SYSTOLIC IMPLEMENTATION OF WINDOWED RECURSIVE LS ESTIMATION*

S.F. Hsieh and **K. Yao**

Electrical Engineering Department
University of California, Los Angeles
Los Angeles, CA 90024 - 1594

ABSTRACT

We propose a *dual-state systolic* structure to perform joint up/downdating operations encountered in windowed recursive least squares (RLS) estimation problems. It is derived by successively performing Givens rotation for updating and hyperbolic rotation for downdating. Due to the data independancy, 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. By using Cordic cells, the system performance can be improved both in hardware complexity and throughput rate.

I 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 ℓ as the window size, p as the order of the system (i.e., no. of sensors in a multichannel filtering case) and n is the time index($n \geq \ell$ is assumed). Our 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.

*This work is partially supported by National Science Foundation Grant NCR-8814407 and the MICRO program.

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. 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(suitable for VLSI design) to perform fixed-windowed RLS filtering. 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.

II 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 $R(n+1)$. 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

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

CH2868-8/90/0000-1931$1.00 © 1990 IEEE

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 $X(n)$. It can also be shown that

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

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'll adopt the latter approach for windowed RLS filtering which involves only scalar computations.

II-A Up/down-dating Cholesky factor

Given $[R(n)\vdots u(n)]$, we can obtain $[R(n+1)\vdots u(n+1)]$ by first updating

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

via p Givens rotations, i.e.,

$$G_{p,p+1}\cdots G_{2,p+1}G_{1,p+1}\begin{bmatrix} R(n) & \vdots & u(n) \\ \mathbf{x}_{n+1}^T & \vdots & y_{n+1} \\ \mathbf{x}_{n-\ell+1}^T & \vdots & y_{n-\ell+1} \end{bmatrix}$$
$$= \begin{bmatrix} \tilde{R}(n+1) & \vdots & \tilde{u}(n+1) \\ 0 & \vdots & . \\ \mathbf{x}_{n-\ell+1}^T & \vdots & y_{n-\ell+1} \end{bmatrix}, \quad (11)$$

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

$$H_{p,p+2}\cdots H_{2,p+2}H_{1,p+2}\begin{bmatrix} \tilde{R}(n+1) & \vdots & \tilde{u}(n+1) \\ 0 & \vdots & . \\ \mathbf{x}_{n-\ell+1}^T & \vdots & y_{n-\ell+1} \end{bmatrix}$$
$$= \begin{bmatrix} R(n+1) & \vdots & u(n+1) \\ 0 & \vdots & . \\ 0 & \vdots & . \end{bmatrix}. \quad (12)$$

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.,

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

where

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

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),

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

where

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

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 eqn. (11) and (12) and *interleave* the Givens matrices G's with the hyperbolic matrices H's in the following manner,

$$H_{p,p+2}G_{p,p+1}\cdots H_{2,p+2}G_{2,p+1}H_{1,p+2}G_{1,p+1}\begin{bmatrix} R(n) & \vdots & u(n) \\ \mathbf{x}_{n+1}^T & \vdots & y_{n+1} \\ \mathbf{x}_{n-\ell+1}^T & \vdots & y_{n-\ell+1} \end{bmatrix}$$
$$= \begin{bmatrix} R(n+1) & \vdots & u(n+1) \\ 0 & \vdots & . \\ 0 & \vdots & . \end{bmatrix}. \quad (17)$$

III 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. (1). For every sensor (column of the data matrix) there is a delay buffer of window size ℓ to queue up data. Therefore each data will be first fetched and processed (updated) and then stays in the queue buffer for ℓ 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 ○ to denote a hyperbolic rotation(downdating).

All processors perform updating and downdating successively, doing flip-flops in *time*. While a processor is performing updating, all its adjacent processors are performing downdating, doing flip-flops in *space*. The phenomenon of flip-flops both in time and space characterizes the dual-state systolic arrays. The wavefronts for the updating and downdating also propagate pairwise toward the lower-right direction in the triarray.

IV Cordic Processors

Cordic (*co*ordinate *r*otation *di*gital *c*omputation) processors [6, 7] 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.

IV-A Givens rotations

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

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

We can approximately split θ 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

$$\begin{bmatrix} a_{i+1} \\ b_{i+1} \end{bmatrix} = \begin{bmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{bmatrix}\begin{bmatrix} a_i \\ b_i \end{bmatrix}$$
$$= \cos\theta_i\begin{bmatrix} 1 & \tan\theta_i \\ -\tan\theta_i & 1 \end{bmatrix}\begin{bmatrix} a_i \\ b_i \end{bmatrix}$$
$$= \cos\theta_i\begin{bmatrix} a_i+\rho_i 2^{-i}b_i \\ -\rho_i 2^{-i}a_i+b_i \end{bmatrix}, i=0,1,\cdots,N-1, \quad (19)$$

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 ρ_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 θ_i in eqn. (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 regular 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 regular 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 regular cells in the same row.

IV-B 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 σ_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.

IV-C Cordic cells

The Cordic implementation of dual-state systolic array has the same structure as in Fig. (1). But 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 ρ and σ moves towards the right in a miniclock rate. We use a solid arrow $\downarrow$ to represent a data movement in a system clock rate and a dashed arrow $--\!>$ in a miniclock rate.

Fig. (3) depicts the boundary and regular Cordic processing cells. To differentiate between updating and downdating operations, all the parameters in the parentheses represent a downdating computation. 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 ($i = 0$) 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 $\rho(\sigma)$. 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. 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.

The regular Cordic cell takes in external data from above in the first miniclock, then successively feedbacks data and rotates according to the sign bits. These sign bits are also propagated to the right. In the last miniclock, both data are multiplied by the correction factors, with one restored to its local memory and the other output downward for further processing. Both boundary and regular Cordic cells share many architectural similarities.

V Conclusions

A dual-state systolic triarray performing up/down-dating operations to facilitate fixed-window RLS filtering has be proposed. Because of the inherent similarity between updating and downdating, they can use the same hardware and alternatively pipelined in parallelism. We also propose Cordic cells to mimic a broadcast along the rows in the triarray to further increase the throughput rate.

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". In *Proc. SPIE, vol. 298: Real-time signal processing IV*, pages 19–26, Bellingham, Washington, 1981. Society of Photo-optical Instrumentaion Engineers.

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

[4] S. T. Alexander, C. T. Pan, and R. J. Plemmons. "Numerical properties of a hyperbolic rotation method for windowed RLS filtering". In *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". In *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.

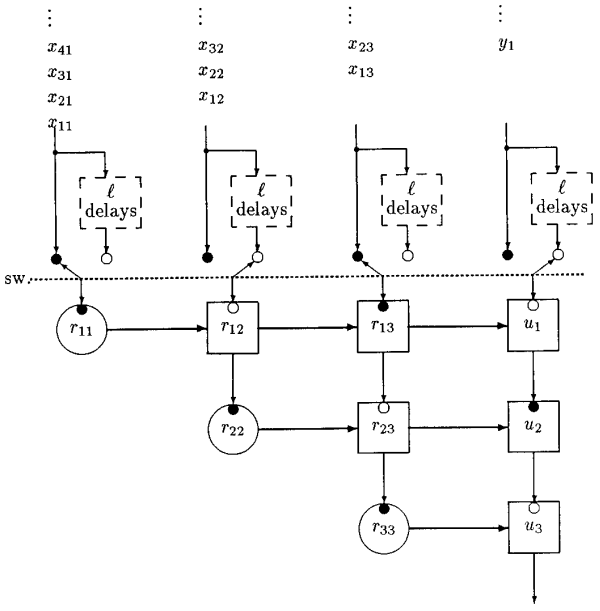


Figure 1: Dual-State Systolic Array for Windowed-RLS Problems

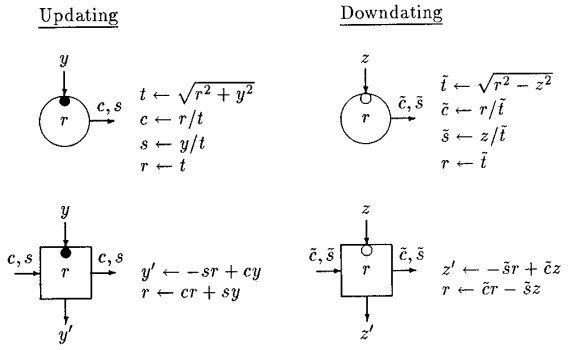
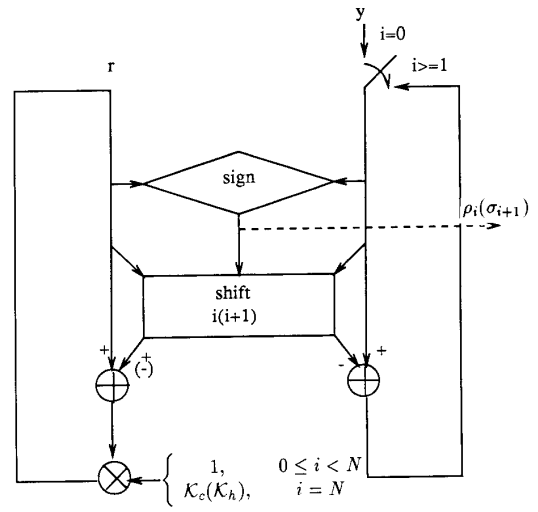
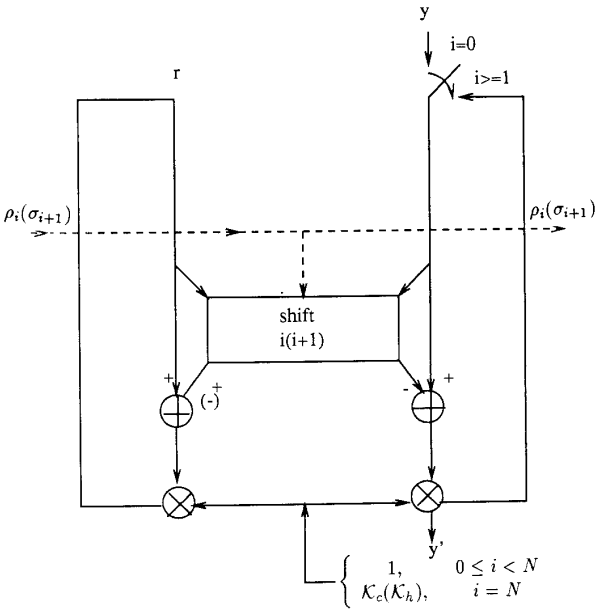


Figure 2: Boundary and regular cells

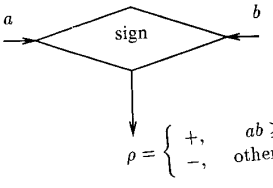
Boundary Cordic cell



Regular Cordic cell



Sign bit



Shift-i operation

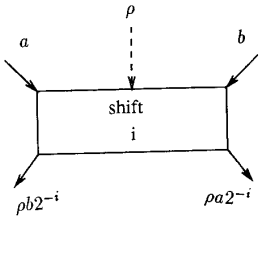


Figure 3: Cordic Cells