AD-A142 747

(2)

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS:
2a. SECURITY CLASSIFICATION AUTHORITY:	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE:	
4. PERFORMING ORGANIZATION REPORT NUMBER(S):	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 84-0492
6a. NAME OF PERFORMING ORGANIZATION: Stanford University (6b. OFFICE SYMBOL (If applicable):)	7a. NAME OF MONITORING ORGANIZATION: Air Force Office of Scientific Research
6c. ADDRESS (City, State and ZIP Code): Department of Electrical Engineering, Stanford CA 94305	7b. ADDRESS (City, State and ZIP Code): Directorate of Mathematical & Information Sciences, Bolling AFB DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR (8b. OFFICE SYMBOL (If applicable): NM)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: AFOSR-83-0228
8c. ADDRESS (City, State and ZIP Code): Bolling AFB DC 20332	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	2304	A6	

11. TITLE (Include Security Classification): ESTIMATION AND CONTROL IN THE VLSI ERA

12. PERSONAL AUTHOR(S): T. Kailath

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	DEC 1983	12

16. SUPPLEMENTARY NOTATION: 22nd IEEE Conference on Decision and Control, Dec 1983, San Antonio TX.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The theme of this lecture is that the availability of very high density integrated circuits will be changing our approach and emphasis to several problems in estimation and control.

For example, the minimality of realizations will be less significant than their modularity, local interconnectedness, area time complexity measures, etc. Similarly, good algorithms for serial processing may be poor candidates for parallel implementation. While it is hard for me in mid-August to predict exactly what I shall say in the lecture in mid-December, I think it might be useful to provide in written form some of the background material on which a good part of my talk will be based. Thus, at this meeting at least, I plan to illustrate the above points by several examples, including: (1) description of a parallel architecture for the measurement update step (in triangular array form) of the Kalman filter; (2) development of the Schur algorithm as a better candidate than the Levinson algorithm (CONTINUED)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
CPT Brian W. Woodruff	(202) 767- 5027	NM

ITEM #19, ABSTRACT, CONTINUED: for VLSI implementation of Toeplitz equation solvers; (3) comparison of the Berlekamp-Massey-Rissanen and Lanczos algorithms in the problems of partial realization and of the decoding of BCH codes; and (4) development of minimal, but pipelined and orthogonally-cascaded, implementations of time-invariant, finite-dimensional (ARMA) systems.

On the other hand, while much of the development and demonstration of new parallel computing structures, such as systolic arrays and dataflow machines, has been carried out by computer scientists, I hope to show (by examples) that system theorists can contribute to the understanding and analysis of such structures and help develop more efficient ones.

ESTIMATION AND CONTROL IN THE VLSI ERA[1]

by

Thomas Kailath
Department of Electrical Engineering
Stanford University
Stanford, CA 94035

INTRODUCTION

The theme of this lecture is that the availability of very high density integrated circuits will be changing our approach and emphasis to several problems in estimation and control.

For example, the minimality of realizations will be less significant than their modularity, local interconnectedness, area time complexity measures, etc. Similarly, good algorithms for serial processing may be poor candidates for parallel implementation. While it is hard for me in mid-August to predict exactly what I shall say in the lecture in mid-December, I think it might be useful to provide in written form some of the background material on which a good part of my talk will be based. Thus, at this meeting at least, I plan to illustrate the above points by several examples, including:

1) description of a parallel architecture for the measurement update step (in triangular array form) of the Kalman filter,
2) development of the Schur algorithm as a better candidate than the Levinson algorithm for VLSI implementation of Toeplitz equation solvers,
3) comparison of the Berlekamp-Massey-Rissanen and Lanczos algorithms in the problems of partial realization and of the decoding of BCH codes,
4) development of minimal, but pipelined and orthogonally-cascaded, implementations of time-invariant, finite-dimensional (ARMA) systems.

On the other hand, while much of the development and demonstration of new parallel computing structures, such as systolic arrays and dataflow machines, has been carried out by computer scientists, I hope to show (by examples) that system theori ts can contribute to the understanding and analysis of such structures and help develop more efficient ones.

1. A PARALLEL ARCHITECTURE FOR MEASUREMENT UPDATES

This part of the talk will be based upon a paper "A Parallel Architecture for Kalman Filter Measurement Update," by J. Jover and T. Kailath, June 1983.

The main theme, in this example and some of the later ones, is to show how reformulation of an algorithm can lead to more efficient implementation.

In this paper we present a parallel computing structure of the systolic array type for implementing a new algorithm for the measurement update step of the Kalman filter for state-space estimation. With a single serial processor, the update of a scalar measurement would take time $O(n^2)$, where n is the state dimension; we present an array with $2n+4$ elementary processors and a bank of delay units, that will carry out the measurement update in time $O(n)$. More savings can be realized by an extended architecture that will update a p-dimensional measurement with $O(np)$ processors in time $O(max\{n, p\})$, instead of time $O(max\{n^2p, np^2, p^3\})$ for the single-processor implementation.

The only earlier work that we are aware of in this direction is that of Andrews (1981) who developed a parallel structure with $O(n^2)$ elementary processors for implementing the so-called $U-D$ algorithm for the measurement update by rearranging the order of computation of certain equations given by Bierman (1975), (1977). Our structure allows us to update not only the covariance factors but also the state estimates themselves, and also has other advantages over Andrews scheme.

The new structure was in fact suggested by a different way (Kailath (1982)) of carrying out the $U-D$ measurement update--using triangularization of an $(n+p)\times(n+p)$ matrix via Modified Givens rotations (as given by Gentleman (1973)). It can be shown that for a scalar measurement and a serial processor, Bierman's equations are equivalent to our triangularization method. However, the fact that there are no explicit equations in our scheme seems to make it easier to conceive of a parallel computing structure. The architecture we propose is of the wavefront or systolic-type (see, e.g., Mead and Conway (1980), M. T. Kung (1982), S. Y. Kung (1982)).

Moreover, it is significant that our algorithm is essentially the same whether we have scalar or vector measurements, while to our knowledge it is difficult to extend Bierman's equations to the vector case. The main reason is that Bierman's derivation is based on a formula of Agee and Turner (1972) for updating the LDU factors of a rank-one perturbation of a given matrix; with a p-dimensional measurement, we have a rank p update and there is no simple extension of the Agee-Turner formula to this case. The usual way around this is to process the measurements sequentially, after some preliminary data transformation, but a direct parallel implementation will then take time $O(np)$ as opposed to the $O(max\{n, p\})$ time for our structure.

We might mention that the ideas and schemes of this paper can also be used for other forms of the

[1] This talk will be based in part on work supported in recent years at Stanford by the Air Force Office of Scientific Research, the Army Research Office, the Joint Services Electronics Program, the Defense Advanced Research Budgets Agency, and the National Science Foundation.

Grant-
AFOSR-83-0228

measurement update, e.g. using strictly triangular factors (with scalar arithmetic square roots) rather than the LDU forms. Parallel architectures for time-update calculations can also be obtained, though apparently with $O(n^2)$ processors rather than $O(n)$; since this question is still under study, we restrict ourselves in this paper to the measurement update problem.

1.1 MEASUREMENT UPDATE AND BIERMAN'S U-D EQUATIONS

We use the by now almost standard notation introduced by Kalman (1960), in terms of which we can state the

MEASUREMENT UPDATE EQUATIONS:

$$\hat{x}_{i|i} = \hat{x}_i + K_{f,i}(y_i - H_i\hat{x}_i) \ , \qquad i \geq 0 \tag{1a}$$

$$\hat{x}_0 := 0 \tag{1b}$$

$$K_{f,i} = P_i H_i^T R_{e,i}^{-1} \tag{2}$$

$$R_{e,i} = H_i P_i H_i^T + R_i \tag{3}$$

$$P_{i|i} = P_i - P_i H_i^T R_{e,i}^{-1} H_i P_i \tag{4a}$$

$$P_0 := \Pi_0 \tag{4b}$$

TIME UPDATE EQUATIONS:

$$\hat{x}_{i+1} = F_i \hat{x}_{i|i} \ , \qquad i \geq 0 \tag{5}$$

$$P_{i+1} = F_i P_{i|i} F_i^T + G_i Q_i G_i^T \tag{6}$$

Several alternatives to Eqns. (1)-(6) have been suggested, based on the idea (first used by Potter (1963)) of propagating square-root factors of P_i and $P_{i|i}$. There are several forms of such algorithms (see Kaminski, Bryson, and Schmidt (1971); Morf and Kailath (1975); Bierman (1977)). In particular Bierman modified a measurement update algorithm derived by Carlson (1973) so that it did not use any arithmetic square roots--a feature that may be important in certain implementations. This algorithm is based on working with LDU (lower-triangular, diagonal, upper-triangular) factorizations of P_i and $P_{i|i}$. Since P_i and $P_{i|i}$ are symmetric, $U = L^T$. Bierman uses the UDU^T form and calls his version of the algorithm for computing the $\{U, D\}$ factors of $P_{i|i}$ from the $\{U, D\}$ factors of P_i the $U-D$ algorithm.

Bierman's Equations

Let

$$P_i = LDL^T \ , \qquad P_{i|i} = L_+ D_+ L_+^T$$

Substituting these expressions into the measurement update Eqn. (4) gives

$$L_+ D_+ L_+^T = L \left(D - DL^T H_i^T R_{e,i}^{-1} H_i L D \right) L^T = L\bar{D}L^T \ , \quad \text{say}$$

Now if we have only a single measurement, i.e. $p = 1$, then H_i will be a row matrix, say

$$H_i = h_i^T \ , \qquad R_{e,i} = r_{e,i} \ , \qquad R_i = r_i$$

and if we now factor $\bar{D}$ as

$$\bar{D} = \bar{L}\bar{D}\bar{L}^T = D - DL^T h_i r_{e,i}^{-1} h_i^T L D$$

then clearly

$$L_+ = L\bar{L} \ , \qquad D_+ = \bar{D}$$

Note that $\bar{D}$ is a rank-one modification of D; in 1972 Agee and Turner showed how to compute the $\{L, D\}$ factors of such a matrix. Incorporating a numerical improvement suggested by Carlson (1973) (in particular replacing a subtraction by a division) Bierman (1975) obtained the following set of equations for the scalar measurement update (in a different notation).

In the following, brackets will refer to a particular element in a matrix or vector, e.g., $L_+[k,j]$ is the (k,j)-th element of the matrix L_+; the diagonal elements will be denoted by a single index, e.g. $D[j]$:

Define: $b^T = h^T L$

Initialize: $a_n = 0 \quad (n \times 1 \text{ vector})$

$$\Delta_n = r \qquad (\text{scalar})$$

Iterate for $j = n, n-1, \ldots, 1$

$$\Delta_{j-1} = \Delta_j + b^2[j] \cdot D[j]$$

$$D_+[j] = \Delta_j \cdot D[j] / \Delta_{j-1}$$

$$\beta = b[j] D[j] / \Delta_{j-1}$$

$$L_+[\cdot, j] = L[\cdot, j] - b[j] \cdot a_j$$

$$a_{j-1} = a_j + \beta \ L_+[\cdot, j]$$

Examination of this algorithm shows that it takes $O(n^2)$ additions and multiplications and $O(n)$ divisions to go from $\{L, D\}$ to $\{L_+, D_+\}$ (see Bierman (1977) p.107).

Andrews' Parallel Implementation

With the advent of VLSI technology, it was natural to ask if the processing time could be reduced by using parallel processing arrays, e.g. of the systolic type. A systolic-type network (see H.T. Kung (1982)) is characterized by the smooth flow of data through a network of simple Processing Elements (PEs) with only local communication among the PEs and with limited access to external data. Such an architecture is convenient for VLSI implementation because of the regularity and simplicity of the PEs and the short path of communication links. However, the algorithms that can be mapped into such an architecture must be such that maximum advantage is taken of the data available at a given time in a PE. [We may remark here that by the term systolic-type networks we are not making any distinction as to whether the final implementation will use the synchronous systolic arrays of H.T. Kung (1982) or the asynchronous Wavefront Array Processors of S.Y. Kung (1982)--the distinction really depends upon the size of the matrix.]

A first attempt at a parallel implementation was made by Andrews (1981), who modified the order of execution of some of the equations in the Bierman algorithm and gave a data flow diagram. However, his scheme cannot be implemented using simple and synchronized Processing Elements (PEs) as in a systolic-type network. As indicated by Andrews' attempt, Bierman's version (1975) of the $U-D$ measurement update does not appear to be the best form for suggesting a parallel implementation. We shall describe in the next section how another look at the problem yielded a new algorithm, with no explicit equations, that suggests some satisfactory parallel implementations.

1.2 A NEW ALGORITHM FOR LDU MEASUREMENT UPDATE

It will be useful to illustrate the difficulty that led to the Bierman equations for the measurement update by first showing how to obtain a square-root algorithm for the time-update equation (6),

$$P_{i+1} = F_i P_{i|i} F_i^T + G_i Q_i G_i^T$$

For any nonnegative-definite matrix A, we shall define a lower triangular matrix $A^{1/2}$ as its unique lower triangular square root factor if

$$A = A^{1/2} A^{T/2} \ ; \qquad A^{T/2} = (A^{1/2})^T$$

and (for uniqueness) $A^{1/2}$ has nonnegative diagonal entries.

Then we claim the following: the time update is

solved by finding *any orthogonal* matrix Θ to triangularize the array $[F_i P_{i|i}^{1/2} \quad G_i Q_i^{1/2}]$, i.e. such that

$$[F_i P_{i|i}^{1/2} \quad G_i Q_i^{1/2}]\Theta = [P_{i+1}^{1/2} \quad 0] \tag{7}$$

This claim can be immediately verified by "squaring" both sides of Eqn. (7).

Computing the triangular factors $P_{i|i}^{1/2}$ and $P_{i+1}^{1/2}$ may imply taking arithmetic square roots, which are often somewhat more expensive to compute than multiplications or divisions, and therefore are sometimes avoided. This can be done by using *LDU* factorizations.

Kailath (1982) noted that the difficulty with the minus sign in the update measurement (Eqn. (4a))

$$P_{i|i} = P_i - P_i H_i^T R_{e,i}^{-1} H_i P_i$$

could be avoided by first noting that the right hand side above is the so-called *Schur complement* of $R_{e,i}$ in the (larger) matrix

$$\mathbf{M} = \begin{bmatrix} R_{e,i} & H_i P_i \\ P_i H_i^T & P_i \end{bmatrix}$$

This Schur complement arises in the block LDU factorization of this matrix; it is easy to check the decomposition

$$\mathbf{M} = \begin{bmatrix} I & 0 \\ K_{f,i} & I \end{bmatrix} \begin{bmatrix} R_{e,i} & 0 \\ 0 & P_{i|i} \end{bmatrix} \begin{bmatrix} I & K_{f,i}^T \\ H_i^T & I \end{bmatrix},$$

$$K_{f,i} = P_i H_i^T R_{e,i}^{-1}$$

On the other hand, we can also consider the Schur complement of P_i in M, which will be (see Eqn. (3))

$$R_{e,i} - H_i P_i P_i^{-1} P_i H_i^T = R_i$$

corresponding to the block LDU factorization

$$\mathbf{M} = \begin{bmatrix} I & H_i \\ 0 & I \end{bmatrix} \begin{bmatrix} R_i & 0 \\ 0 & P_i \end{bmatrix} \begin{bmatrix} I & 0 \\ H_i^T & I \end{bmatrix}$$

From these two block factorizations of M, we can conclude that there must be an orthogonal transformation matrix Θ such that

$$\begin{bmatrix} R_i^{1/2} & H_i P_i^{1/2} \\ 0 & P_i^{1/2} \end{bmatrix} \Theta = \begin{bmatrix} R_{e,i}^{1/2} & 0 \\ P_i H_i^T R_{e,i}^{-T/2} & P_{i|i}^{1/2} \end{bmatrix} \tag{8}$$

This is in fact the form of a well-known square root algorithm due to Dyer and McReynolds (1969).

As mentioned before, the transformation Θ in such algorithms may involve the use of scalar square roots, and in some situations it is useful to be able to avoid them. This can be attempted by using the LDU decompositions

$$R = L_R D_R L_R^T, \quad P_i = LDL^T, \quad P_{i|i} = L_+ D_+ L_+^T$$

and rewriting the above expression as (dropping all time-index subscripts for convenience)

$$\begin{bmatrix} L_R & HL \\ 0 & L \end{bmatrix} \begin{bmatrix} D_R & 0 \\ 0 & D \end{bmatrix}^{1/2} \Theta = \begin{bmatrix} L_{Re} & 0 \\ K_f L_{Re} & L_+ \end{bmatrix} \begin{bmatrix} D_{Re} & 0 \\ 0 & D_+ \end{bmatrix}^{1/2} \tag{9a}$$

or

$$\begin{bmatrix} L_R & H_L \\ 0 & L \end{bmatrix} \Theta_D = \begin{bmatrix} L_{Re} & 0 \\ K_f L_{Re} & L_+ \end{bmatrix} \tag{9b}$$

Kailath (1982) showed how to use an algorithm due to Gentleman (1973) to find Θ_D as a product of elementary matrices none of which contains any scalar square roots. In the case of scalar measurements ($p = 1$), writing out the steps explicitly (which is not necessary in the array methods) will lead exactly to the U–D equations of Bierman's (1975), which were obtained in the quite different way described before.

The Givens and Modified Givens Methods

The Givens method of matrix triangularization is to note that we can readily find an elementary orthogonal transformation to rotate any given 1×2 vector $[p_1 \; p_2]$ to make it lie along the first coordinate axis. In fact, note that

$$[p_1 \; p_2] \begin{bmatrix} \dfrac{p_1}{\sqrt{p_1^2+p_2^2}} & \dfrac{-p_2}{\sqrt{p_1^2+p_2^2}} \\ \dfrac{-p_2}{\sqrt{p_1^2+p_2^2}} & \dfrac{p_1}{\sqrt{p_1^2+p_2^2}} \end{bmatrix} = [\sqrt{p_1^2+p_2^2} \quad 0]$$

By systematically applying a sequence of such elementary transformations we can triangularize any given matrix, as any uncertain reader can check on a simple example.

The issue is how to avoid the (scalar) square roots in the transformation. Gentleman (1973) showed that this could be done by introducing weighted norms: rotate $[p_1 \; p_2]$ to lie along $[1 \; 0]$, keeping equality of the weighted norms

$$[p_1 \; p_2] \begin{bmatrix} d_{p1} & 0 \\ 0 & d_{p2} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = [1 \; 0] \begin{bmatrix} d_{q1} & 0 \\ 0 & d_{q2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{10}$$

In most problems $\{d_{p1}, d_{p2}\}$ will be given (as for example in (9)) and $\{d_{q1}, d_{q2}\}$ have to be determined. It is not hard to see what to do. We need to find an orthogonal matrix Θ such that

$$[p_1 \; p_2] \begin{bmatrix} d_{p1}^{1/2} & 0 \\ 0 & d_{p2}^{1/2} \end{bmatrix} \Theta = [1 \; 0] \begin{bmatrix} d_{q1}^{-1/2} & 0 \\ 0 & d_{q2}^{-1/2} \end{bmatrix}$$

or equivalently to find Θ_D such that

$$[p_1 \; p_2]\,\Theta_D = [1 \; 0] \tag{11a}$$

where

$$\Theta_D = \begin{bmatrix} d_{p1}^{1/2} & \\ & d_{p2}^{1/2} \end{bmatrix} \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} d_{q1}^{-1/2} & 0 \\ 0 & d_{q2}^{-1/2} \end{bmatrix} \tag{11b}$$

$$c^2 + s^2 = 1 \tag{11c}$$

Now note that from (10) d_{q1} is determined as

$$d_{q1} = p_1^2 d_{p1} + p_2^2 d_{p2} \tag{12}$$

Next it is easy to check that

$$[p_1 \; p_2] \begin{bmatrix} p_1 d_{p1}/d_{q1} & -p_2 \\ p_2 d_{p2}/d_{q1} & p_1 \end{bmatrix} = [1 \; 0] \tag{13}$$

So we have a candidate for Θ_D and the only issue is if (cf. (11b))

$$\Theta = \begin{bmatrix} d_{p1}^{1/2} & \\ & d_{p2}^{1/2} \end{bmatrix} \begin{bmatrix} p_1 d_{p1}/d_{q1} & -p_2 \\ p_2 d_{p2}/d_{q1} & p_1 \end{bmatrix} \begin{bmatrix} d_{q1}^{1/2} & 0 \\ 0 & d_{q2}^{1/2} \end{bmatrix} = \begin{bmatrix} p_1\sqrt{d_{p1}/d_{q1}} & -p_2\sqrt{d_{q2}/d_{p1}} \\ p_2\sqrt{d_{p2}/d_{q1}} & p_1\sqrt{d_{q2}/d_{p2}} \end{bmatrix}$$

is of the form

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix}, \qquad c^2 + s^2 = 1 \ .$$

This constraint can be met by choosing

$$d_{q2} = \frac{d_{p1} d_{p2}}{d_{q1}} \tag{14}$$

To apply the transformation in Eqn. (13) to an arbitrary vector $[p_1' \ p_2']$,

$$[p_1' \ p_2'] \begin{bmatrix} p_1 d_{p1}/d_{q1} & -p_2 \\ p_2 d_{p2}/d_{q1} & p_1 \end{bmatrix} = [q_1' \ q_2'] \tag{15}$$

will require four multiplications and two additions. In many applications, including ours, it happens that $p_1 = 1$ and for this case Golub (see Gentleman (1973)) noted that one could manage with two multiplies and two adds. To see this note that we can write

$$q_2' = -p_2 p_1' + p_2' \tag{16a}$$

$$\begin{aligned} q_1' &= \frac{d_{p1}}{d_{q1}} p_1' + p_2 \frac{d_{p2}}{d_{q1}} p_2' \\ &= \frac{d_{p1}}{d_{q1}} p_1' + p_2 \frac{d_{p2}}{d_{q1}} (q_2' + p_2 p_1') \\ &= \frac{d_{p1} + p_2^2 d_{p2}}{d_{q1}} p_1' + p_2 \frac{d_{p2}}{d_{q1}} q_2' \\ &= p_1' + (p_2 \frac{d_{p2}}{d_{q1}}) q_2' \end{aligned} \tag{16b}$$

Since $p_2 d_{p2}/d_{q1}$ is given as an element of the matrix (15), we see that (16a) and (16b) each requires only one multiply and one add.

This is the Modified Givens transformation introduced by Gentleman (1973); because scalar square roots are avoided, the use of the modified Givens transformation generally yields a speedup in computation, leading to the name fast Givens transformation.

An Example

The following numerical example shows how modified Givens transformations can be used to zero out one row of elements in a matrix with the same structure as in the measurement update algorithm presented in this paper (Eqn. (9)), with H a row vector (scalar measurement update) and L_R, D_R, L_{Re}, and D_{Re} scalars. To triangularize the matrix we will zero out all but the first element of the first row, and to avoid fill-in we will start zeroing out the element (1,4) [using the first and forth columns]. Then we will zero out the element (1,3) [using the first and third columns]; and so on. Let

$$\Theta_D = \Theta_D(1, n+1) \cdot \Theta_D(1, n) \cdots \Theta_D(1,2)$$

where the elementary rotation $\Theta_D(i, j)$ places a zero in the (i, j) element of the matrix, and n is the dimension of the state vector (in this numerical example $n = 3$). We write in the diagonal matrices for convenience of reference:

$$\left\{ \begin{bmatrix} 1 & 65 & 103 & 107 \\ 0 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & -2 & 3 & 1 \end{bmatrix} \cdot \begin{bmatrix} 571 & & & \\ & 1 & & \\ & & 3 & \\ & & & 2 \end{bmatrix} \right\} \quad \Theta_D(1,4)$$

$$\left\{ \begin{bmatrix} 1 & 65 & 103 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 9116e-2 & -2 & 3 & 1 \end{bmatrix} \cdot \begin{bmatrix} 23469 & & & \\ & 1 & & \\ & & 3 & \\ & & & .0487 \end{bmatrix} \right\} \quad \Theta_D(1,3)$$

$$\left\{ \begin{bmatrix} 1 & 65 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 5588e-2 & 3 & 1 & 0 \\ 2063e-1 & -2 & 2.0608 & 1 \end{bmatrix} \cdot \begin{bmatrix} 55296 & & & \\ & 1 & & \\ & & 1.2733 & \\ & & & .0487 \end{bmatrix} \right\} \quad \Theta_D(1,2)$$

$$\left\{ \begin{bmatrix} 1 & 0 & 0 & 0 \\ .1092e-2 & 1 & 0 & 0 \\ .8468e-2 & 2.6368 & 1 & 0 \\ .1699e-1 & -3.3412 & 2.0608 & 1 \end{bmatrix} \cdot \begin{bmatrix} 59521 & & & \\ & .9290 & & \\ & & 1.2733 & \\ & & & .0487 \end{bmatrix} \right\}$$

In the next section we describe a parallel architecture for the Kalman filter measurement update with the same structure as this numerical example.

1.3 ARCHITECTURE FOR PARALLEL MEASUREMENT UPDATE

It might seem that our problem could be solved by applying existing architectures for triangularizing matrices. Gentleman and Kung (1981) presented a triangular systolic array for this purpose; so did Ahmed, Delosme, and Morf (1982), who used a structure based on CORDICS as the basic Processing Element. However, our problem has special structure and requires to do additional computations (like determining the product $h^T \cdot L$). We shall combine all these special requirements in an architecture that (i) solves the problem with a linear number of processors (as opposed to $O(n^2)$ in general), (ii) computes the error covariance *and* state updates at the same time, and (iii) avoids a significant bottleneck in many systolic networks--namely, that the Processing Elements (usually on the boundary) that have to do more complex calculations slow down the throughput rate for the whole system.

For simplicity, we start with the scalar measurement update (so $H = h^T$, $L_R = 1$, $D_R = r$, $L_{Re} = 1$, $D_{Re} = r_e$). In the next section we extend this architecture to the vector case. The architecture we propose is depicted in Figure 1, where for simplicity of illustration, and without loss of generality, we have taken the number of states, n, to be 4.

From Eqn. (9) we see that we have to compute the product $h^T \cdot L = b^T$ and then zero each of the components of the vector b. Each time we input a set of data to our architecture, we shall zero out one element of the vector b, using the transformation $\Theta_D(1, j+1)$, as in the example of the previous section.

Eqn. (9) suggests that the architecture should have four parts. First, a column of elementary processors at the left whose function is to compute the inner product $h^T \cdot L = b^T$. Second, a processor at the top of Figure 1 whose objective is to compute the parameters for the elementary transformation, $\Theta_D(1,j+1)$. Third, a column of processors (at the right of the figure) to apply the transformation. Finally, a delay network that will delay the values of the matrix L until the appropriate moment to apply the transformation.

A more detailed description of the different Processing Elements (PEs) involved in our architecture follows. The column of processors that perform the product $h^T \cdot L$ is composed of n PEs. One of these PEs is depicted in Figure 2; with $b[\ ]$, $h[\ ]$, and $L[\ ,\ ]$ representing any element of the vectors b, h and of the matrix L. Every PE in this column of processors has a memory cell to hold one floating point number (one element of the vector h) which can be changed as explained at the end of Section V. The function of each of these PEs is to compute the product of the real number at one of the inputs with the value stored in memory and add the result to the other input.

We define *Processing Element Time* (PET) as the total time it takes a given PE to compute the output--including the time to transfer the data to a neighboring processor (with a protocol if the system is asynchronous). Note that the PET is independent of whether the individual PE is internally pipelinable (which will affect the throughput rate for the PE). In the literature on systolic-type arrays, the PET is usually called the clock (which can be misleading), and it is usually assumed to be 1. In systolic-type arrays the PEs with the largest PET will determine when data will be available to neighboring processors; therefore these processors will lower the throughput rate of the whole array. In our architecture--see Eqns. (17)-(19)--the largest PET corresponds to the processor that computes the parameters for the elementary transformation $\Theta_D(1, j+1)$, $j = n, n-1, \cdots 1$; by dividing this processor into four stages (other numbers can be chosen if required by hardware considerations) and by adding some appropriate delay banks, we make the whole architecture work with the PET corresponding to the simplest PE (viz. one of the processors in the left column of Figure 1). From now on we will refer to this PET as the *architecture PET*, namely, the time it takes the whole architecture to accept another set of data; this is the time required to perform a multiplication of two real numbers, plus one addition, plus the time involved in transferring data to a neighboring processor (see Figure 2). Note that, in general, the PET is greater than the clock period of the hardware.

The processor that computes the parameters of the transformation is composed of of four PEs (see Figure 3); its structure allows us to pipeline the computation of parameters for different transformations. We can divide the computation of $\Theta_D(1, j+1)$ into more than four stages to make sure that every single stage will take only the architecture PET to compute. We assume four stages; then it takes 4 PETs to compute the transformation parameters. Note that only one of the four PEs requires a memory cell.

The transformation $\Theta_D(1, j+1)$ is applied, in the form shown in Eqn. (16), by a column of n processors. A PE of this group is depicted in Figure 4; each of these PEs can be thought as being composed of three of the PEs of Figure 2. Finally, the delay is achieved by $n \times (n + \alpha - 1)$ registers, where usually $\alpha = 5$--as discussed at the end of this section.

Description of the Input

The inputs to our architecture are as follows (see Figure 1): n inputs to the column of processors at the left, plus one input to initialize the h processors, plus one input to the processor at the top of the architecture. We use the following notation: we represent the signals at a given point of the architecture by a row vector whose elements are the numbers appearing at that point of the architecture at different instants of time (usually every PET); the most recent values are in the leftmost column. Therefore we can talk about *input matrices*, comprised of several rows containing the input to several processors at consecutive PETs. In this matrix, each column will show all the inputs to a defined set of processors at a given instant of time; the next column shows the input after one PET. Similarly for *output matrices*. [Note that these systems can be asynchronous; in a row of the matrix, data in successive columns merely show the fact that one datum succeeds the other, but not necessarily after exactly one PET.]

For the column of processors that computes the product $h^T \cdot L$ we have to input the elements of the matrix L in Eqn. (9) (for $n = 4$)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ L[2,1] & 1 & 0 & 0 \\ L[3,1] & L[3,2] & 1 & 0 \\ L[4,1] & L[4,2] & L[4,3] & 1 \end{bmatrix}^{\dagger}$$

We shall read this data in by diagonals starting at the upper left corner; so that the input matrix is

$$I_0 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & L[2,1] & 1 \\ 0 & L[3,1] & L[3,2] & 1 \\ L[4,1] & L[4,2] & L[4,3] & 1 \end{bmatrix}$$

In each time interval, PET, we shall process one column of the matrix I_0, starting from the one at the right.

The first h processor requires an initializing datum at each PET, for which we shall enter a row of zeros:

$$I_1 = [0 \ 0 \ 0 \ 0]$$

The input to the top processor (see Figure 3) will be based on the weighting matrix in the left hand side of Eqn. (9), viz.

$$\begin{bmatrix} D[1] & & & \\ & D[2] & & \\ & & D[3] & \\ & & & D[4] \end{bmatrix}$$

arranged as follows (recall that we do not need to take the square root of this matrix, see Section III)

$$I_D = [D[1] \ D[2] \ D[3] \ D[4] \ * \ * \ *]$$

where the asterisks stand for *don't care* inputs. The matrices I_0 and I_D contain all the information we need for processing one measurement.

Description of the Output

The output from the column of processors at the right can be described by a matrix, O_0, which will be delayed $(n+\alpha+1)$PETs with respect to I_0 (i.e. $(n+\alpha-1)$PETs due to delay network plus 2 PETs due to the processing by the right and left columns of processors).

$$O_0 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & L_*[2,1] & 1 \\ 0 & L_*[3,1] & L_*[3,2] & 1 \\ L_*[4,1] & L_*[4,2] & L_*[4,3] & 1 \end{bmatrix}$$

The output from the top processor is given by O_D which will be delayed 3 PET with respect to I_D (i.e. the time it takes to propagate through three stages of the top processor):

$$O_D = [D_*[1] \ D_*[2] \ D_*[3] \ D_*[4] \ * \ * \ *]$$

This completes the description of the architecture for updating the covariance factors (Eqns. (2)-(4)) of the measurement update.

Updating the State Estimate

An important advantage of our architecture is that the same structure allows us to compute the updated measurement--viz. Eqn. (1); this fact does not follow from Eqn. (9), but from an understanding of the architecture. It is easy to show that if we include the state vector x in the input matrix I_0 as follows

$$I_0 = \begin{bmatrix} 0 & 0 & 0 & x[1] & 1 \\ 0 & 0 & x[2] & L[2,1] & 1 \\ 0 & x[3] & L[3,1] & L[3,2] & 1 \\ x[4] & L[4,1] & L[4,2] & L[4,3] & 1 \end{bmatrix},$$

and also initialize the h processors to include the scalar measurement y as follows

$$I_1 = [0 \ 0 \ 0 \ -y \ 0]$$

† Square brackets refer to an element of a vector or matrix.

then the output will have the same structure, and will include the updated state vector x_+.

$$O_1 = \begin{bmatrix} 0 & 0 & 0 & x_+[1] & 1 \\ 0 & 0 & x_+[2] & L_+[2,1] & 1 \\ 0 & x_+[3] & L_+[3,1] & L_+[3,2] & 1 \\ x_+[4] & L_+[4,1] & L_+[4,2] & L_+[4,3] & 1 \end{bmatrix}$$

The input of the diagonal elements and the corresponding output remain unchanged.

Initialization

For the algorithm to work properly we need to initialize the memory cells in the right column of processors with the value zero. This initialization can be done using the following input matrix before entering I_0

$$IC_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & * \\ 0 & * & * \\ * & * & * \end{bmatrix}$$

The structure of the matrix IC_0 will be preserved at the output. Additionally, we need to enter a string of $n-1$ zeros to initialize the first h processor

$$IC_1 = [\ 0\ 0\ 0\ ]$$

Implementation Issues

This architecture shows potential for a practical implementation: each processor can be implemented with standard chips now being developed (see Fisher et al. (1983) and S.Y. Kung (1984)). Furthermore, the flow of information follows a very simple and regular path. However other structures are also possible, including for example the use of CORDIC modules (see Section 2 below).

1.4 EXTENSION TO VECTOR CASE

As shown in Equation (9), in the vector case we have to zero the $p \times n$ matrix HL using as a reference the lower triangular matrix L_R. Each Givens transformation will zero out one element; however, in order to preserve the elements already zeroed we must proceed in a given order that must fulfill the following requirements:

1. ***All the elements above the one we want to zero (in the same column) must be zeroed first.*** This requirement comes by our choosing as a reference the *lower* triangular matrix L_R.
2. ***All the elements to the right of the one we want to zero (in the same row) must be zeroed first.*** This requirement comes by choosing a particular structure for the L matrix (lower triangular).

One way to fulfill both requirements is to zero the elements of the HL matrix by diagonals, starting in the upper right corner, and in every diagonal to start by zeroing out the uppermost element. Figure 5 shows the extension of the architecture to process p measurements (with $p = 3$ and $n = 4$). As expected, this architecture requires more processors that the scalar one, but presents the following additional advantages:

- The processors and structure are the same as for the scalar update.
- It takes time $O(max\{n, p\})$ (see below), as opposed to $O(np)$ for processing all the measurements using the scalar architecture.
- Processing the measurements one at a time requires the additional work of uncorrelating the measurements; using the architecture for the vector case, there is no need for uncorrelating the measurements.

Most of the considerations discussed in Section IV for the scalar measurement architecture apply directly here, such as the the way the coefficients of the H matrix are introduced in the memory cells. The input and output matrices I_0 and O_0 are exactly the same as for the scalar case. The initialization matrix will be also the same, IC_0. The vector of measurements, $\mathbf{y}$, ($p = 3$) is input as

$$I_1 = [\ 0\ 0\ 0\ -y[1]\ 0\ ]$$
$$I_2 = [\ 0\ 0\ 0\ -y[2]\ 0\ ]$$
$$I_3 = [\ 0\ 0\ 0\ -y[3]\ 0\ ]$$

To initialize the top processor in the columns of processors which perform the $H\ L$ product we will use

$$IC_1 = [\ 0\ 0\ 0\ ]$$
$$IC_2 = [\ 0\ 0\ 0\ *\ ]$$
$$IC_3 = [\ 0\ 0\ 0\ *\ *\ ]$$

where the asterisks show *don't care* values; here they are used to indicate proper timing. There is much more to be said about timing, operation counts and other complexity aspects, for which we must refer to the full paper.

2. Some Algorithms for Toeplitz Matrices

The material in this section is based on part of a paper to appear in *Modern Signal Processing and VLSI*, S.Y. Kung, H. Whitehouse and T. Kailath (eds.), Prentice-Hall, 1984.

The by now well-known linear predictive coding (LPC) methods for speech analysis are based on the hypothesis that speech waveforms can be modeled as the output of a linear time-invariant filter driven white noise.

The linear time-invariant filter can be modeled in many ways, but for a variety of reasons early attention was focused on using "all-pole" models, or equivalently on modeling speech as a stationary autoregressive discrete-time random process:

$$y_t + A_{N,1}y_{t-1} + \quad + A_{N,N}y_{t-N} = e_{N,t}$$

where $\{e_{N,t}\}$ is a zero-mean white noise process. The modeling problem is to choose the order N, the coefficients $\{A_{N,i}\}$ and the noise-variance, $R_{e,N}$ say, so as to best fit the observed speech signal $\{y_t,\ t \geq 0\}$.

The standard procedure is to form the so-called 'sample covariance' estimate of the second-order statistic,

$$R_i = E\ y_t y_{t+i}$$

of the stationary process $\{y_t,\ t \geq 0\}$, and to notice that the coefficients $\{A_{N,i}\}$ can be obtained by solving the so-called Yule-Walker equations

$$[A_{N,N} \cdots A_{N,1}\ 1] \begin{bmatrix} R_0 & R_1 & \cdots & R_N \\ R_1 & R_0 & & \\ \vdots & & & \\ R_N & \cdots & R_1 & R_0 \end{bmatrix} = [0 \quad \cdots \quad 0\ I] \tag{1}$$

There is room for disagreement with this formulation. However, one reason it is popular is that the special constant-along-diagonals (Toeplitz) nature of the coefficient matrix in (1) lends itself to a convenient fast recursive solution algorithm -- the so-called Levinson algorithm:

$$\begin{bmatrix} A_{N+1}(z) \\ B_{N+1}(z) \end{bmatrix} = \begin{bmatrix} z & -k_{N+1} \\ -zk_{N+1}z & 1 \end{bmatrix} \begin{bmatrix} A_N(z) \\ B_N(z) \end{bmatrix}, \tag{2}$$

$$A_0(z) = 1 = B_0(z)$$

where

$$A_N(z) = A_{N,N} + A_{N,N-1}z + \cdots + A_{N,1}z^{N-1} + z^N ,$$

$B_N(z)$ = the so-called 'reverse polynomial' of $A_N(z)$

$$= A_{N,N}z^N + A_{N,N-1}z^{N+1} - \cdots A_{N,1}z + 1 ,$$

and

$$k_{N+1} = \frac{A_{N,N}R_1 + A_{N,N-1}R_2 + \ldots + A_{N,1}R_N + R_{N+1}}{R_N^e} \tag{3a}$$

$$R_{N+1}^e = R_N^e(1 - k_{N+1}^2) , \quad R_0^e = R_0 \tag{3b}$$

The point is that we successively build up the solutions $\{A_{N,i}, N = 0,1,\ldots\}$, with the major computational burden being that of forming the 'inner product' for k_{N+1}, which requires N multiplications and N additions. Therefore, computing $\{k_1,\ldots,k_N\}$ will require of the order of

$$1 + 2 + \cdots + N = N(N+1)/2$$

or $O(N^2)$ elementary computations, which is an order of magnitude less than the $O(N^3)$ computations required to solve an arbitrary set of linear equations, i.e., one without the special Toeplitz structure of (1).

The Levinson algorithm can be used to produce a family of autoregressive models of increasing order, and we have to decide in some way on the appropriate order. There are various tests (e.g., Akaike's Information Criterion) and other considerations (e.g., practical design limits on integrated circuit implementations) involved in this choice, which we shall not elaborate here. What we do wish to emphasize is that Levinson algorithm has a special structure that allows us some flexibility in making a decision on the order.

The traditional way of implementing a filter to compute $e_{N,i}$ is via a transversal (tapped-delay-line) filter with coefficients $\{A_{N,0},\ldots,A_{N,N}\}$, see Figure 6; we should remark, to avoid confusion, that the transfer function of the filter that computes $e_{n,i}$ is not $A_N(z)$ but rather

$$A_{N,N} + A_{N,N-1}z^{-1} + \cdots + z^{-N} = z^{-N}A_N(z) .$$

However, if we were uncertain about our choice of order and wished to compute say $e_{N+2,i}$, then we would have not only a longer transversal filter but we would have to reset all the tap-gain coefficients from $\{A_{N,0},\ldots,A_{N,N}\}$ to $\{A_{N+2,0},\ldots,A_{N+2,N+2}\}$. There are many examples in which it is desirable to compute the filter response over a range of values of N before deciding on a fixed order, and this is not convenient with transversal filter implementations.

Now the Levinson recursion (2) shows that the $\{k_i, i = 1,\ldots,N\}$ along with the fact that $A_0(z) = B_0(z) = 1$, provide an alternative parametrization of the filter: knowing these values allows us to construct $A_N(z)$, and hence the coefficients $\{A_{N,i}\}$, from (2). If we wish to go to a different order, say $N+2$, in this parametrization we need only to add two more coefficients $\{k_{N+1},k_{N+2}\}$ without having to change any of the earlier values. This invariance property of the $\{k_i\}$-parametrization can be exploited by using a different implementation of the filter in terms of the $\{k_i, 1 \le i \le N\}$ rather than the $\{A_{N,i}\}$. Examination of (2) suggests that we can build the filter as a cascade of "lattice sections", as in Figure 7(a), or in a certain *normalized* (see Eq. (5) below) form as in Figure 7(b).

It is easy to calculate the inverse of the lattice filters in Figure 7 by using signal flow graph rules -- the result of doing this for Figure 7(b) is shown in Figure 8 and provides the "modeling" filter. It has the form of a discrete transmission line, and helps explain why the $\{k_i\}$ are often called *reflection coefficients*.

This physical interpretation suggests that we must have

$$|k_i| \le 1 \tag{4}$$

a fact that also follows from (3b) (since the variances $\{R_N^e\}$ must be nonnegative). There is a corresponding constraint on $A_N(z)$: the roots of the polynomial $z^N A(z)$ must lie within the unit circle, but this condition does not translate easily into bounds on the coefficients $\{A_{N,i}\}$. For this and other reasons the numerical properties of the (normalized) lattice filter tend to be better than those of the transversal filter. Therefore, it may not be surprising that Texas Instruments chose to use this structure when building their very successful speech synthesis chip (Wiggins and Brantingham (1978)): the modular structure, local interconnections and rhythmic data flow all make for convenient VLSI implementation (Mead and Conway (1980), H. T. Kung (1979), (1982), S. Y. Kung et al. (1982)).

CORDIC Implementations

Each section in Figure 7(a) requires two multiplications and two additions; the multiplications tend to be more expensive and, therefore, rearrangements have been devised that use scaling to manage with only one multiplication (see e.g., Markel and Gray (1976, Sec. 5.5)). For numerical reasons, one often goes to a section with four multipliers, as in Figure 7(b). Somewhat ironically, it turns out that this section can in fact be implemented without using any explicit multiplications at all, by using the so-called CORDIC (Coordinate Rotation Digital Computer) implementations, based on the property shown in (5) below.

Such circuits have been proposed and already used in several applications (e.g., by Despain (1974), (1979), Haviland and Tuszynski (1980), see also Schelin (1983)); in recent years they have been suggested for a variety of one- and two-dimensional VLSI computing structures of the systolic and wavefront array types -- see, e.g., the papers of Ahmed, Delosme and Morf (1982), Rao and Kailath (1983a,b), Dewilde et al. (1984), and also the thesis of Ahmed (1982) and Delosme (1982).

Parallel Computation and the Schur Algorithm

A natural question with VLSI technology is to explore the question of how much the determination of the reflection coefficients can be speeded up by using parallel computation -- say with N processors working together. The hope is that if a processor takes one unit of time for each elementary computation, then we would hope to use N processors to obtain the answer in time $O(N)$ as compared to $O(N^2)$ with a single serial processor.

However, it is not hard to see that a parallel implementation of the Levinson algorithm will take time $O(N \log N)$ because of the inner product operation needed to form the $\{k_i\}$ -- N processors can carry out the N multiplications needed to form k_{N+1} in parallel in one time unit, but the additions required will take at least $\log N$ steps.

This seems disappointing, but there is hope. It turns out that there is a mathematically equivalent algorithm, traceable to Schur (1917), that avoids the inner product step in forming the $\{k_i\}$ and thus will allow implementation in time $O(N)$ with N processors.

The Schur algorithm was originally presented in 1917 as a test to see if a power series was analytic and bounded in the unit disc. In our problem it reduces to a simple three step procedure (see Dewilde, Vieira and Kailath (1978) and Lev-Ari and Kailath (1984)) for the computation of the $\{k_i\}$ associated with a covariance sequence $\{R_0,R_1,\ldots,R_N\}$.

Step 1: Start with a generator matrix (superscript * denotes matrix transpose)

$$G_0^* := \begin{bmatrix} R_0 & R_1 & \cdots & R_N \\ 0 & R_1 & \cdots & R_N \end{bmatrix}$$

and shift the first column down one row to get

$$G_1^* = \begin{bmatrix} 0 & R_0 & \cdots & R_{N-1} \\ 0 & R_1 & \cdots & R_N \end{bmatrix}$$

Step 2: Compute k_1 as the ratio of the (2,2) and (2,1) entries of G_1.

Step 3: Form a matrix

$$\Theta(k_1) = \frac{1}{\sqrt{1-k_1^2}} \begin{bmatrix} 1 & -k_1 \\ -k_1 & 1 \end{bmatrix} \tag{5a}$$

and apply it to G_1 to obtain a new Schur-reduced generator of the form

$$G_1^* = \Theta(k_1) G_1^* = \begin{bmatrix} 0 & x & x & \cdots & x \\ 0 & 0 & x & \cdots & x \end{bmatrix} \tag{5b}$$

where the x denotes elements whose exact value is not relevant at the moment; Θ is known as a J-rotation matrix because

$$\Theta(k)\, J\, \Theta^*(k) = J \quad , \quad J = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} .$$

Its role is to rotate in the J-metric (i.e., to hyperbolic) the 2^{nd} row of G to lie along the first coordinate direction.

Now we can repeat Steps 1, 2, 3 to obtain k_2 and a new reduced generator matrix G_2. And so on.

It can be shown that the $\{k_i\}$ computed in the Schur algorithm are exactly the same quantities as defined before in the Levinson algorithm. However, note that the Schur algorithm only requires sets of 2×1 row vector by 2×2 matrix multiplications, which can be carried out in parallel at each stage. Therefore, we will need only $O(N)$ time units to carry out the Schur algorithm with N processors, as compared to $O(N \log N)$ for the Levinson algorithm.

In fact, a parallel and pipelined lattice VLSI computing structure based on the Schur algorithm has already been designed and built (see Kung and Hu (1983).

Cholesky Factorization

The Schur algorithm is also closely connected with the so-called Cholesky factorization of the Toeplitz covariance matrix $\mathbf{R}$ in (1). Such factorizations are important in many calculations involving the corresponding stochastic process. The Cholesky factor, say $\mathbf{C}$, is the unique lower triangular matrix with positive diagonal entries such that

$$\mathbf{R} = \mathbf{C}\mathbf{C}^*$$

It turns out that $\mathbf{C}$ is immediately determined by the Schur algorithm:

$$\text{the } i\text{-}th \text{ column of } \mathbf{C} = \text{the first column of } G_i \ , \tag{6}$$

where the $\{G_i\}$ are the reduced generator matrices obtained in the Schur algorithm. Thus we have a fast algorithm, using $O(N^2)$ computations, for Cholesky factorization of a Toeplitz matrix, as compared to $O(N^3)$ in general, (see Bareiss (1969), Morf (1970), Rissanen (1973)).

Transmission Line Interpretations and Inverse Scattering Problems

A good indication of a natural connection between the Schur algorithm and potential VLSI or other implementation methods can be obtained from the graphical representations of the Schur algorithm shown in Figure 9. Then the above result on the relation between $\mathbf{C}$ and the $\{G_i\}$ means that the values at the input of the i-th delay element are the entries of the i-th column of the Cholesky factor $\mathbf{C}$. Consequently, we can also say that the values at the inputs of the delay elements at any time t are the entries of the t-th row of the Cholesky factor $\mathbf{C}$. [Incidentally, this interpretation removes much of the mystique sometimes associated with the so-called fast Cholesky-by-column and fast Cholesky-by-row algorithms.]

This transmission-line interpretation suggests that the computations could be carried out by exciting structures as in Figure 9 with light or other electromagnetic waves, depending upon the medium chosen for implementation (e.g., fiber optics or acoustic waves in solids, etc.).

In fact, we might mention that starting with the graphical representations in Figure 9, and using some fundamental wave propagation and energy conservation concepts, we can obtain simple proofs of all the above results and in the process develop intimate and useful connections between transmission line theory, Cholesky and inverse Cholesky factorization, inverse scattering theory and the Schur and dual Schur algorithms (see Bruckstein and Kailath (1983)). This transmission line picture was also helpful in obtaining certain orthogonal cascade implementations of ARMA (pole-and-zero) filters with excellent numerical properties (Rao and Kailath (1983)) and excellent potential for VLSI implementation.

On the other hand, we can approach the results from quite another direction by noting that Cholesky factorization is actually a way of testing for positive definiteness of a matrix; if we now ask what matrix structures lend themselves to easy testing in this way, we shall be led to a class of matrices (with "displacement" structure) of which the Toeplitz family is only one special case (Lev-Ari and Kailath (1984)). By this route, all the previously mentioned results on parallel computation and transmission line interpretations and implementations can be carried over to a much larger class of problems as we shall briefly review in the next section.

3. Minimal Realizations and Decoding of BCH Codes

Consider a discrete-time system with transfer function

$$H(z) = \sum_1^{\infty} h_i z^{-i}$$

so that the $\{h_i\}$ define the impulse response of the system. Suppose, to take a very simple case, that we know the system is finite dimensional of order n, and that we wish to find polynomials $\{a(z), b(z)\}$ such that

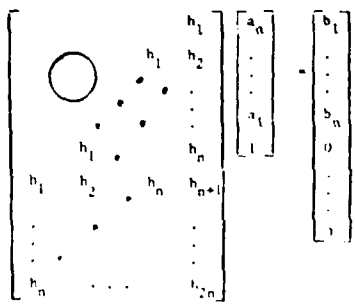
$$H(z) = b(z) / a(z) \ ,$$

where

$$a(z) = z^n + a_1 z^{n-1} + \ldots + a_n$$

$$b(z) = b_1 z^{n-1} + \cdots + b_n \ .$$

Then it is easy to check that we have the relation



(1)

From this set of equations, it is clear that $b(z)$ may be easily found via back substitution once $a(z)$ is known. Thus, the main work involved is to find $a(z)$ by solving the Hankel matrix equation,

$$\begin{bmatrix} h_1 & \cdots & h_{n+1} \\ \vdots & & \\ h_n & & h_{2n} \end{bmatrix} \begin{bmatrix} a_n \\ \vdots \\ a_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2}$$

General methods of solving n linear equations in n unknowns require $O(n^3)$ elementary computations. However, here the special Hankel structure can be exploited to solve the equations with $O(N^2)$ computations. This result was perhaps first noted in the coding theory literature, where Berlekamp (1968) developed an important decoding algorithm for BCH codes, which essentially involved the solution of Hankel linear equations over Galois field with elements $\{0,1\}$; Massey (1969) later showed the relation to minimal realization problems over this field. The Berlekamp-Massey algorithm can be in its simplest form described as follows:

$$a(z) = \Lambda^{(2n)}(z)$$

where

$$\begin{bmatrix} \Lambda^{(k+1)}(z) \\ B^{(k+1)}(z) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_{k+1} z \\ \Delta_{k+1}^{-1}\delta_{k+1} & (1-\delta_{k+1})z \end{bmatrix} \begin{bmatrix} \Lambda^{(k)}(z) \\ B^{(k)}(z) \end{bmatrix}, \quad \begin{bmatrix} \Lambda^{0}(z) \\ B^{0}(z) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{3}$$

$$\delta_{k+1} = 1 - \delta_k \quad \delta_0 = 0$$

$$\Delta_{k+1} = \sum_{j=0}^{\lfloor k/2 \rfloor} \Lambda_j^{(k)} h_{k+1-j} \tag{4}$$

By "simplest form" we mean that the coefficient matrix is assumed to have all leading minors nonzero. This unrealistic assumption can be removed, at the cost of some further complexity in the algorithm. We ignore this (important) refinement here, because our point is that the inner product in (4) prevents effective parallel implementation of the Berlekamp-Massey algorithm: with n processors we will need $O(n\log n)$ elementary computations rather than $O(n)$ as we might expect.

An alternative method of solving the linear equations can be based on attempting a triangular factorization of the coefficient matrix in the form

$$\begin{bmatrix} h_1 & \cdots & h_{n+1} \\ \vdots & & \\ h_n & & h_{2n} \end{bmatrix} \begin{bmatrix} 1 & x & x & \cdots & a_n \\ 0 & 1 & x & & \\ \vdots & & 1 & & \\ 0 & & & & a_1 \\ 0 & \bigcirc & & & 1 \end{bmatrix} = \begin{bmatrix} x & & & \bigcirc & \\ \vdots & & & & \\ x & \cdots & & & 0 \end{bmatrix}. \tag{5}$$

More compactly, we denote this as

$$M(n,n+1)P(n+1,n+1) = Q(n,n+1)$$

The factorization of M is generated by recursively computing the columns of P. This leads to recursions of the form

$$p_{k+1}(z) = (z - \gamma_k)p_k(z) - \delta_k p_{k-1}(z) \tag{6}$$

where

$$\delta_k = \frac{\beta_k^{(0)}}{\beta_{k-1}^{(0)}}, \quad \gamma_k = \frac{\beta_k^{(1)}}{\beta_k^{(0)}} - \frac{\beta_{k-1}^{(1)}}{\beta_{k-1}^{(0)}},$$

$p_k(z)$ is a polynomial formed from the $(k+1)^{st}$ column of P with the top element assigned the power z^0 and so forth. $\beta_k^{(i)}$ is the ith element below the main diagonal in the $(k+1)^{st}$ column of Q, e.g., $\beta_k^{(0)}$ is the element in the $(k+1)^{st}$ column that is on the main diagonal. This "three-term" recursion is standard in the theory of orthogonal polynomials and in fact the above polynomials are already known in that literature as Lanczos polynomials. Again the classical theory has to be extended when $M(n,n+1)$ can have zero leading minors; this can be done by replacing the term $(z - \gamma_k)$ in (6) by a polynomial in z of higher degree; see Kung ((1977), Ch. 4). The main point here, however, is that the coefficients $\{\beta_k, \gamma_k\}$ in (6) arise naturally in each step of the factorization, and no inner products are involved. Therefore, as noted by Todd Citron, the Lanczos recursions are better suited for parallel implementation. In his thesis research at Stanford, Citron has developed a fast highly parallel implementation of the generalized Lanczos recursions for solving the partial realization and BCH decoding problems.

4. VLSI Implementations of Signal Processing Operations

In the last few years, there has been great activity in the area of systolic array implementation of basic signal processing operations such as convolution, polynomial multiplication and division, matrix multiplication and LU and Cholesky matrix decomposition (see, e.g., H. T Kung (1982) and the chapter by Kung and Leiserson in Mead and Conway (1980)). Systolic arrays have the advantages of modularity, local interconnectedness and regular data flow and therefore are being studied as prime candidates for VLSI implementation. However, numerical issues important for actual implementation, such as sensitivity to coefficient truncation and rounding and susceptibility to overflow and limit-cycle oscillations, do not appear to have been addressed as yet. In Rao and Kailath (1983b), it is shown that most presently known systolic arrays would in fact suffer from such problems, because they can be interpreted as rearrangements of the so-called controller and observer (or direct) canonical form of linear system theory; the rearrangements are essentially those required to make these canonical forms pipelineable. In linear system theory, it is known that numerical problems can be reduced by using cascade realizations, and that in fact overflow and limit cycle oscillations can be completely avoided by using cascades of orthogonal sections. The Schur algorithm described in Section 2 can be used to derive two new classes of pipelined, systolic ARMA digital filters of this type. In the AR (all pole) case, they are exactly the lattice filters shown in Figs. 7 and 8. In the ARMA (pole-zero) case, we have single input, two-output structures of the types shown in Fig. 10; these are derived by applying energy-conservation constraints to the transmission-line analysis mentioned at the end of Section 2.

There is a tradeoff between chip area and computation time in VLSI implementation and several authors have developed lower bounds for the area in terms of a quadratic function of the computer rate. Computer-science analyses of signal processing chips have not generally taken into account the rationality of the transfer functions that may be involved; by doing this, tighter lower bounds have been obtained in Rao and Kailath (1983b). It turns out that the observer and controller canonical forms, and the parallel cascade and orthogonal cascade structures mentioned above all meet the complexity lower bounds; however, the widely-used (and very low sensitivity) wave-digital filters and most noncanonical state-space realizations do not.

5. Computer-Aided Engineering Analysis and Design

A less analytical, but no less significant development in the VLSI era is the growing availability of desk-top computers and work stations with powerful simulation, graphical and computational capabilities. Dramatic improvements can be expected in the level and sophistication of advanced theoretical techniques that can be brought to bear, in often user-transparent and very user-friendly ways, on practical problems. Not the least significant will be the development of the proper combination of human skills and machine capabilities that Professor Rosenbrock presciently identified several years ago (1980 CDC Plenary Lecture; also in *Systems & Control Letters*, Vol. 1, pp. 2-6, July 1981) as an essential ingredient in rightly shaping the relationship between man and machine, automation and society.

6. Concluding Remarks

This written version is only meant to provide some more background and detail on the major technical topics

that I expect to use, to varying degrees, in the actual lecture. I am indebted to several colleagues and students, including E. Ahmed, T. Citron, J. Delosme, P. Dewilde, N. Gupta, H. Jagadish, J. Jover, S. Y. Kung, D. T. Lee, H. Lev-Ari, M. Morf, S. K. Rao, S. Shah and H. Whitehouse, for many patient discussions and talks that have helped shape my understanding of a fascinating new set of problems and applications. I am also grateful to the conference organizers, and especially S. Marcus and J. Melsa, for the invitation to present this lecture.

REFERENCES

Agee, A.S. and Turner, R.H. (1972), White Sands Missile Range, Tech. Rept. 38.

Ahmed, H.M. (1082), Ph.D. Dissertation, Electrical Engineering, Stanford University, Stanford, CA.

Ahmed, H.M., Delosme, J-M. and Morf, M. (1982), *Computer*, Vol. 15, No. 1, pp. 65-82.

Andrews, A. (1981), *Proc. Int. Conf. on Parallel Processing*, Columbus, Ohio, pp. 216-220.

Bareiss, E.H., (1969), *Numer. Math.*, Vol. 13, pp. 404-424.

Berlekamp, E. R. (1968), *Algebraic Coding Theory*, McGraw-Hill.

Bierman, G.J. (1975), *Proc. IEEE Conf. on Decision & Control*, Houston, Texas, pp. 337-346.

Bierman, G.J. (1977), *Factorization Methods for Discrete Sequential Estimation*, Academic Press: New York, 1977.

Browning, S.A. (1980), Technical Report 3760, Computer Science Department, California Inst. of Tech., Pasadena, CA.

Bruckstein, A. and T. Kailath (1983), Tech. Rept., Inform Systems Lab., Stanford University.

Carlson, N.A. (1973), *AIAA Journal*, Vol. 11, No. 9, pp.1259-1265.

Delosme, J.M. (1982), Ph.D. Dissertation, Electrical Engineering, Stanford University, Stanford, CA.

Dennis, J.B. (1980), *Computer*, Vol. 13, No. 11, pp. 48-56.

Despain, A.M., (1974), *IEEE Trans. Computers*, Vol. C-23, pp. 993-1001.

Despain, A.M., (1979), *IEEE Trans. Computers*, Vol. C-28, no. 5, pp. 333-341.

Dewilde, P., Vieira, A. and Kailath, T. (1978) *IEEE Trans. Circuits & Systems*, Vol. CAS-25, no. 9, pp. 663-675, Sept. 1978.

Dewilde, P., Deprettere, E. and Nouta, R. (1984), in *VLSI and Modern Signal Processing*, S. Kung, H. Whitehouse and T. Kailath eds., Prentice-Hall.

Dyer, P. and McReynolds, S.R. (1969), *J. Opt. Theory and Applications*, Vol. 3, No. 6, pp. 444-458.

Fisher, A.L., Kung, H.T., Monier, L.M. and Walker, H. (1983), *Third Caltech Conference on VLSI*, ed. R. Bryant, Computer Science Press: Rockville (MD), pp. 287-302.

Gentleman, W.M. (1973), *J. Inst. Maths. Applics.*, Vol. 12, pp.329-336.

Gentleman, W.M. and Kung, H.T. (1981), Systolic Arrays", *Real Time Signal Processing IV, SPIE*, Vol. 298, pp. 19-26.

Haviland, G.L. and A. Tuszynski, (1980), *IEEE Trans. Computers*, Vol. C-29, No. 2, pp. 68-79.

Jover, J. M. and T. Kailath, submitted *IFAC 1984*, Budapest, Hungary.

Kailath, T. (1982), Course Notes for EE378, Stanford University.

Kalman, R. (1960), *Trans. ASME*, Vol. 82D, pp. 35-50.

Kaminski, P.G., Bryson, A.E. Jr and Schmidt, S.F. (1971), *IEEE Trans on Automatic Control*, Vol. AC-16, No. 6, pp. 727-736.

Kung, H.T. (1982), *Computer*, Vol. 15, No. 1, pp. 37-46.

Kung, S.Y. (1977), Ph.D. Dissertation, Electrical Engineering, Stanford University, Stanford, CA.

Kung, S.Y., Arun, K.S., Gal-Ezer, R.J. and Bhaskar Rao, D.V. (1982), *IEEE Trans on Computers*, Vol. C-31, No. 11, pp. 1054-1066.

Kung, S.Y. and H. Hu, (1983), *IEEE Trans Acoust., Speech and Sig. Proc.*, Vol. ASSP-31, no. 1, pp. 66-75.

Kung, S.Y. (1984), in *VLSI and Modern Signal Processing*, eds S.Y. Kung, H.J. Whitehouse, and T. Kailath, Prentice-Hall, N.J.

Kung, S.Y., Whitehouse, H.J., and Kailath, T., editors (1984), *VLSI and Modern Signal Processing*, Prentice-Hall, N.J.

Lee, D.T., (1980), Ph.D. Dissertation, Department of Electrical Engineering, Stanford University, Stanford, CA.

Lev-Ari, H., (1983a), Ph.D. Dissertation, Stanford University, Stanford, CA.

Lev-Ari, H., (1983b), *IEEE Inter'l. Conference on ASSP*, pp. 455-458, Boston, MA, April 1983.

Lev-Ari, H. and Kailath T. (1984), *IEEE Trans. Inform. Thy.*, Vol. IT-30, no. 1, 1984.

Markel, J. and Gray, A. H., Jr. (1976), *Linear Prediction of Speech*, Springer-Verlag.

Massey, J.L. (1969), *IEEE Trans. Inform. Thy.*, Vol. IT-15, pp. 122-127.

Mead, C. and Conway, L. (1980), *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA.

Morf, M. (1970), Internal Memorandum, Information Systems Lab., Stanford University, Stanford, CA.

Morf, M. and Kailath, T. (1975), *IEEE Trans. on Automatic Control*, Vol. AC-20, No. 4, pp. 487-497.

Rao, S.K. and T. Kailath, (1983a), "Orthogonal Digital Filters for VLSI Implementation," submitted for publication.

Rao, S.K. and T. Kailath, (1983b), "VLSI and the Digital Filtering Problem," submitted for publication.

Rissanen, J. (1973), *Math. Comp.*, Vol. 27, pp. 147-154.

Schelin, C.W. (1983), *Amer. Math. Monthly*, Vol. 90, pp. 317-325.

Schur, I. (1917), *Journal fur die Reine und Angewandte Mathematik*, Vol. 147, pp. 205-232, Berlin.

Thornton, C.L. (1976), Jet Propulsion Laboratory, Pasadena, Technical Memo 33-798.

Wiggins, R. and Brantingham, L. (1978), *Electronics*, Vol. 51, no. 18, pp. 109-116.

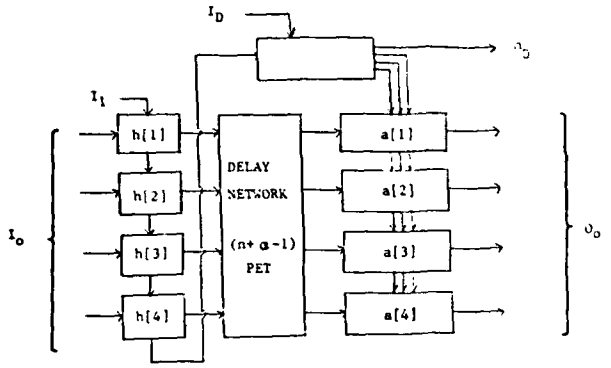


Figure 1. Linear Network for Scalar Measurement ($n = 4$)

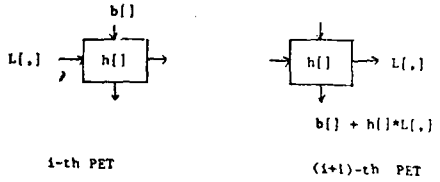


Figure 2. Processor in Column that Performs Inner Product

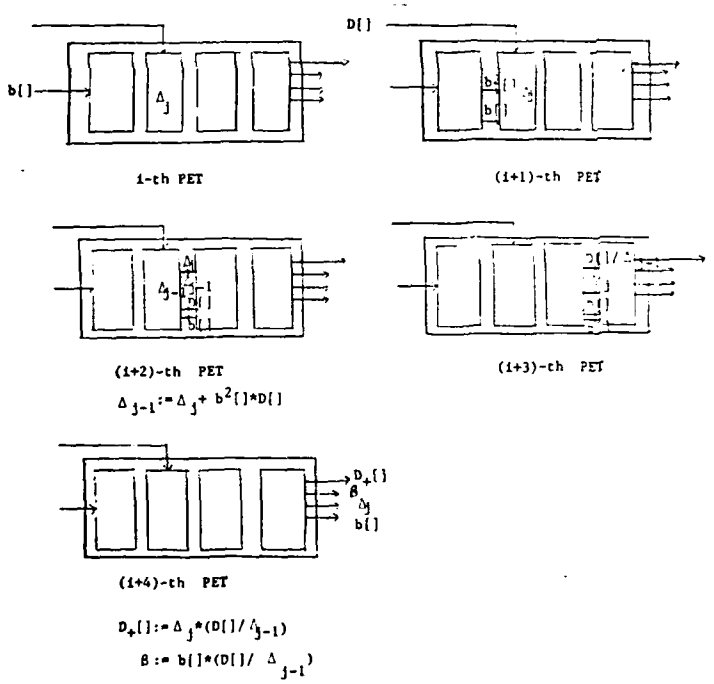


Figure 3. Processor for Computing the Transformation Parameters

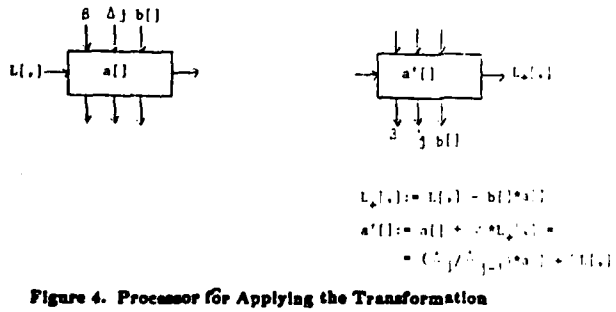


Figure 4. Processor for Applying the Transformation

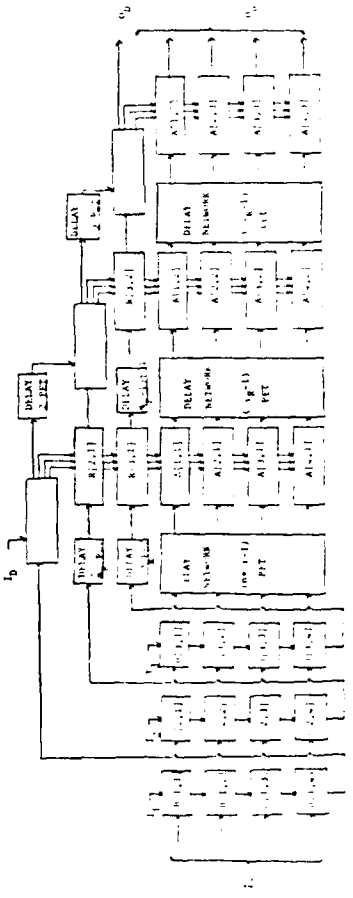
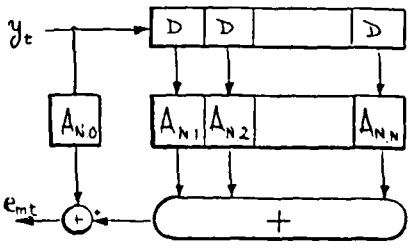
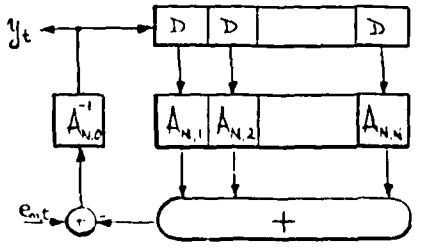


Figure 5. Architecture for Vector Measurement ($n = 4$, $p = 3$)

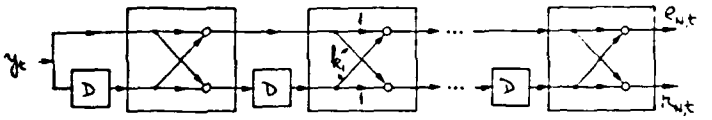


a) Whitening Filter

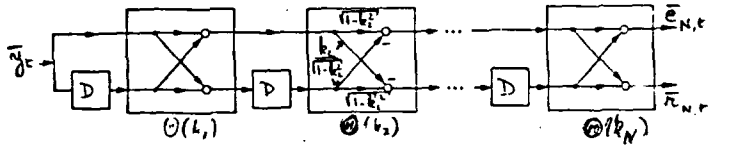


b) Modeling Filter

Fig. 6. Tapped Delay Line Implementation

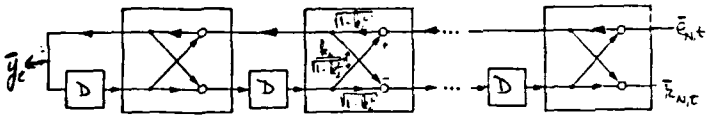


a) Unnormalized Form

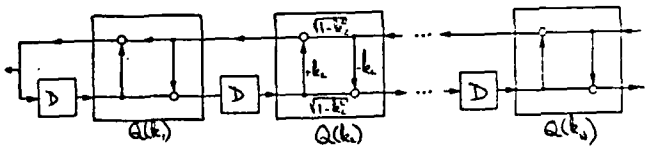


b) Normalized Form

Fig. 7. Lattice Filter Implementations of Whitening Filters

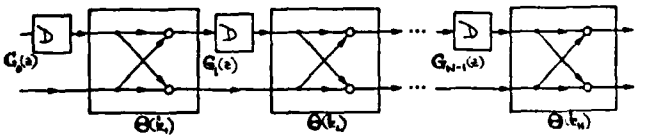


a) Feedback Lattice Form - Normalized

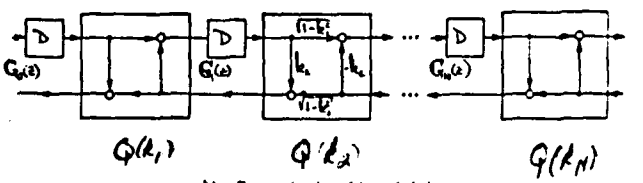


b) Transmission Line Form

Fig. 8. Modeling Filters



a) Direct Form



b) Transmission Line Model

Fig. 9. Representations of the Schur Algorithm

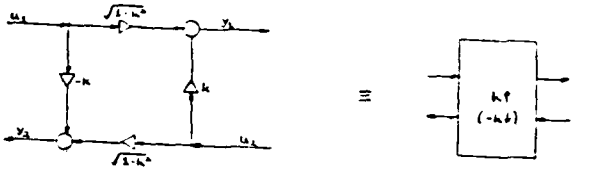


Fig. 10a). The Givens Orthogonal Rotational Module. The equivalent symbol on the right indicates the direction of the $\sin(-k)$ multiplier in the implementation. These modules are used as basic building blocks in the structures in b) and c).

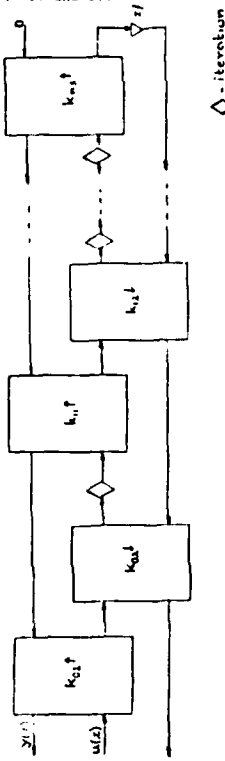


Fig. 10b). Orthogonal Digital Filter obtained via a Direct Embedding Procedure.

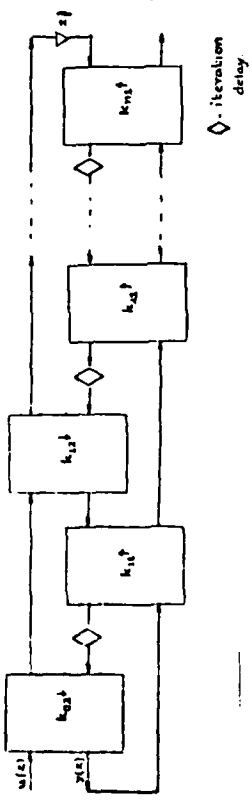


Fig. 10c). Orthogonal Digital Filter obtained via an Autoregressive [illegible] Procedure.