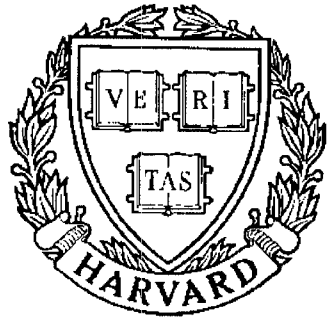
# TECHNICAL RESEARCH REPORT


SYSTEMS RESEARCH CENTER




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

# Dual-State Systolic Architectures for Adaptive Filtering Using Up/Downdating RLS

*by S.F. Hsieh, K.J.R. Liu and K. Yao*

TR 91-47

# Dual-State Systolic Architectures for Adaptive Filtering Using Up/Downdating RLS

**S.F. Hsieh**
Dept. of Communication Engineering
Nat'l Chiao Tung University
Hsinchu, Taiwan 30039

**K.J.R. Liu**
Electrical Engineering Dept.
Systems Research Center
University of Maryland
College Park, MD 20742

**K. Yao**
Electrical Engineering Dept.
UCLA
Los Angeles, CA 90024

## ABSTRACT

We propose a *dual-state systolic* structure to perform joint up/down-dating operations encountered in windowed recursive least squares (RLS) estimation problems. It is derived by successively performing Givens rotations for updating and hyperbolic rotations for downdating. Due to the data independency, a series of Givens and hyperbolic rotations can be interleaved and parallel processing can be achieved by alternatively performing updating and downdating both in time and space. This flip-flop nature of up/down-dating characterizes the feature of dual-state systolic triarray. To further reduce the complexity and increase the throughput rate, Cordic cells can be used to mimic the operations of row-broadcasting and only one control bit is required along each row of processors. Efficient implementation to obtain optimal residuals and a transformation of the hyperbolic rotation to an algebraically equivalent orthogonal operation to provide a more stable implementation are also considered. This systolic architecture is very promising in VLSI implementation of the sliding-window recursive least squares estimations.

# 1 Introduction

Consider a least squares (LS) problem at time $n$,

$$X(n)w(n) \approx y(n), \tag{1}$$

where $X(n)$ is an $\ell \times p$ fixed-windowed data matrix,

$$X(n)=\begin{bmatrix} x_{n-\ell+1,1} & x_{n-\ell+1,2} & \cdots & x_{n-\ell+1,p} \\ x_{n-\ell+2,1} & x_{n-\ell+2,2} & \cdots & x_{n-\ell+2,p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,p} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{n-\ell+1}^T \\ \mathbf{x}_{n-\ell+2}^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix} \in \Re^{\ell \times p}, \tag{2}$$

and $y(n)$ is the desired response vector,

$$y(n) = \begin{bmatrix} y_{n-\ell+1} \\ y_{n-\ell+2} \\ \vdots \\ y_n \end{bmatrix} \in \Re^{\ell}. \tag{3}$$

We denote $\ell$ as the window size, $p$ as the order of the system (i.e., number of sensors in a multichannel filtering case) and $n$ is the time index($n \geq \ell$ is assumed). The LS problem is to find an optimum coefficient vector $\hat{w}(n) \in \Re^p$, such that the Euclidean norm of its associated residual

$$e(n) = X(n)w(n) - y(n) \tag{4}$$

is minimized. If $X(n)$ has full column rank, then it is well known that from the normal equation (NE) approach $\hat{w}(n)$ is given by

$$\hat{w}(n) = (X^T(n)X(n))^{-1}X^T(n)y(n). \tag{5}$$

But the increased dynamic range (because the condition number [1] is squared) precludes the NE method for applications in modern digital signal processing. Therefore, in order to

achieve the same computational precision, direct matrix factorization methods employing orthogonalization to preserve the condition number, like the QR decomposition (QRD), are preferred especially when it is likely that the numerical instability may arise due to ill-conditioning.

In adaptive signal processing, we are not only interested in $\hat{w}(n)$ for any specific time $n$, but also all the subsequent $\hat{w}(t), t \geq n$. QRD has been proved to be an effective tool in performing this recursive LS problem[2, 3]. However, under time-varying conditions, much attention has been focused on schemes employing exponential forgetting factors, while less on fixed-windowed ones. This is partially due to the difficulty of downdating obsolete data encountered in the windowed RLS model. But, fixed-window scheme should not be precluded simply because its computational burden. Other factors, especially fast parameters tracking ability, actually favors this method under some nonstationary conditions. To motivate the need for fixed-window under nonstationary condition, a computer experiment is given to demonstrate the advantage of its faster convergence of the fixed-window method over the method using an exponential forgetting factor.

A second order autoregressive (AR) process $\{u(i)\}$ is given as follows [9, pp. 204-6]. $u(i) + a_1 u(i-1) + a_2 u(i-2) = v(i), i = 1, \ldots, 100$ with $a_1 = -0.9750$ and $a_2 = 0.9500$ and $u(i) + b_1 u(i-1) + b_2 u(i-2) = v(i), i = 101, \ldots, 250$ with $b_1 = -1.5955$ and $b_2 = 0.9500$. $v(\cdot)$ is a white Gaussian noise with standard deviation equal to 0.1, except that from $i = 20$ to 30 $v(i)$'s are intentionally increased by a factor of 20 to account for temporary noisy perturbation. This is equivalent to lower down the SNR to $-6$ dB. There is also a step change at iteration 100 in the parameters of the AR model, i.e., $(a_1, a_2)$ is changed from $(-0.9750, 0.9500)$ to $(b_1, b_2) = (-1.5955, 0.9500)$.

To make a fair comparison between the fixed-window scheme with a window size $\ell$ and an exponentially weighting scheme with a forgetting factor $\lambda$ in the sense that both schemes have the same *self noise*(fluctuation of the estimated parameters with respect to

the optimum AR parameters)[10], we choose that $\ell = 50$ and $\lambda = \sqrt{(\ell-1)/(\ell+1)}$. 100 simulations with different noise realizations have been performed. Fig. 1 depicts the mean bias versus number of iterations in estimating the first AR parameter $a_1$ and $b_1$, before and after the step change at iteration 100 respectively. For fixed-window scheme, convergence is reached after $\ell = 50$ more iterations following the step change at iteration 100, while for exponentially weighting method, 150 more iterations are required.

This comparison shows that a fixed-window scheme is indeed necessary and important in speedy tracking parameters under some nonstationary conditions.

Until recently, efficient downdating algorithms have been proposed [4, 5]. But efficient implementations and architectures of fixed-windowed RLS filtering are still rarely considered. Therefore we propose two systolic arrays, which is suitable for VLSI design, to perform fixed-windowed RLS estimation. The first one is denoted as the *dual-state* systolic triarray, which resembles Gentleman and Kung's triarray [2] with the same hardware complexity, except the clock rate of the processor is two times higher. The second one is realized by using Cordic cells to reduce the hardware complexity. Also, efficient scheme to obtain optimal residual has not yet been addressed for the windowed RLS estimation. We will show what can and cannot be obtained by using the systolic implementation. A transformation of the hyberbolic rotation to a more stable orthogonal operation is also considered in this paper.

In section 2, the basic up/downdating RLS estimation is considered, followed by the dual-state systolic architecture in Section 3 and Cordic processors implementation in Section 4. In Section 5, we consider the recursive estimation of optimal residual from systolic implementation. Finally, in Section 6, a transformation of hyperbolic rotation to a more stable orthogonal operation is derived. Conclusions are then given in Section 7.

## 2 Windowed RLS Estimation

Suppose at time $n$, the QRD of $[X(n) \vdots y(n)]$ is available, i.e.,

$$Q(n)[X(n)\vdots y(n)] = \begin{bmatrix} R(n) & \vdots & u(n) \\ 0 & \vdots & v(n) \end{bmatrix}, \tag{6}$$

where $Q(n) \in \Re^{\ell \times \ell}$ is orthogonal and $R(n) \in \Re^{p \times p}$ is upper triangular. Then the optimum $\hat{w}(n)$ is given [1] by

$$R(n)\hat{w}(n) = u(n). \tag{7}$$

$R(n)$ is called the Cholesky factor of $X^T(n)X(n)$ in that $R^T(n)R(n) = X^T(n)X(n)$. The Cholesky factor can be obtained by computing the $p \times p$ *sample covariance matrix* $X^T(n)X(n)$ first, followed by Cholesky decomposition. But, this method will have the condition number squared while forming the covariance matrix. A numerical stable approach is to perform QRD directly on the data matrix $X(n)$ and in this way the condition number of the LS problem can be maintained.

Now at time $n+1$, how do we obtain $R(n+1), u(n+1)$ and hence $w(n+1)$ with the minimum effort? If the window size is growing, then we can simply update $R(n)$ by $p$ Givens orthogonal transformations to zero out $\mathbf{x}_{n+1}^T$ and obtain $R(n+1)$ [3]. But with a fixed sliding window scheme, in addition to zeroing out the new data row $\mathbf{x}_{n+1}^T$ by orthogonalization, it is still necessary to *downdate* the obsolete data row, $\mathbf{x}_{n-\ell+1}^T$. We define *updating* as a series of operations (Givens rotations) such that an *additive* rank-one modification of the Cholesky factor is accomplished, and *downdating* as operations (hyperbolic rotations) such that a *subtractive* rank-one modification is made. It is noticed that at time $n+1$, the data

matrix

$$X(n+1) = \begin{bmatrix} \mathbf{x}_{n-\ell+2}^T \\ \vdots \\ \mathbf{x}_n^T \\ \mathbf{x}_{n+1}^T \end{bmatrix} \tag{8}$$

is obtained by adding a new row data $\mathbf{x}_{n+1}^T$ and removing an old row data $\mathbf{x}_{n-\ell+1}^T$ from $X(n)$. Since $R^T(n+1)R(n+1) = X^T(n+1)X(n+1)$, we have

$$R^T(n+1)R(n+1) = R^T(n)R(n) + \mathbf{x}_{n+1}\mathbf{x}_{n+1}^T - \mathbf{x}_{n-\ell+1}\mathbf{x}_{n-\ell+1}^T. \tag{9}$$

Rader and Steinhardt [5] proposed hyperbolic Householder transformation to update multiple new data rows and downdate multiple undesired ones simultaneously. Alexander *et al.* [4] suggested performing orthogonal rotations for updating followed by hyperbolic rotations for downdating. It can be shown that hyperbolic rotation is merely a degenerate case of hyperbolic Householder transformation, if we do not distinguish a rotation matrix from a reflection matrix [1]. This is just like that Givens rotation can be considered as a special case of Householder transformation. To facilitate systolic array processing, we will adopt the latter approach for windowed RLS filtering which involves only scalar computations.

## 2.1 Up/down-dating Cholesky factor

The basic up/downdating of the Cholesky factor is considered in this section. Given $[R(n)\vdots u(n)]$, we can obtain $[R(n+1)\vdots u(n+1)]$ by first updating

$$\begin{bmatrix} R(n) & \vdots & u(n) \\ \mathbf{x}_{n+1}^T & \vdots & y_{n+1} \\ \mathbf{x}_{n-\ell+1}^T & \vdots & y_{n-\ell+1} \end{bmatrix} \tag{10}$$

via $p$ Givens rotations, i.e.,

$$G_{p,p+1}\cdots G_{2,p+1}G_{1,p+1}\begin{bmatrix} R(n) & \vdots & u(n) \\ \mathbf{x}_{n+1}^T & \vdots & y_{n+1} \\ \mathbf{x}_{n-\ell+1}^T & \vdots & y_{n-\ell+1} \end{bmatrix}$$

$$= \begin{bmatrix} \tilde{R}(n+1) & \vdots & \tilde{u}(n+1) \\ 0 & \vdots & v_1(n+1) \\ \mathbf{x}_{n-\ell+1}^T & \vdots & y_{n-\ell+1} \end{bmatrix}, \tag{11}$$

then downdating the right-hand-side via $p$ hyperbolic rotations, i.e.,

$$H_{p,p+2}\cdots H_{2,p+2}H_{1,p+2}\begin{bmatrix} \tilde{R}(n+1) & \vdots & \tilde{u}(n+1) \\ 0 & \vdots & v_1(n+1) \\ \mathbf{x}_{n-\ell+1}^T & \vdots & y_{n-\ell+1} \end{bmatrix}$$

$$= \begin{bmatrix} R(n+1) & \vdots & u(n+1) \\ 0 & \vdots & v_1(n+1) \\ 0 & \vdots & v_2(n+1) \end{bmatrix}. \tag{12}$$

Here a $(p+2)\times(p+2)$ Givens rotation matrix $G_{i,p+1}$ is used to zero out the $(p+1,i)$ element of the matrix in (10), i.e.,

$$G_{i,p+1}\begin{bmatrix} \vdots \\ \alpha_i \\ \vdots \\ \alpha_{p+1} \\ \cdot \end{bmatrix} = \begin{bmatrix} I & & & \\ & c_i & s_i & \\ & & I & \\ & -s_i & c_i & \\ & & & 1 \end{bmatrix}\begin{bmatrix} \vdots \\ \alpha_i \\ \vdots \\ \alpha_{p+1} \\ \cdot \end{bmatrix} = \begin{bmatrix} \vdots \\ \sqrt{\alpha_i^2+\alpha_{p+1}^2} \\ \vdots \\ 0 \\ \cdot \end{bmatrix}, \tag{13}$$

where

$$c_i = \alpha_i/\sqrt{\alpha_i^2+\alpha_{p+1}^2} \quad \text{and} \quad s_i = \alpha_{p+1}/\sqrt{\alpha_i^2+\alpha_{p+1}^2}. \tag{14}$$

Similarly a $(p+2)\times(p+2)$ hyperbolic rotation matrix $H_{i,p+2}$ is used to zero out the $(p+2,i)$ element of the matrix in (11),

$$H_{i,p+2}\begin{bmatrix}\vdots\\ \alpha_i\\ \vdots\\ \alpha_{p+2}\end{bmatrix}=\begin{bmatrix} I & & & \\ & \tilde{c}_i & & -\tilde{s}_i \\ & & I & \\ & -\tilde{s}_i & & \tilde{c}_i\end{bmatrix}\begin{bmatrix}\vdots\\ \alpha_i\\ \vdots\\ \alpha_{p+2}\end{bmatrix}=\begin{bmatrix}\vdots\\ \sqrt{\alpha_i^2-\alpha_{p+2}^2}\\ \vdots\\ 0\end{bmatrix}, \tag{15}$$

where

$$\tilde{c}_i=\alpha_i/\sqrt{\alpha_i^2-\alpha_{p+2}^2}\quad\text{and}\quad \tilde{s}_i=\alpha_{p+2}/\sqrt{\alpha_i^2-\alpha_{p+2}^2}. \tag{16}$$

Since $G_{i,p+1}$ only affects the $i^{th}$ and $p+1^{th}$ rows of the matrix in (10), and $H_{i,p+2}$ affects the $i^{th}$ and $p+2^{th}$ rows, we can combine (11) and (12) in the following manner,

$$H_{p,p+2}\cdots H_{2,p+2}H_{1,p+2}G_{p,p+1}\cdots G_{1,p+1}\begin{bmatrix}R(n) & \vdots & u(n)\\ \mathbf{x}_{n+1}^T & \vdots & y_{n+1}\\ \mathbf{x}_{n-\ell+1}^T & \vdots & y_{n-\ell+1}\end{bmatrix}$$

$$=\begin{bmatrix}R(n+1) & \vdots & u(n+1)\\ 0 & \vdots & v_1(n+1)\\ 0 & \vdots & v_2(n+1)\end{bmatrix}. \tag{17}$$

# 3 Dual-State Systolic Triarray

Similar to the systolic QRD triarray proposed by Gentleman and Kung [2], which only performs updating, a *dual-state* systolic triarray performing both updating and downdating is given in Fig. 2. For every sensor (column of the data matrix) there is a delay buffer of window size $\ell$ to queue up data. Therefore each data will be first fetched and processed (updated) and then stays in the queue buffer for $\ell$ data clocks and finally will be *re*processed (downdated) by the triarray. Before the skewed data rows enter the arrays, there is an array of selection switches to alternatively take in new data and old data. The clock rate

for the processors should be two times higher than the data input rate so that both new and old data can be processed within one data clock. We use a black circle • to denote a processor working on a Givens rotation(updating) and a white circle o to denote a hyperbolic rotation(downdating). We also note that only one control bit is required in determining whether updating or downdating operation needs to be performed.

To this dual-state systolic triarray, data rows are skewed with updating and downdating data interleaved to form a sequence of up/down-dating wavefronts which will then hit upon this triarray sequentially. All of the wavefronts are consistent, i.e., the involved processors will all perform updating or downdating according to the underlying wavefront. As one updating wavefront finds its way along the triarray, one downdating wavefront follows next to it, which is then followed by another updating wavefront, and so forth.

Every processor, after experiencing one updating wavefront, will switch from updating to downdating operation as the next downdating wavefront will pass through it immediately following the previous updating wavefront. Therefore, all processors perform updating and downdating successively. This is just like they are doing flip-flops in *time*, which characterizes the temporal duality of this systolic triarray.

A spatial duality can be also observed as follows. While a processor is performing updating, all its adjacent processors, either vertical or horizontal , but not diagonal neighbors, are performing downdating. To see this, let us take the $(2,3)$ processor in Fig. 2 for an example. When this internal cell is performing updating, its right neighbor, the $(2,4)$ internal cell, and its lower neighbor, the $(3,3)$ boundary cell, are being hit by the downdating wavefront just *before* the updating wavefront that hit upon this very $(2,3)$ internal cell (recall that up/downdating wavefronts occur consecutively). Similarly, its left neighbor, the $(2,2)$ boundary cell, and its upper neighbor, the $(1,3)$ internal cell, must be performing downdating, too, as these two neighbor cells are confronting the downdating wavefront which follows right *after* the updating wavefront associated with this $(2,3)$ cell. We therefore say

that this triarray also functions like doing flip-flops in *space*.

In all, for every snapshot, we can see all processors are doing updating and downdating evenly distributed over the entire triarray, and for the very next snapshot, they change their roles. The phenomenon of flip-flops both in time and space characterizes the dual-state systolic triarrays. The wavefronts for the updating and downdating also propagate pairwise toward the lower-right direction in the triarray.

# 4 Cordic Processors

Cordic (*co*ordinate *r*otation *di*gital *c*omputation) processors [6, 7, 8] have been shown to be able to efficiently perform Givens and hyperbolic rotations with simple operations like `add`, `subtract` and `shift`, and one fixed-number multiplication.

## 4.1 Givens rotations

First consider the determination of the rotation angle $\theta$, such that a vector $[a, b]^T$ is rotated into $[\frac{a}{|a|}\sqrt{a^2+b^2}, 0]^T$, i.e.,

$$\begin{bmatrix} \frac{a}{|a|}\sqrt{a^2+b^2} \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}. \tag{18}$$

We can approximately split $\theta$ into $N$ predetermined minirotation angles with the proper choice of the directions of these angles, such that each minirotation only involves additions and shift registers. To see this, a recurrence of minirotations can be written as

$$\begin{aligned} \begin{bmatrix} a_{i+1} \\ b_{i+1} \end{bmatrix} &= \begin{bmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{bmatrix} \begin{bmatrix} a_i \\ b_i \end{bmatrix} \\ &= \cos\theta_i \begin{bmatrix} 1 & \tan\theta_i \\ -\tan\theta_i & 1 \end{bmatrix} \begin{bmatrix} a_i \\ b_i \end{bmatrix} \end{aligned}$$

$$= \cos\theta_i \begin{bmatrix} a_i + \rho_i 2^{-i} b_i \\ -\rho_i 2^{-i} a_i + b_i \end{bmatrix}, i = 0, 1, \cdots, N-1, \tag{19}$$

where

$$\begin{bmatrix} a_0 \\ b_0 \end{bmatrix} \equiv \begin{bmatrix} a \\ b \end{bmatrix} \tag{20}$$

and

$$\tan\theta_i = \rho_i 2^{-i}. \tag{21}$$

The planar sign bit $\rho_i$ is determined by

$$\rho_i = \begin{cases} 1\,, & \text{if } a_i b_i \geq 0, \\ -1\,, & \text{otherwise,} \end{cases} \tag{22}$$

and the intentional choice of the minirotaion angle $\theta_i$ in (21) renders the **shift-by-$i$ bits** (multiplied by $2^{-i}$) operations.

If the number of minirotation stages $N$ is large enough, it can be shown that

$$\begin{bmatrix} a_N \\ b_N \end{bmatrix} = \left( \prod_{i=0}^{N-1} \cos\theta_i \right) \left( \prod_{i=0}^{N-1} \begin{bmatrix} 1 & \tan\theta_i \\ -\tan\theta_i & 1 \end{bmatrix} \right) \begin{bmatrix} a \\ b \end{bmatrix} \tag{23}$$

$$\approx \begin{bmatrix} \frac{a}{|a|}\sqrt{a^2+b^2} \\ 0 \end{bmatrix}. \tag{24}$$

$\mathcal{K}_c = \prod_{i=0}^{N-1} \cos\theta_i \approx 0.60725$ is called a planar rotation correction factor and is usually independent of $N$ when $N$ is large enough. The rotation angle is thus uniquely determined by the planar sign bits $\rho's$,

$$\theta \approx \sum_{i=0}^{N-1} \rho_i \tan^{-1} 2^{-i}. \tag{25}$$

In our updating scheme, it is necessary to apply the same rotation to all the subsequent data on the two data rows involved (i.e, with one being in the triarray and the other the new data row being updated). In fact, it is *not* necessary to wait until all the minirotation planar sign bits $\rho's$ are generated from the boundary cell. In order to take advantages of

the fact that all the subsequent data on these two rows of data are to be rotated in the same manner as that in the boundary cell, we can pipeline these minirotaion angles to the internal cells as soon as they become available. Therefore every time a planar sign bit is generated by the boundary cell, it can propagate to the rest of its right-hand-side internal cells such that the others can start doing minirotations as soon as possible. Thus along the horizontal direction, the clock rate is the same as a mini-clock of the Cordic cells, and the vertical direction has the rate of $(N+1)$ times the mini-clock rate, which can be set equal to the external data rate. Because of the systolic miniclock along the horizontal data rows is much smaller then the data clock rate, we can consider that the Givens rotation is almost simultaneously applied to every data on these data rows, or, the rotation angles are broadcast along the rest of the internal cells in the same row.

### 4.2 Hyperbolic rotations

For the same reason, a sequence of mini-hyperbolic rotations can be found to accomplish a hyperbolic rotation as follows:

$$\begin{bmatrix} \frac{a}{|a|}\sqrt{a^2-b^2} \\ 0 \end{bmatrix} = \begin{bmatrix} \cosh\phi & -\sinh\phi \\ -\sinh\phi & \cosh\phi \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}. \tag{26}$$

Now, a recurrence of mini-hyperbolic rotations are given as

$$\begin{aligned} \begin{bmatrix} a_{i+1} \\ b_{i+1} \end{bmatrix} &= \begin{bmatrix} \cosh\phi_i & -\sinh\phi_i \\ -\sinh\phi_i & \cosh\phi_i \end{bmatrix} \begin{bmatrix} a_i \\ b_i \end{bmatrix} \\ &= \cosh\phi_i \begin{bmatrix} 1 & -\tanh\phi_i \\ -\tanh\phi_i & 1 \end{bmatrix} \begin{bmatrix} a_i \\ b_i \end{bmatrix} \\ &= \cosh\phi_i \begin{bmatrix} a_i - \sigma_i 2^{-i} b_i \\ -\sigma_i 2^{-i} a_i + b_i \end{bmatrix}, \quad i = 1, \cdots, N, \end{aligned} \tag{27}$$

where

$$\begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \equiv \begin{bmatrix} a \\ b \end{bmatrix} \tag{28}$$

and

$$\tanh \phi_i = \sigma_i 2^{-i}, \tag{29}$$

and the hyperbolic sign bit $\sigma_i$ is determined by

$$\sigma_i = \begin{cases} 1\,, & \text{if } a_i b_i \geq 0, \\ -1\,, & \text{otherwise.} \end{cases} \tag{30}$$

If $N$ is large enough, it can be shown that

$$\begin{bmatrix} a_{N+1} \\ b_{N+1} \end{bmatrix} = \left( \prod_{i=1}^{N} \cosh \phi_i \right) \left( \prod_{i=1}^{N} \begin{bmatrix} 1 & -\tanh \phi_i \\ -\tanh \phi_i & 1 \end{bmatrix} \right) \begin{bmatrix} a \\ b \end{bmatrix} \tag{31}$$

$$\approx \begin{bmatrix} \frac{a}{|a|}\sqrt{a^2 - b^2} \\ 0 \end{bmatrix}. \tag{32}$$

We call $\mathcal{K}_h = \prod_{i=1}^{N} \cosh \phi_i \approx 1.2051$ the hyperbolic rotation correction factor.

### 4.3 Cordic cells

The Cordic implementation of dual-state systolic array has the same structure as in Fig. 2. Since we have splitted a rotation into $N$ minirotations, a system block used in Fig. 2 will be divided into $N$ miniclocks, too. We notice that rotating two rows of data needs not take place sequentially from one column to another, which is the case used in Fig. 2. In fact we can broadcast the rotation data $((c, s)/(\tilde{c}, \tilde{s})$ from the boundary cell to all its right-hand-side internal cells such that their rotations can be completed simultaneously. Unfortunately, while implementing a VLSI circuit, the routing difficulty incurred due to non-local connections prohibits such broadcast. This is the reason why systolic array processing is much favored from the VLSI viewpoints in addition to its massive parallelism

and regularity. However, by using Cordic cells, we are able to minic the row broadcast of rotation data with only local connections.

Cordic rotations distinguish themselves by performing minirotations sequentially. The minirotations data are carried merely by a stream of one sign bit data. These minirotation bits can be sequentially passed along without waiting until the whole stream is available. Therefore, the right-hand-side internal cells can start doing minirotations as soon as every bit of minirotation data becomes available. The stream of sign bits are propagate horizontally along the right internal cells. By doing this, new data are skewed with only minicloak in between, instead of one system block. We also observe that the Cordic implementation reduces the wavefronts of skewed data from an tilting slope of 1 to $1/N$. If $N$ is big enough, we can say the data is almost not skewed and a rotation is taking place simultaneously on each row of the triarray.

Fig. 4 depicts the boundary and internal Cordic processing cells. To differentiate between updating and downdating operations, all the parameters in the parentheses represent a downdating computation. We use a solid arrow $\downarrow$ to represent a data movement in a system clock rate and a dashed arrow $--\!>$ in a miniclock rate. Along the horizontal direction, instead of passing the rotation parameters $c, s$ and $\tilde{c}, \tilde{s}$ in a system clock rate (which is the same as the clock rate along the vertical direction), the minirotation sign bits $\rho$ and $\sigma$ moves towards the right in a miniclock rate.

The boundary cell is responsible for determining the sign bits $\rho's$ and $\sigma's$. It has an internal memory to store the diagonal element $r$ in the upper triangular matrix. In the first miniclock, $i = 0$ ($i$ denoted the miniclock index of a complete rotation cycle), it fetches data $y$ from above. In the following miniclocks ($0 < i < N$), $r$ and $y$ are cyclically fedback to the minirotator to successively generate the minirotation sign bits $\rho(\sigma)$ and propagate them to the right-hand-side internal cells. In the last minirotation stage, the internal data $r$ is multiplied by a correction factor $\mathcal{K}_c(\mathcal{K}_h)$ and restored to its local memory. This completes

a rotation cycle of the boundary Cordic cell.

As to the internal Cordic cell, it also takes in external data from above in the first miniclock, then successively feedbacks data and rotates according to incoming sign bits. In the meantime, these sign bits are also propagated to the right. In the last miniclock, both data $r$ and $y$ on two feedback arms are multiplied by the correction factors, with one restored to its local memory and the other output downward for further processing.

Both boundary and internal Cordic cells share many architectural similarities. The differences between updating and downdating are: (1). the correction factors; (2). the downdating skips the first minirotation; (3). sign in the lower left adder input of the minirotators.

# 5 Recursive Estimation of Optimal Residuals

We have considered the recursive evaluation of coefficient vector $\hat{w}(n)$ in Section 2. In many applications such as beamformation, array processing and filtering, and communication, the optimal weight coefficient vector is not of direct interest. Instead, we are interested in the newest optimal residual $\hat{e}_n$ which is the last element of $\hat{e}(n)$ in (4). Information is then extracted from the optimal residual. In this section, we consider an efficient implementation to obtain the newest residuals under the up/downdating operations.

¿From (6), we can separate $Q(n)$ into two terms as

$$Q(n) = \begin{bmatrix} Q_1(n) \\ Q_2(n) \end{bmatrix}, \tag{33}$$

where $Q_1(n) \in \Re^{p \times l}$, $Q_2(n) \in \Re^{(l-p) \times l}$, such that

$$Q_1(n)X(n) = R(n),$$

$$Q_2(n)X(n) = 0.$$

Also from (4), (6), and (17), we can rewrite the optimal residual vector as

$$\hat{e}(n+1) = -Q_2^T(n+1)\begin{bmatrix} v_1(n+1) \\ v_2(n+1) \end{bmatrix}. \tag{34}$$

Now, the question is how to obtain $Q_2(n+1)$ recursively and efficiently? Define

$$\bar{Q}(n+1) = \begin{bmatrix} Q_1(n) & & \\ & 1 & \\ & & 1 \end{bmatrix}, \tag{35}$$

then

$$\bar{Q}(n+1) = [X(n+1) \vdots y(n+1)] = \begin{bmatrix} R(n) & \vdots & u(n) \\ \mathbf{x}_{n+1}^T & \vdots & y_{n+1} \\ \mathbf{x}_{n-l+1}^T & \vdots & y_{n-l+1} \end{bmatrix}. \tag{36}$$

¿From (17) and discussions in Section 2, denote

$$H(n+1) = H_{p,p+2} \cdots H_{2,p+2} H_{1,p+2},$$

$$G(n+1) = G_{p,p+2} \cdots G_{2,p+2} G_{1,p+2},$$

we have

$$Q(n+1) = H(n+1)G(n+1)\bar{Q}(n+1) \tag{37}$$

if updating is performed first and

$$Q(n+1) = G(n+1)H(n+1)\bar{Q}(n+1) \tag{38}$$

if downdating is performed first. Suppose updating is performed first, we have

$$Q(n+1) = H(n+1)Q_u(n+1), \tag{39}$$

where $Q_u(n+1) = G(n+1)\bar{Q}(n+1)$ is defined as the $Q$ matrix associated with updating only. It can be shown that $G$ is of the form

$$G(n+1) = \begin{bmatrix} Z(n+1) & h(n+1) & 0 \\ k^T(n+1) & \prod_{i=1}^{p} c_i & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{40}$$

where $Z(n+1)$ is a $p \times p$ matrix, and therefore $Q_u$ is of the form

$$Q_u(n+1) = \begin{bmatrix} Z(n+1)Q_1(n) & h(n+1) & 1 \\ k^T(n+1)Q_1(n) & \prod_{i=1}^{p} c_i & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{41}$$

It can also be shown that $H$ is of the form

$$H(n+1) = \begin{bmatrix} \tilde{Z}(n+1) & 0 & \tilde{h}(n+1) \\ 0 & 1 & 0 \\ \tilde{k}^T(n+1) & 0 & \prod_{i=1}^{p} \tilde{c}_i \end{bmatrix}, \tag{42}$$

and therefore $Q(n+1)$ is of the form

$$Q(n+1) = \begin{bmatrix} \tilde{Z}(n+1)Z(n+1)Q_1(n) & \tilde{Z}(n+1)h(n+1) & \tilde{h}(n+1) \\ k^T(n+1)Q_1(n) & \prod_{i=1}^{p} c_i & 0 \\ \tilde{k}^T(n+1)Z(n+1)Q_1(n) & \tilde{k}^T(n+1)h(n+1) & \prod_{i=1}^{p} \tilde{c}_i \end{bmatrix}. \tag{43}$$

¿From (34) and (41), we can obtain the residual vector when the updating wavefront passes through the array and which is

$$\hat{e}_u(n+1) = \begin{bmatrix} \bar{e}_u(n+1) \\ e_{u_1}(n+1) \\ e_{u_2}(n+1) \end{bmatrix} = \begin{bmatrix} -Q_1^T(n)k(n+1)v_1(n+1) \\ -\prod_{i=1}^{p} c_i \cdot v_1(n+1) \\ -v_2(n+1) \end{bmatrix}, \tag{44}$$

where $e_{u_1}$ and $e_{u_2}$ are the newest residuals associated with the updating and downdating respectively at time $n+1$. Since at this point the downdating has not yet been performed, $e_{u_2}(n+1)$ is not considered as residual.

¿From (34) and (43), we can obtain the residual vector when the downdating wavefront passes through the triarray. Again, we are only interested in the newest residuals (the last two elements) and which are

$$\begin{bmatrix} e_1(n+1) \\ e_2(n+1) \end{bmatrix} = \begin{bmatrix} -\prod_{i=1}^{p} c_i \cdot v_1(n+1) - h^T(n+1)\tilde{k}(n+1)v_2(n+1) \\ -\prod_{i=1}^{p} \tilde{c}_i \cdot v_2(n+1) \end{bmatrix}, \tag{45}$$

where $e_1$ and $e_2$ are the residuals associated with updating and downdating respectively. From (17), it can be seen that $v_1(n+1)$ and $v_2(n+1)$ can be obtained naturally from the up/downdating operations in the triarray. If the updating parameters $c_i$'s are propagated down to the diagonal boundary cells and are cumulatively multiplied as in [3], when the updating wavefront passes through the triarray, the term $\prod c_i$ in (44) has been obtained. A multiplier call is then used to obtained $e_{u_1}(n+1) = -\prod_{i+1}^{p} c_i \cdot v_1(n+1)$ as in [3]. In fact, although the window size is $l$ as described in (2), the residual $e_{u_1}(n+1)$ is estimated from an $l+1$ window $[\mathbf{x}_{n-l+1}, \cdots, \mathbf{x}_n, \mathbf{x}_{n+1}]^T$ and $[y_{n-l+1}, \cdots, y_n, y_{n+1}]^T$ since downdating of $\mathbf{x}_{n-l+1}$ has not yet been done. That is

$$e_{u_1}(n+1) = \mathbf{x}_{n+1}^T \hat{w}_{[n-l+1,n+1]} - y_{n+1}, \tag{46}$$

where $\hat{w}_{[n-l+1,n+1]}$ denotes the optimal coefficient vector estimated from data $[\mathbf{x}_{n-l+1}, \cdots, \mathbf{x}_n, \mathbf{x}_{n+1}]^T$ and $[y_{n-l+1}, \cdots, y_n, y_{n+1}]^T$.

Also, when the downdating wavefront passes through the triarray, if $\tilde{c}_i$'s are propagated down to the diagonal boundary cells and are cumulatively multiplied, from (45), the downdating residual can be obtained easily. It is estimated from a $l$ window $[\mathbf{x}_{n-l+2}, \cdots, \mathbf{x}_n, \mathbf{x}_{n+1}]^T$. That is,

$$e_2(n+1) = \mathbf{x}_{n-l+1}^T \hat{w}_{[n-l+2,n+1]} - y_{n-l+1}. \tag{47}$$

Obviously, the residual of time $n-l+1$ is *post estimated* by data from $n-l+2$ to $n+1$ and appears at time $n+1$. This kind of property may or may not be of practical interest in real-life applications. As to the updating residual $e_1(n+1)$, due to the term $h^T(n+1)\tilde{k}(n+1)v_2(n+1)$ which is not available from the systolic implementation, we are unable to extract $e_1(n+1)$ from the triarray. However, (45) provides a simple relation for this updating residual before and after the downdating. That is,

$$e_1(n+1) = e_{u_1}(n+1) - h^T(n+1)\tilde{k}(n+1)v_2(n+1). \tag{48}$$

If downdating is performed first, then by the same analysis it shows that we can obtained

$$e_{d_2}(n+1) = -\prod_{i=1}^{p} \tilde{c}_i \cdot v_2(n+1) = \mathbf{x}_{n-l+1}^T \hat{w}_{[n-l+2,n]} - y_{n-l+1}, \tag{49}$$

and

$$e_1(n+1) = -\prod_{i=1}^{p} c_i \cdot v_1(n+1) = \mathbf{x}_{n+1}^T \hat{w}_{[n-l+2,n+1]} - y_{n+1}. \tag{50}$$

¿From (4) and (8), it is obvious that this $e_1(n+1)$ is the exact residual we are looking for. However, a drawback for this scheme is that downdating first before the updating may incur numerical stability problem [4]. A dual-state up/downdating systolic array for the recursive residual estimation is shown in Fig. 2.

# 6 Operations Transformations

¿From numerical stability consideration, the hyperbolic rotation for the downdating is not of practical interest, even though it has been proven to be forward (weakly) stable in [11]. One of the reasons is that, from Fig. 3, $\tilde{c}$ and $\tilde{s}$ generalized by the boundary cell could be very larger. Once these $\tilde{c}$ and $\tilde{s}$ are sent to the internal cells for further processing, the computations involve two more parameters $r$ and $z$ which are not scaled (*i.e.* they could be very large, too.). Therefore, $z'$ and the updated $r$ may suffer large amount of roundoff error, or even overflow. To stabilize this problem, we are looking for an algebraically equivalent orthogonal parameters to replace $\tilde{c}$ and $\tilde{s}$. To do so, let us first consider the relation between updating and downdating.

Suppose we have known an $p \times p$ upper triangular matrix $R$ and want to downdate a vector $\mathbf{x}$ to obtain an upper triangular matrix $\tilde{R}$. That is,

$$\tilde{R}^T \tilde{R} = R^T R - \mathbf{x}\mathbf{x}^T. \tag{51}$$

If we know $\tilde{R}$ instead of $R$, then

$$\tilde{R}^T \tilde{R} + \mathbf{x}\mathbf{x}^T = R^T R \tag{52}$$

becomes an updating problem.

To downdate $R$, we use a sequence of hyperbolic rotations to zero out $\mathbf{x}$ as

$$\begin{bmatrix} \tilde{R} \\ 0 \end{bmatrix} = H_{p,p+1} \cdots H_{2,p+1} H_{1,p+1} \begin{bmatrix} R \\ \mathbf{x}^T \end{bmatrix}. \tag{53}$$

On the other hand, to update $\tilde{R}$, we use a sequence of Givens rotations to zero out $\mathbf{x}$ as

$$G_{p,p+1} \cdots G_{2,p+1} G_{1,p+1} \begin{bmatrix} \tilde{R} \\ \mathbf{x}^T \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix}. \tag{54}$$

Suppose now, for this updating problem, we know $R$ instead of $\tilde{R}$, and $k-1$ updating has been done, *i.e.*

$$G_{k-1,p+1} \cdots G_{1,p+1} \begin{bmatrix} \tilde{R} \\ \mathbf{x}^T \end{bmatrix} = \begin{bmatrix} R^{k-1} \\ \tilde{R}_{p-k+1} \\ 0, \cdots, 0, x_k^{(k-1)}, \cdots, x_p^{(k-1)} \end{bmatrix}, \tag{55}$$

where $R^{k-1}$ denotes the first $k-1$ rows of $R$, $\tilde{R}_{p-k+1}$ denotes the last $p-k+1$ rows of $\tilde{R}$, and $x^{(k)}$ denotes an element of the $k^{th}$ updated vector $\mathbf{x}$. At the $k^{th}$ rotation, let us focus only on the $k^{th}$ and the last rows, we have

$$\begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix} \begin{bmatrix} 0, & \cdots, & 0, & \tilde{r}_{k,k}, & \cdots, & \tilde{r}_{k,p} \\ 0, & \cdots, & 0, & x_k^{(k-1)}, & \cdots & x_p^{(k-1)} \end{bmatrix} = \begin{bmatrix} 0, & \cdots, & 0, & r_{k,k} & \cdots & \cdots, & r_{k,p} \\ 0, & \cdots, & 0, & 0 & x_{k+1}^{(k)}, & \cdots & x_p^{(k)} \end{bmatrix}, \tag{56}$$

where $\tilde{r}_{i,j}$ and $r_{i,j}$ are the $(i,j)$ elements of $\tilde{R}$ and $R$ respectively,

$$r_{k,k} = \sqrt{\tilde{r}_{k,k}^2 + x_k^{(k-1)^2}}, \tag{57}$$

$$c_k = \frac{\tilde{r}_{k,k}}{r_{k,k}}, s_k = \frac{x_k^{(k-1)}}{r_{k,k}},$$

and for $j = k+1, \cdots, p$,

$$\begin{cases} r_{k,j} = c_k \tilde{r}_{k,j} + s_k x_j^{(k-1)}, \\ x_j^{(k)} = -s_k \tilde{r}_{k,j} + c_k x_j^{(k-1)}. \end{cases} \tag{58}$$

¿From (57) and (58), we can solve $\tilde{r}$ and $x^{(k)}$ easily and the downdating can now be achieved by using Givens rotation parameters as

$$\tilde{r}_{k,k} = \sqrt{r_{k,k}^2 - x_k^{(k-1)^2}}, \tag{59}$$

$$c_k = \frac{\tilde{r}_{k,k}}{r_{k,k}}, s_k = \frac{x_k^{(k-1)}}{r_{k,k}},$$

for for $j = k+1, \cdots, p$,

$$\begin{cases} \tilde{r}_{k,j} = (r_{k,j} - s_k x_j^{(k-1)})/c_k, \\ x_j^{(k)} = -s_k \tilde{r}_{k,j} + c_k x_j^{(k-1)}. \end{cases} \tag{60}$$

As we can see that $c_k$ and $s_k$ are bounded parameters ($\leq 1$). Even $\tilde{r}_{k,j}$ could be very large, when computing $x_j^{(k)}$, it is only multiplied with $s_k$ which is bounded. That is to say, $\tilde{r}_{k,j}$ will never be magnified during the computing. Thus, this scheme is more stable than hyperbolic rotation. The operations of the processing cells for the systolic implementation are shown in Fig. 5.

With this transformation, the operations of the downdating part are different from that of the updating part. However, both provide stable numerical property especially under finite precision computation. If we want to make both operations the same for some implementation consideration, a transformation on $c$ and $s$ in Fig. 3 can be performed such that a new $\hat{c}$ and $\hat{s}$ are obtained as

$$\hat{c} = \frac{1}{c}, \qquad \hat{s} = \frac{s}{c}, \tag{61}$$

and therefore,

$$\begin{aligned} r &= (r + \hat{s}y)/\hat{c}, \\ y' &= -\hat{s}r + \hat{c}y, \end{aligned} \tag{62}$$

which share the same forms as that of the downdating part. However, it loses the numerical property of an orthogonal transformation. The operations of this transformed updating are shown in Fig. 5.

# 7 Conclusions

A dual-state systolic triarray performing up/down-dating operations to facilitate fixed-window RLS filtering has been proposed. While previous researches have been centered on QRD-based systolic triarray with exponentially forgetting factors to perform updating, no VLSI-suitable structure, like systolic array, is yet known to perform fixed-window RLS filtering. We also provide a computer simulation to substantiate the necessity of fixed-window scheme under some nonstationary conditions despite its two-times higher computational load than that of exponentially forgetting methods.

Due to the inherent similarity between updating and downdating, they can use the same hardware and alternatively pipelined in parallelism in this dual-state systolic triarray. A flip-flop systolic behavior of this array is observed both in temporal and spatial domain. We also propose Cordic cells to mimic a broadcast along the rows in the triarray and also reduce the propagation of rotation data (word-level) alongs rows down to minirotation sign bit stream. The hardware complexity using Cordic cells is simpler and the involved computations simply comprise of simple arithmetics, and square root and division operations are not required. As to the optimal residuals, we have also shown what can and cannot be obtained by using the systolic implementation.

To remedy potential round-off errors associated with downdating, transformed operations are also considered. By providing these stabilized operations, the issues of numerical stability and pipelined computation are addressed.

We have investigated efficient and symmetric algorithms and architecture to perform fixed-window RLS filtering problems. We also extend previous results from updating to up/down-dating operations. The proposed new dual-state systolic triarray is very promising for VLSI implementation.

## References


[1] G. H. Golub and C. F. Van Loan, *"Matrix computations"*. Johns Hopkins University Press, Baltimore, MD, 2nd edition, 1989.

[2] W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic array". *Proc. SPIE, vol. 298: Real-time signal processing IV*, pages 19–26, Bellingham, Washington, 1981.

[3] J. G. McWhirter, "Recursive least-squares minimisation using a systolic array". *Proc. SPIE 431, Real time signal processing VI*, pages 105–112, 1983.

[4] S. T. Alexander, C. T. Pan, and R. J. Plemmons, "Numerical properties of a hyperbolic rotation method for windowed RLS filtering". Proc. *IEEE ICASSP*, pages 423–426, 1987.

[5] C. M. Rader and A. O. Steinhardt, "Hyperbolic Householder transformations". *IEEE Trans. Acoust., Speech, Signal Processing*, 34(6):1589–1602, Dec. 1986.

[6] J. S. Walther, "A unified algorithm for elementary functions". Proc. *AFIPS Conf. Proc., Vol. 38*, pages 379–385, 1971.

[7] H. M. Ahmed, J.-M. Delosme, and M. Morf, "Highly concurrent computing structures for matrix arithmetic and signal processing". *IEEE Computer*, 15(1):65–82, Jan. 1982.

[8] C. M. Rader, "Wafer-scale systolic array for adaptive antenna processing filtering". Proc. *IEEE ICASSP*, pages 2069–2071, 1988.

[9] S. Haykin. *"Adaptive filter theory"*. Prentice-Hall, Englewood Cliffs, NJ, 1986.

[10] D. Manolakis, F. Ling, and J. G. Proakis. "Efficient time-recursive least-squares algorithms for finite-memory adaptive filtering". *IEEE Trans. on Circuits and Systems*, CAS-34(4):400–407, Apr. 1987.

[11] S. T. Alexander, C. T. Pan, and R. J. Plemmons, "Analysis of a recursive least squares hyperbolic rotation algorithm for signal processing", *Linear Algebra and Its Applications*, 98:3-40, 1988.

[12] S.I. Chou and C.M. Rader, "Algorithm-based error detection of a Cholesky factor updating systolic array using CORDIC processors", Proc. SPIE, Real-time Signal Processing XI, pp.104-111, Aug. 1988.

[13] H.T. Kung, "Why systolic architectures?", IEEE Computer, Vol 15, pp.37, Jan. 1982.

[14] S.Y. Kung, *VLSI Array Processors*, Prentice Hall, 1988.

[15] K.J.R. Liu, S.F. Hsieh, and K. Yao, "Two-level pipelined implementation of systolic block Householder transformations with application to RLS algorithm" Int'l Conf. on Application-Specific Array Processors, Princeton, Sep. 1990.

[16] J.G. McWhitter and T.J. Shepherd, "Systolic array processor for MVDR beamforming", IEE Proc. Vol 135, Pt. F, pp.75-80, 1989.

[17] J.G. Nash, K.W. Przytula, and S. Hansen, "Systolic/Cellular processor for linear algebraic operations", VLSI Signal Processing II, pp.306-315, 1986.

[18] R. Schreiber, "Implementation of adaptive array algorithms", IEEE Trans. ASSP Vol ASSP-34, pp.1038-1045, Oct. 1986.

[19] C.R. Ward, P.J. Hargrave and J.G. McWhirter, "A novel algorithm and architecture for adaptive digital beamforming", IEEE Trans. Antennas Propagat., Vol AP-34, pp.338-346, March, 1986.

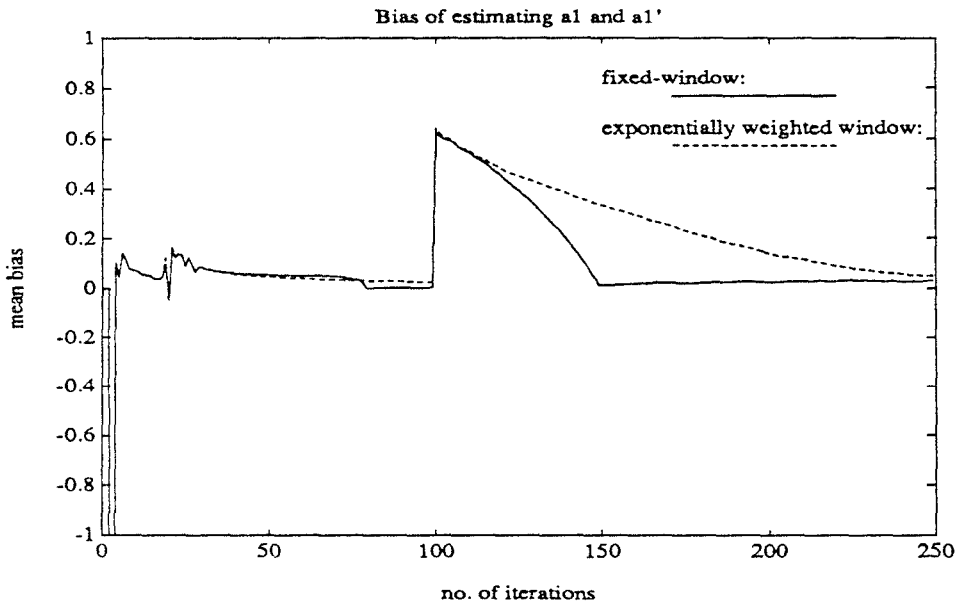



Figure 1: Mean bias of estimating nonstationary AR parameter for fixed and exponentially weighted windows.

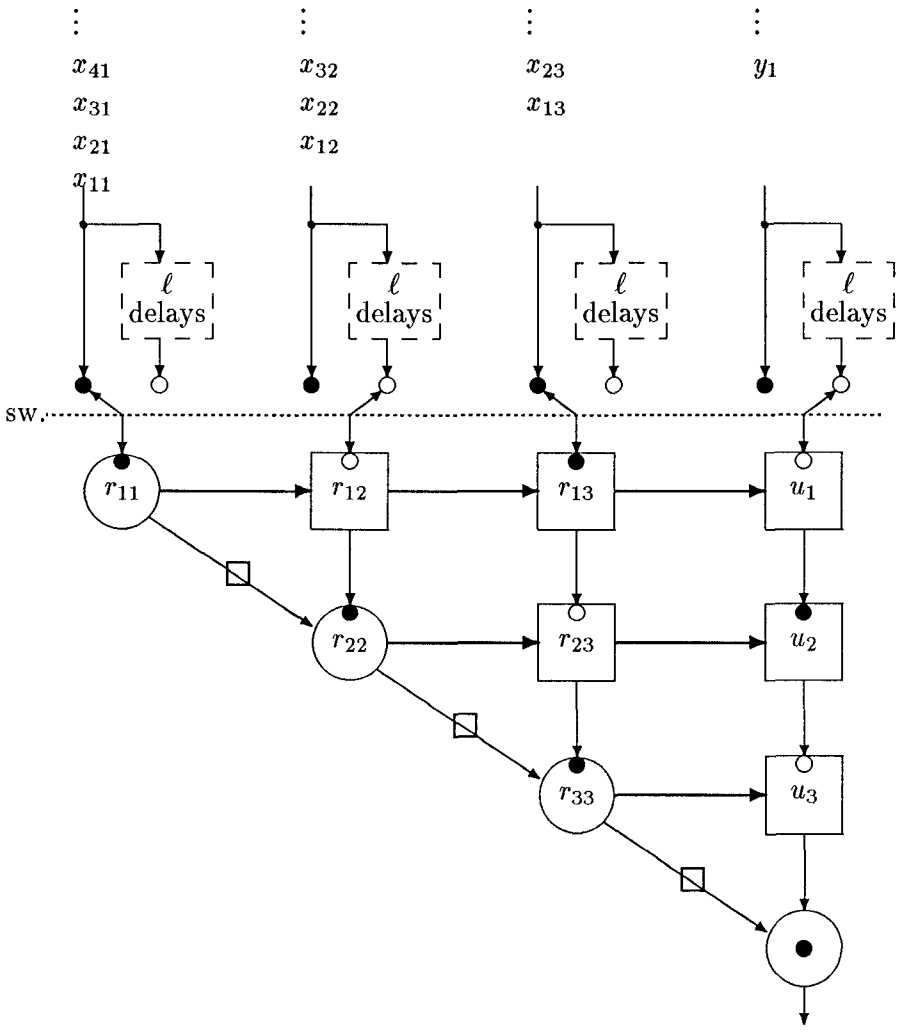



Figure 2: Dual-State Systolic Array for Windowed-RLS Problems

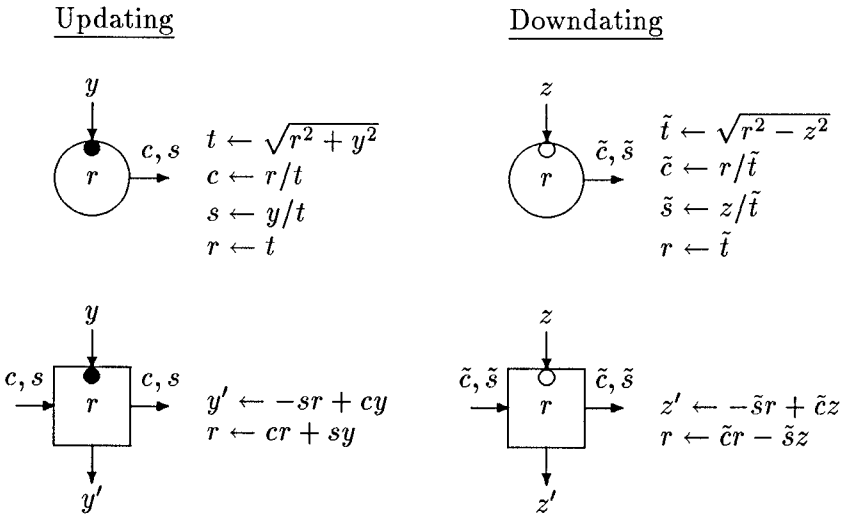
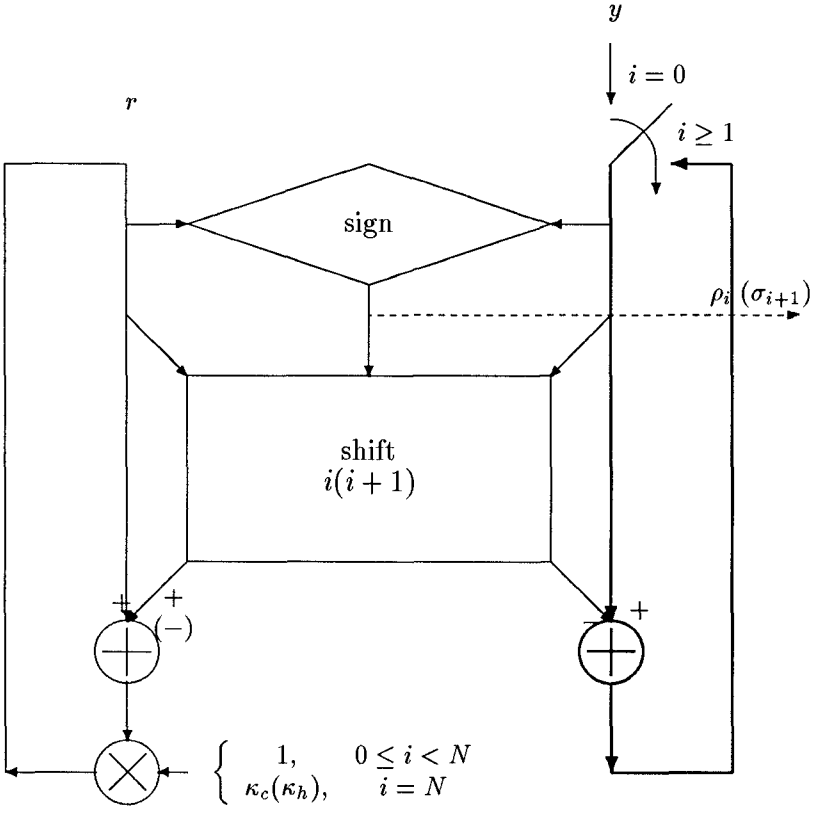
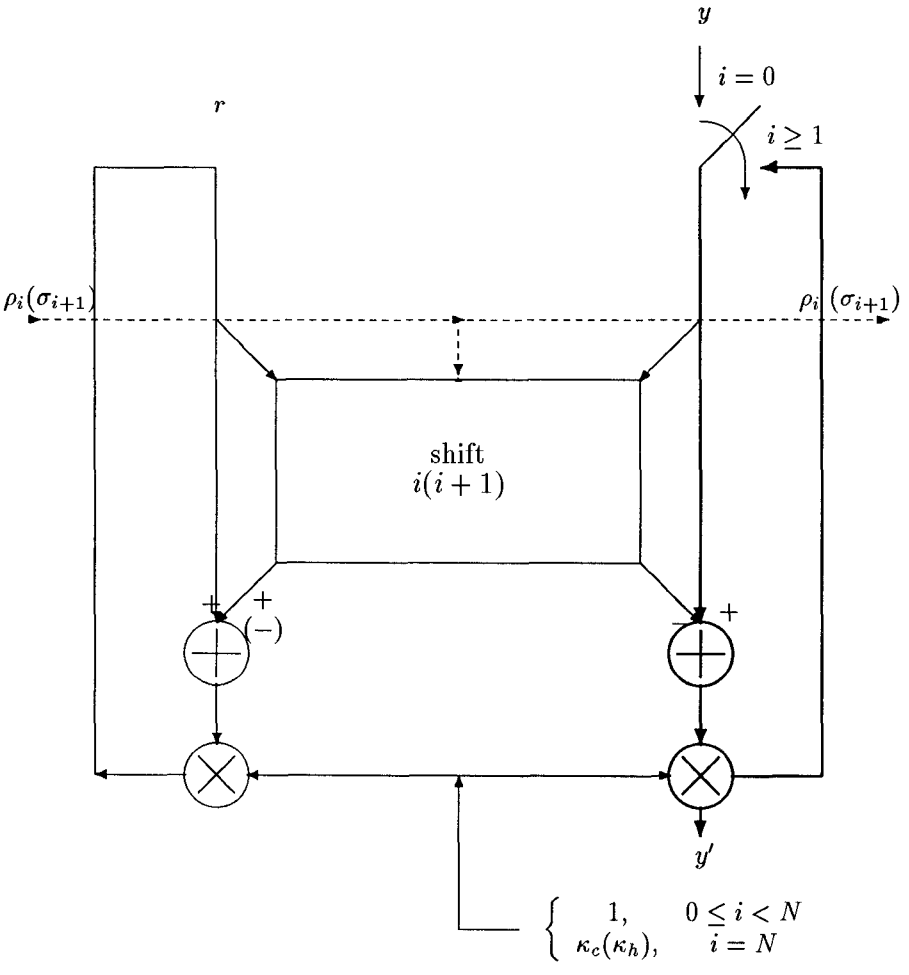


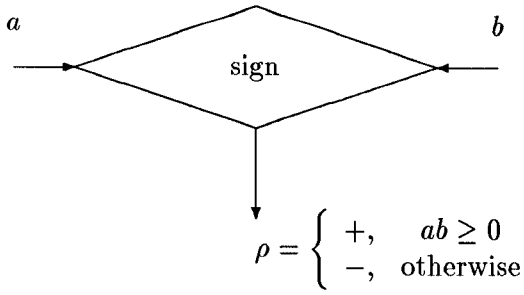

Figure 3: Boundary and internal cells



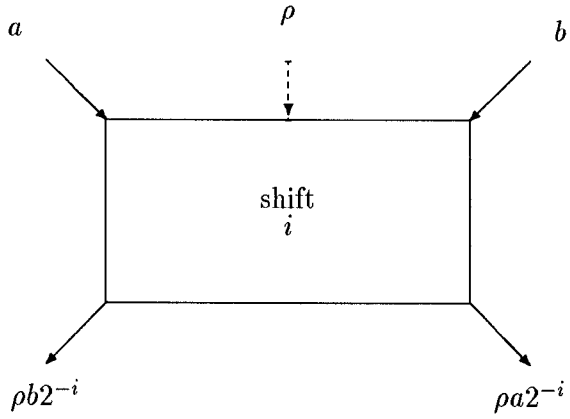

(a). Boundary cordic processor




**(b).** Internal cordic processor




**(c).** Sign bit




**(d).** Shift-$i$ operation

Figure 4: Cordic cells

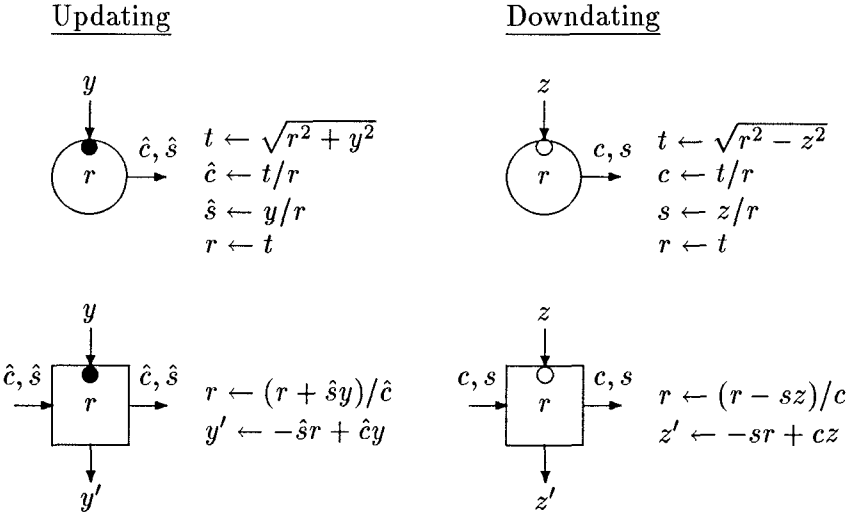



Figure 5: Transformed boundary and internal cells