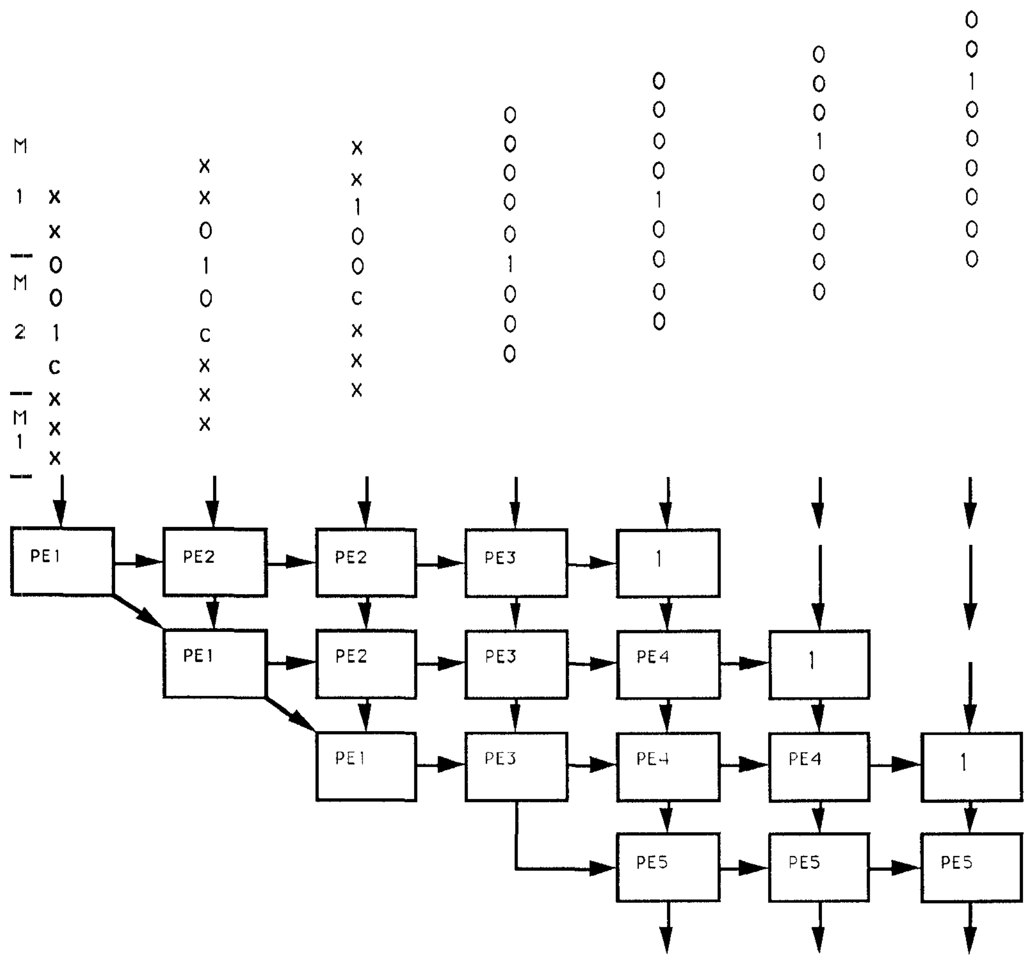

Figure 12: Fast MVDR Systolic Array Processor with One Constraint

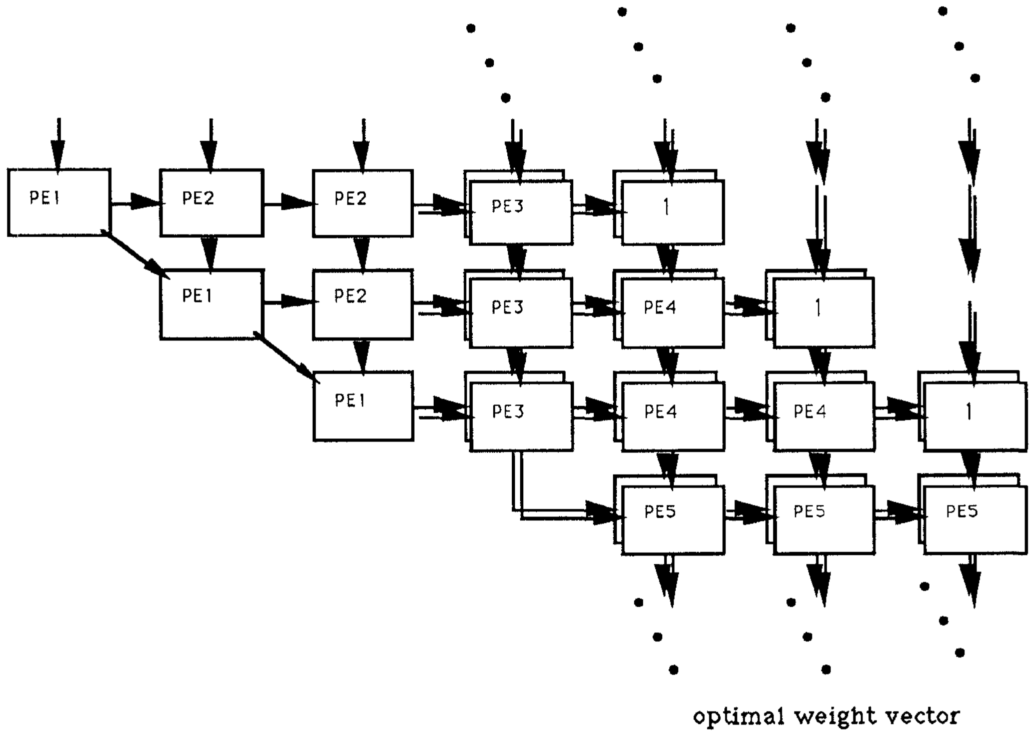

Figure 13: Fast MVDR Systolic Array Processor with Multiple Constraints

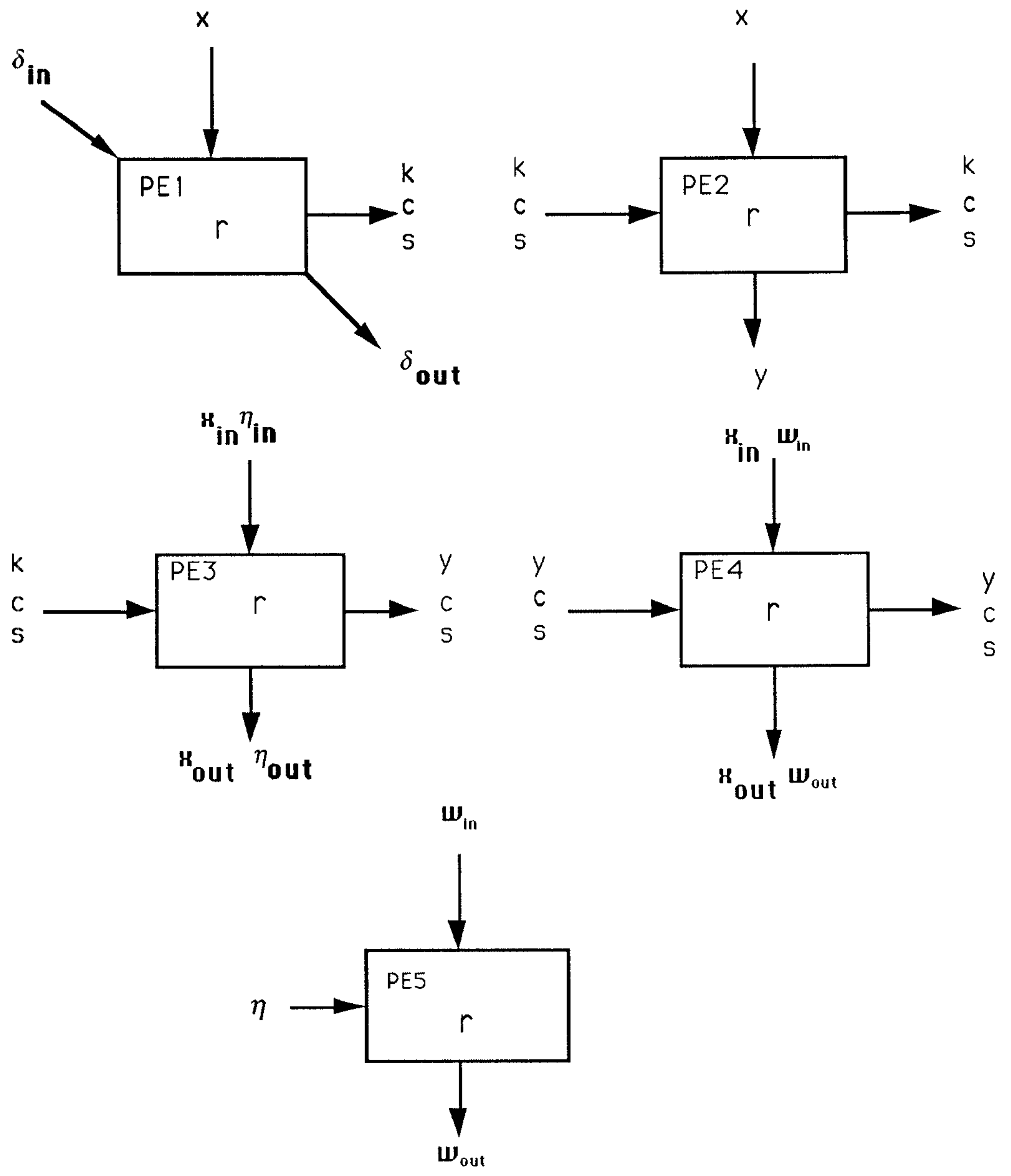

Figure 14: Processor Elements of Fast MVDR Systolic Array Processor