[8] V. F. Fusco, *Microwave Circuits Analysis and Computer-Aided Design*. Englewood Cliffs, Prentice Hall, NJ, 1987.
[9] E. O. Hammerstadt and F. Bekkadal, "A microstrip handbook," *ELAB Report*, STF 44 A74169, Univ. of Trondheim, Norway, 1975, pp. 98-110.
[10] M. V. Schneider, "Microstrip lines for microwave integrated circuits," *Bell System Tech. J.*, vol. 48, no. 5, pp. 1421-1444, May-June 1969.
[11] R. P. Owens, "Accurate analytical determination of quasi-static microstrip line parameters," *The Radio and Electronic Engineer*, vol. 46, no. 7, pp. 360-364, July 1976.
[12] E. M. Siomacco and M. Tummala, "Parametric modeling of picosecond pulse propagation on integrated circuit Interconnections," presented at 32nd Midwest Symp. on Circuits and Systems, University of Illinois, pp. 1030-1033, Aug. 1989.
[13] A. G. Evans and R. Fischel, "Optimal least squares time-domain synthesis of recursive digital filters," *IEEE Trans. Audio and Electroacoust.*, vol. AU-21, pp. 61-65, Feb. 1973.
[14] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *J. Appl. Phys.*, vol. 19, pp. 55-83, 1948.
[15] D. R. Bowman, *High Speed Polycrystalline Silicon Photoconductors for On-Chip Pulsing and Gating*, Ph.D. dissertation, Stanford University, 1985.
[16] E. M. Siomacco, *Parametric Modeling and Estimation of Pulse Propagation on Microwave Integrated Circuit Interconnections*, Ph.D. dissertation, Naval Postgraduate School, 1990.

Dual-State Systolic Architectures For Up/Downdating RLS Adaptive Filtering

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

***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 based on successively performing Givens rotations for updating and hyperbolic rotations for downdating. Due to 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 the dual-state systolic triarray. Efficient implementation on the evaluation of optimal residuals is also considered. This systolic architecture is promising for the VLSI implementation of fixed size sliding-window recursive least-squares estimations.**

I. Introduction

Consider a fixed-windowed least-squares (LS) problem, $X(n)w(n) \approx y(n)$, where $X(n) = [x_{n-l+1}^T, x_{n-l+2}^T, \cdots, x_n^T]^T \in R^{l \times p}$ is a data matrix with elements taken either from a single or p time-indexed multichannel data sequences, and $y(n) = [y_{n-l+1}, y_{n-l+2}, \cdots, y_n]^T \in R^l$ is the desired response vector. We denote l as the window size, p as the order of the system, and n as the time index ($n \geq l$ is assumed). The LS problem is to find a $p \times 1$ optimum coefficient vector $\hat{w}(n) \in R^p$, such that the Euclidean norm of its associated residual $e(n) = X(n)w(n) - y(n)$ is minimized.

Manuscript received April 25, 1991; revised October 8, 1991. This work was supported in part by the National Science Council of the ROC; by the Minta Martin Award, College of Engineering, University of Maryland; by the National Science Foundation under Grant NCR-8814407; and by the UCMICRO Program. This paper was recommended by Associate Editor K. K. Parhi.

S. F. Hsieh is with the Department of Communication, Engineering, National Chiao Tung University, Hsinchu, Taiwan 30050.
K. J. R. Liu is with the Systems Research Center, Electrical Engineering Department, University of Maryland, College Park, MD 20742.
K. Yao is with the Electrical Engineering Department, UCLA, Los Angeles, CA 90024-1594.

IEEE Log Number 9201312.

In adaptive signal processing QRD has been proven to be an effective tool in performing this recursive LS problem [2], [3], [7], [8]. 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. Recently, some efficient up/downdating algorithms have been proposed [4]-[6]. But work on efficient implementations and architectures for a fixed-windowed RLS filtering with such up/downdating is still fragmentary. In this paper, we propose a *dual-state* systolic array which is suitable for VLSI designs, to perform fixed-windowed RLS estimation. Efficient schemes to obtain optimal residual have not been fully addressed for the windowed RLS estimation. Along this direction, we consider the feasibilities and limitations based on systolic implementations.

In Section II, the basic up/downdating RLS estimation is considered, followed by the dual-state systolic architecture in Section III. In Section IV, we consider the recursive estimation of optimal residual with systolic implementation. Conclusions are then given in Section V.

II. Windowed RLS Estimation

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

$$Q(n)[X(n) \vdots y(n)] = \begin{bmatrix} R(n) & \vdots & u(n) \\ 0 & \vdots & v(n) \end{bmatrix} \tag{1}$$

where $Q(n) \in R^{l \times l}$ is orthogonal and the Cholesky factor $R(n) \in R^{p \times p}$ is upper triangular. Thus the optimum $\hat{w}(n)$ is given by $R(n)\hat{w}(n) = u(n)$.

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

$$\begin{bmatrix} R(n) & \vdots & u(n) \\ \mathbf{x}_{n+1}^T & \vdots & y_{n+1} \\ \mathbf{x}_{n-l+1}^T & \vdots & y_{n-l+1} \end{bmatrix} \tag{2}$$

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

$$G_{p,p+1} \cdots G_{2,p+1} G_{1,p+1} \begin{bmatrix} R(n) & \vdots & u(n) \\ \mathbf{x}_{n+1}^T & \vdots & y_{n+1} \\ \mathbf{x}_{n-l+1}^T & \vdots & y_{n-l+1} \end{bmatrix} = \begin{bmatrix} \tilde{R}(n+1) & \vdots & \tilde{u}(n+1) \\ 0 & \vdots & v_1(n+1) \\ \mathbf{x}_{n-l+1}^T & \vdots & y_{n-l+1} \end{bmatrix} \tag{3}$$

$$H_{p,p+2} \cdots H_{2,p+2} H_{1,p+2} \begin{bmatrix} \tilde{R}(n+1) & \vdots & \tilde{u}(n+1) \\ 0 & \vdots & v_1(n+1) \\ \mathbf{x}_{n-l+1}^T & \vdots & y_{n-l+1} \end{bmatrix} = \begin{bmatrix} R(n+1) & \vdots & u(n+1) \\ 0 & \vdots & v_1(n+1) \\ 0 & \vdots & v_2(n+1) \end{bmatrix}. \tag{4}$$

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

$$G_{i,p+1}\begin{bmatrix} \vdots \\ \alpha_i \\ \vdots \\ \alpha_{p+1} \\ \cdot \end{bmatrix} = \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ 0 & c_i & 0 & s_i & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & -s_i & 0 & c_i & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \vdots \\ \alpha_i \\ \vdots \\ \alpha_{p+1} \\ \cdot \end{bmatrix} = \begin{bmatrix} \vdots \\ \sqrt{\alpha_i^2 + \alpha_{p+1}^2} \\ \vdots \\ 0 \\ \cdot \end{bmatrix} \tag{5}$$

where $c_i = \alpha_i / \sqrt{\alpha_i^2 + \alpha_{p+1}^2}$ and $s_i = \alpha_{p+1} / \sqrt{\alpha_i^2 + \alpha_{p+1}^2}$.

Similarly, $a(p+2) \times (p+2)$ hyperbolic rotation matrix $H_{i,p+2}$ is used to zero out the $(p+2, i)$th element of the matrix in (3),

$$H_{i,p+2}\begin{bmatrix} \vdots \\ \alpha_i \\ \vdots \\ \alpha_{p+2} \end{bmatrix} = \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ 0 & \tilde{c}_i & 0 & -\tilde{s}_i & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & -\tilde{s}_i & 0 & \tilde{c}_i & 0 \end{bmatrix}\begin{bmatrix} \vdots \\ \alpha_i \\ \vdots \\ \alpha_{p+2} \end{bmatrix} = \begin{bmatrix} \vdots \\ \sqrt{\alpha_i^2 - \alpha_{p+2}^2} \\ \vdots \\ 0 \end{bmatrix} \tag{6}$$

where $\tilde{c}_i = \alpha_i / \sqrt{\alpha_i^2 - \alpha_{p+2}^2}$ and $\tilde{s}_i = \alpha_{p+2} / \sqrt{\alpha_i^2 - \alpha_{p+2}^2}$. Let $H(n+1) = H_{p,p+2} \cdots H_{1,p+2}$ and $G(n+1) = G_{p,p+2} \cdots G_{1,p+2}$. By combining (3) (4), we have

$$H(n+1)G(n+1)\begin{bmatrix} R(n) & \vdots & u(n) \\ \mathbf{x}_{n+1}^T & \vdots & y_{n+1} \\ \mathbf{x}_{n-l+1}^T & \vdots & y_{n-l+1} \end{bmatrix} = \begin{bmatrix} R(n+1) & \vdots & u(n+1) \\ 0 & \vdots & v_1(n+1) \\ 0 & \vdots & v_2(n+1) \end{bmatrix}. \tag{7}$$

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. In a multichannel filtering problem, for every sensor (i.e., column of the data matrix) there is a delay buffer of window size l to queue up the data. Therefore, each data will be first fetched and processed (updated) and then stays in the queuing 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 that alternatively take in new data and old data. The clock rate for the processors is set at twice the input data 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). 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

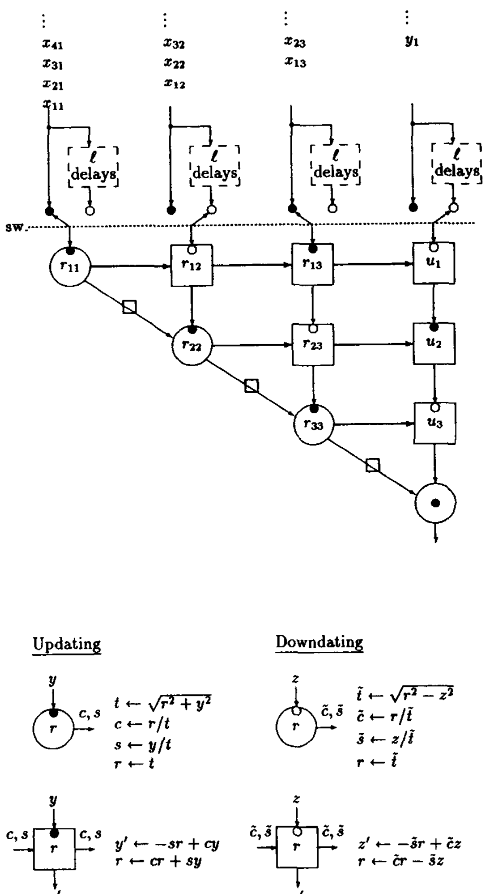

Fig. 1. Dual-state systolic array for windowed-RLS problems.

updating and downdating data interleaved to form a sequence of up/down-dating wavefronts which will then impact 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 immediately behind, and 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. Thus 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. In all, for each time snapshot, we see all processors are doing updating and downdating evenly distributed over the entire triarray, and for the 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 then propagate pairwise toward the lower-right direction in the triarray.

IV. Recursive Estimation of Optimal Residuals

In many applications, the optimal weight coefficient vector may not be of direct interest. Instead, we may be interested in the newest optimal residual $\hat{e}_n$, which is the last (i.e., the lth) element of $\hat{e}(n)$. In this section, we consider an efficient implementation to obtain the newest residuals under the up/downdating operations.

From (1), we can separate $Q(n)$ into two terms as $Q(n) = [Q_1^T(n), Q_2^T(n)]^T$, where $Q_1(n) \in R^{p \times l}$, $Q_2(n) \in R^{(l-p) \times l}$. We can also rewrite the optimal residual vector as

$$\hat{e}(n+1) = -Q_2^T(n+1)\begin{bmatrix} v_1(n+1) \\ v_2(n+1) \end{bmatrix}. \tag{8}$$

Thus, a basic issue is the efficient recursive evaluation of $Q_2(n+1)$. Define

$$\bar{Q}(n+1) = \begin{bmatrix} Q_1(n) & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

and from (7), we have $Q(n+1) = H(n+1)G(n+1)\bar{Q}(n+1)$ and $Q(n+1) = H(n+1)Q_u(n+1)$, 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 and H are of the form

$$G(n+1) = \begin{bmatrix} Z(n+1) & h(n+1) & 0 \\ k^T(n+1) & \Pi_{i=1}^{p} c_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } H(n+1) = \begin{bmatrix} \tilde{Z}(n+1) & 0 & \tilde{h}(n+1) \\ 0 & 1 & 0 \\ \tilde{k}^T(n+1) & 0 & \Pi_{i=1}^{p} \tilde{c}_i \end{bmatrix} \tag{10}$$

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

$$Q(n+1) = \begin{bmatrix} \tilde{Z}(n+1)Z(n+1)Q_1(n) & \tilde{Z}(n+1)h(n+1) & \tilde{h}(n+1) \\ k^T(n+1)Q_1(n) & \Pi_{i=1}^{p} c_i & 0 \\ \tilde{k}^T(n+1)Z(n+1)Q_1(n) & \tilde{k}^T(n+1)h(n+1) & \Pi_{i=1}^{p} \tilde{c}_i \end{bmatrix}. \tag{11}$$

We can obtain the residual vector when the updating wavefront passes through the array and is given by

$$\hat{e}_u(n+1) = \begin{bmatrix} \bar{e}_u(n+1) \\ e_{u_1}(n+1) \\ e_{u_2}(n+1) \end{bmatrix} = \begin{bmatrix} -Q_1^T(n)k(n+1)v_1(n+1) \\ -\Pi_{i=1}^{p} c_i \cdot v_1(n+1) \\ -v_2(n+1) \end{bmatrix} \tag{12}$$

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 a residual.

From (8) and (11), 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 are given by

$$\begin{bmatrix} e_1(n+1) \\ e_2(n+1) \end{bmatrix} = \begin{bmatrix} -\Pi_{i=1}^{p} c_i \cdot v_1(n+1) - h^T(n+1)\tilde{k}(n+1)v_2(n+1) \\ -\Pi_{i=1}^{p} \tilde{c}_i \cdot v_2(n+1) \end{bmatrix} \tag{13}$$

where e_1 and e_2 are the residuals associated with updating and downdating, respectively. From (7), 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 $\Pi_{i=1}^{p}, c_i$ in (12) can be obtained. A multiplier cell is then used to obtain $e_{u_1}(n+1) = -\Pi_{i+1}^{p} c_i \cdot v_1(n+1)$ as in [3]. In fact, although the window size is l, the residual $e_{u_1}(n+1)$ is estimated from $[x_{n-l+1}, \cdots, x_n, x_{n+1}]^T$ and $[y_{n-l+1}, \cdots, y_n, y_{n+1}]^T$ of window size $l+1$ since downdating of x_{n-l+1} has not yet been performed. That is, $e_{u_1}(n+1) = x_{n+1}^T \hat{w}_{[n-l+1,n+1]} - y_{n+1}$, where $\hat{w}_{[n-l+1,n+1]}$ denotes the optimal coefficient vector estimated from data $[x_{n-l+1}, \cdots, x_n, x_{n+1}]^T$ and $[y_{n-l+1}, \cdots, y_n, y_{n+1}]^T$.

Also, if $\tilde{c}_i$'s are propagated down to the diagonal boundary cells and are cumulatively multiplied, when the downdating wavefront passes through the triarray, the downdating residual in (13) can be obtained easily. It is estimated from $[x_{n-l+2}, \cdots, x_n, x_{n+1}]^T$ of window size l. That is, $e_2(n+1) = x_{n-l+1}^T \hat{w}_{[n-l+2,n+1]} - y_{n-l+1}$. Obviously, the residual at 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, (13) provides a simple relation for this updating residual before and after the downdating. That is, $e_1(n+1) = e_{u_1}(n+1) - h^T(n+1)\tilde{k}(n+1)v_2(n+1)$. If downdating is performed first, then by the same argument as above, we can obtain $e_{d_2}(n+1) = -\Pi_{i=1}^{p} \tilde{c}_i \cdot v_2(n+1) = x_{n-l+1}^T \hat{w}_{[n-l+2,n]} - y_{n-l+1}$, and $e_1(n+1) = -\Pi_{i=1}^{p} c_i \cdot v_1(n+1) = x_{n+1}^T \hat{w}_{[n-l+2,n+1]} - y_{n+1}$. 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 a numerical stability problem [4]. A dual-state up/downdating systolic array for the recursive residual estimation is also shown in Fig. 1.

V. Conclusions

A dual-state systolic triarray performing up/down-dating operations for fixed-window RLS filtering has been proposed. Due to the inherent similarity between updating and downdating, they can use the same hardware and alternatively pipelined to achieve parallelism in this dual-state systolic triarray. A flip-flop systolic behavior of this array is observed both in temporal and spatial domains. Extracting the optimal residuals in real-time using the proposed up/downdating systolic array is also considered in this paper.

References

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

[2] W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic array," *Proc. SPIE*, vol. 298: pp. 19-26, 1981.

[3] J. G. McWhirter, "Recursive least-squares minimisation using a systolic array," *Proc. SPIE* vol. 431, pp. 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*, pp. 423-426, 1987.

[5] C. M. Rader and A. O. Steinhardt, "Hyperbolic Householder transformations," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 34, pp. 1589-1602, Dec. 1986.

[6] A. W. Bojanczyk, R. P. Brent, P. van Dooren, and F. R. De Hoog, "A note on downdating the Cholesky factorization," *SIAM J. Sci. Stat. Comput.*, vol. 8, pp. 210-221, May 1987.

[7] K. J. R. Liu, S. F. Hsieh, and K. Yao, "Systolic block Householder transformation for RLS algorithm with two-level pipelined implementation," *IEEE Trans. Signal Processing*, vol. 40, pp. 946-958, Apr. 1992.

[8] J. G. McWhirter, "A systolic array for recursive least squares minimisation and adaptive beamforming," in *Numerical Linear Algebra, Digital Signal Processing and Parallel Algorithms*. G. H. Golub and P. Van Dooren, Eds. New York: Springer Verlag, 1991.

High-Speed VLSI Architectures for Huffman and Viterbi Decoders

Keshab K. Parhi

***Abstract*—This paper presents pipelined and parallel architectures for high-speed implementation of Huffman and Viterbi decoders (both of which belong to the class of tree-based decoders). Huffman decoders are used for lossless compression. The Viterbi decoder is commonly used in communications systems. The achievable speed in these decoders is inherently limited due to their sequential nature of computation. This speed limitation is overcome using our previously proposed technique of *look-ahead computation*. The *incremental computation* technique is used to obtain efficient parallel (or block) implementations. The *decomposition* technique is exploited to reduce the hardware complexity in pipelined Viterbi decoders, but not in Huffman decoders. Logic minimization is used to reduce the hardware overhead complexity in pipelined Huffman decoders.**

I. Introduction

The difficulty of pipelining the feedback algorithms was removed by the use of look-ahead computation. Look-ahead can be used in the form of pipelining [1], parallel processing [2], or both. This paper considers design of high-speed architectures for two classes of tree-based decoders: the Huffman decoder [3], and the Viterbi decoder (which is based on dynamic programming calculations) [4], [5]. Huffman decoders are used for lossless compression of speech and image signals. Viterbi decoders are used in communications systems. Section II of this paper addresses the design of Huffman decoder architectures and Section III addresses Viterbi decoder architectures.

II. Architectures for Huffman Decoders

Huffman decoders [3] are used for lossless compression of speech and image signals. In this decoder, probability of occurrence of different symbols is assumed to be known. More probable symbols are assigned shorter code words; this leads to an overall reduction in the number of bits to be transmitted.

Manuscript received May 28, 1991; revised December 1, 1991 and April 5, 1992. This research was supported by the Army Research Office under contract number DAAL03-90-G-0063. This paper was recommended by Associate Editor K. H. Tzou.

The author is with the Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455.

IEEE Log Number 9201391.

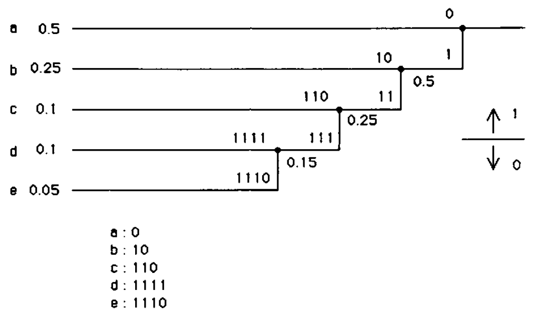

Fig. 1. A Huffman tree.

Although applicability of look-ahead in finite state machines was observed in [6] and [7], it has not been used to design high-speed Huffman decoders. The unequal code wordlength in Huffman decoders makes it difficult to apply look-ahead in a traditional way. However, by transforming the Huffman decoder to an equivalent finite state machine [8], we can apply look-ahead. In this section, based on [9], we present completely pipelined and parallel Huffman decoder implementations. For transmission using fewer symbols (say four to eight), the proposed look-ahead pipelining can be more attractive than the approaches in [8].

Consider an example of a Huffman decoder which needs to decode one of five symbols, denoted as a, b, c, d, and e. Let the probability of occurrence of these symbols be respectively 0.5, 0.25, 0.1, 0.1, and 0.05. The Huffman code for this symbol set is constructed using the Huffman tree shown in Fig. 1. The probabilities of the two lowest probability symbols are added in each step. At the end, we assign code words to symbols starting from the top of the tree such that the higher probability occurrence is assigned a code 1 and lower a 0 (ties are broken arbitrarily). For the example symbol set, the chosen code words are a:0, b:10, c:110, d:1111, and e:1110. The average code word length is 1.9 bits (a simple binary encoding would require three bits per symbol).

Our example Huffman decoder can be represented by a four-state finite state machine as shown in Fig. 2. At the end of each codeword, the finite state machine returns to the initial state S_1. The finite state machine processes one input bit, represented as $x(n)$, and has five outputs represented as $a(n)$, $b(n)$, $c(n)$, $d(n)$, and $e(n)$. Each output signal wire represents the presence or absence of the corresponding symbol in that cycle. For example, $c(n)$ is 1 if the decoder detects the symbol c in cycle n, and 0 otherwise. Note that this output representation is chosen for simplicity and more efficient output encoding can be used in practice.

The finite state machine can be represented by the state update representation

$$s(n+1) = s(n)T(n) \tag{1}$$

where $s(n) = [s_1(n)\ s_2(n)\ s_3(n)\ s_4(n)]$, and

$$T(n) = \begin{bmatrix} \overline{x(n)} & x(n) & 0 & 0 \\ \overline{x(n)} & 0 & x(n) & 0 \\ \overline{x(n)} & 0 & 0 & x(n) \\ 1 & 0 & 0 & 0 \end{bmatrix}$$