the time index \( n \geq l \) is assumed. The LS problem is to find a \( p \times 1 \) optimum coefficient vector \( \hat{w}(n) \in \mathbb{R}^p \), such that the Euclidean norm of its associated residual \( e(n) = X(n)w(n) - y(n) \) is minimized.

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 feasibility 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 \( \{X(n) : y(n)\} \) is available. Then

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

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

\[
R(n)\hat{w}(n) = u(n)
\]

We can obtain \( \{R(n+1) : u(n+1)\} \) by first updating

\[
\begin{bmatrix}
R(n) & u(n) \\
x_{n+1} & y_{n+1} \\
x_{n+1} & y_{n+1}
\end{bmatrix}
\]

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

\[
G_{p,p+1} \cdots G_{2,p+1} G_{1,p+1} [R(n) : u(n) : x_{n+1} : y_{n+1} ]
\]

\[
= \begin{bmatrix} \hat{R}(n+1) & \hat{u}(n+1) \\ 0 & \hat{v}(n+1) \end{bmatrix}
\]

\[
H_{p,p+2} \cdots H_{2,p+2} H_{1,p+2} \begin{bmatrix} \hat{R}(n+1) & \hat{u}(n+1) \\ 0 & \hat{v}(n+1) \end{bmatrix}
\]

\[
= \begin{bmatrix} R(n+1) & u(n+1) \\ 0 & v(n+1) \end{bmatrix}
\]

\[
= \begin{bmatrix} R(n+1) & u(n+1) \\ 0 & v(n+1) \end{bmatrix}
\]

\[
= \begin{bmatrix} R(n+1) & u(n+1) \\ 0 & v(n+1) \end{bmatrix}
\]
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.,

\[
G_{i,p+1} = \begin{bmatrix}
\cdots & a_i & \cdots \\
0 & c_i & 0 & s_i & 0 & \cdots \\
0 & 0 & 1 & 0 & 0 & \cdots \\
0 & -s_i & 0 & c_i & 0 & \cdots \\
0 & 0 & 0 & 0 & 1 & \cdots \\
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\sqrt{a_i^2 + a^2_{p+1}} \\
\cdots \\
\cdots \\
\cdots \\
0 \\
\end{bmatrix}
\]  

(5)

where \(c_i = a_i / \sqrt{a_i^2 + a^2_{p+1}}\) and \(s_i = a_{p+1} / \sqrt{a_i^2 + a^2_{p+1}}\).

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

\[
H_{i,p+2} = \begin{bmatrix}
\cdots & a_i & \cdots \\
0 & \xi_i & 0 & -\xi_i & 0 & \cdots \\
0 & 0 & 1 & 0 & 0 & \cdots \\
0 & -\xi_i & 0 & \xi_i & 0 & \cdots \\
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\sqrt{a_i^2 - a^2_{p+2}} \\
\cdots \\
\cdots \\
\cdots \\
0 \\
\end{bmatrix}
\]  

(6)

where \(\xi_i = a_i / \sqrt{a_i^2 - a^2_{p+2}}\) and \(\bar{\xi}_i = a_{p+2} / \sqrt{a_i^2 - a^2_{p+2}}\).

Let \(H(n + 1) = H_{p+2} \cdots H_{i,p+2}\) and \(G(n + 1) = G_{p+2} \cdots G_{i,p+1}\). By combining (3) (4), we have

\[
H(n + 1)G(n + 1) \begin{bmatrix}
R(n) \\
x_{n+1}^T \\
\end{bmatrix} = \begin{bmatrix}
u(n + 1) \\
0 \\
0 \\
\end{bmatrix}.
\]  

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 reprocessed (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 \(\bullet\) to denote a processor working on a Givens rotation (updating) and a white circle \(\circ\) to denote a hyperbolic rotation (downdating). We also note that only one control bit is required in determining whether updating or downdating operation needs to be performed.

To this dual-state systolic triarray, data rows are skewed with updating and downdating data interleaved to form a sequence of up/down-dating wavefronts which will then 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 also be 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 $n$th) element of $\hat{e}(n)$. In this section, we consider an efficient implementation to obtain the newest residuals under up/downdating operations.

From (1), we can separate $Q(n)$ into two terms as $Q(n) = \{Q_1^2(n), Q_2^2(n)\}^T$, where $Q_1(n) \in R^{p \times p}$, $Q_2(n) \in R^{(p-1) \times p}$. We can also rewrite the optimal residual vector as

$$\hat{e}(n + 1) = -Q_1^2(n + 1) \begin{bmatrix} v_1(n + 1) \\ v_2(n + 1) \end{bmatrix}.$$  

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

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

and from (7), we have $Q(n + 1) = H(n + 1)G(n + 1)Q(n + 1)$ and $Q(n + 1) = H(n + 1)Q_1(n + 1)$, where $Q_2(n + 1) = G(n + 1)Q_2(n + 1)$ is defined as the $Q$ matrix associated with updating only. It can be shown that $G$ and $H$ are of the form

$$G(n + 1) = \begin{bmatrix} Z(n + 1) & h(n + 1) \\ k^T(n + 1) & \Pi_{n+1}^c c_l \end{bmatrix} \text{ and } H(n + 1) = \begin{bmatrix} k^T(n + 1) \\ \Pi_{n+1}^c c_l \end{bmatrix}$$

where

$$Z(n + 1) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad h(n + 1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad k(n + 1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and

$$Q(n + 1) = \begin{bmatrix} Z(n + 1) & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

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

$$Q(n + 1) = \begin{bmatrix} Z(n + 1) & h(n + 1) \\ k^T(n + 1) & \Pi_{n+1}^c c_l \end{bmatrix} \text{ and } H(n + 1) = \begin{bmatrix} k^T(n + 1) \\ \Pi_{n+1}^c c_l \end{bmatrix}$$

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

$$\hat{e}_n(n + 1) = \begin{bmatrix} \tilde{e}_n(n + 1) \\ e_1(n + 1) \\ e_2(n + 1) \end{bmatrix} = \begin{bmatrix} -Q_1^2(n)k(n + 1)v_1(n + 1) \\ -\Pi_{n+1}^c c_l \cdot v_2(n + 1) \\ -v_2(n + 1) \end{bmatrix} = \begin{bmatrix} -\Pi_{n+1}^c c_l \cdot v_1(n + 1) - k^T(n + 1) \tilde{e}_n(n + 1) \\ -\Pi_{n+1}^c c_l \cdot v_1(n + 1) \end{bmatrix}$$

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 $c_1$'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 $\Pi_{n+1}^c c_l$ in (12) can be obtained. A multiplier cell is then used to obtain $e_1(n + 1) = -\Pi_{n+1}^c c_l \cdot v_1(n + 1)$ as in [3]. In fact, although the window size is $I$, the residual $e_1(n + 1)$ is estimated from $\{x_{n-I+1}, \cdots, x_n, x_{n+1}\}^T$ and $\{y_{n-I+1}, \cdots, y_n, y_{n+1}\}^T$ of window size $I + 1$ since downdating of $x_{n-I+1}$ has not yet been performed. That is, $e_1(n + 1) = x_{n+1}^T \tilde{w}_{[n-I+1,n+1]} - y_{n+1}$, where $\tilde{w}_{[n-I+1,n+1]}$ denotes the optimal coefficient vector estimated from data $\{x_{n-I+1}, \cdots, x_n, x_{n+1}\}^T$ and $\{y_{n-I+1}, \cdots, y_n, y_{n+1}\}^T$.

Also, if $c_l$'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 $\{x_{n-I+1}, \cdots, x_n, x_{n+1}\}^T$ of window size $I$. That is, $e_2(n + 1) = x_{n-I+1}^T \tilde{w}_{[n-I+2,n+1]} - y_{n-I+1}$. Obviously, the residual at time $n - I + 1$ is past estimated by data from $\{x_{n-I+1}, \cdots, x_n, x_{n+1}\}$ 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 $h^T(n + 1)k(n + 1)v_1(n + 1)$ 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, $e_1(n + 1) = e_1(n + 1) - h^T(n + 1)k(n + 1)v_1(n + 1)$. If downdating is performed first, then by the same argument as above, we can obtain $e_1(n + 1) = e_1(n + 1) - \Pi_{n+1}^c c_l \cdot v_2(n + 1) = x_{n-I+1}^T \tilde{w}_{[n-I+2,n+1]} - y_{n-I+1}$, and $e_1(n + 1) = e_1(n + 1) - \Pi_{n+1}^c c_l \cdot v_2(n + 1) = x_{n-I+1}^T \tilde{w}_{[n-I+1,n+1]} - y_{n+1}$.

V. CONCLUSIONS

A dual-state systolic triarray performing up/downdating 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

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 techniques are 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.

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 S0. 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

\[ s(n+1) = s(n) T(n) \]  

where

\[ s(n) = [s_a(n), s_b(n), s_c(n), s_d(n), s_e(n)] \]

and

\[
T(n) = \begin{bmatrix}
x(n) & x(n) & 0 & 0 \\
x(n) & 0 & x(n) & 0 \\
x(n) & 0 & 0 & x(n) \\
1 & 0 & 0 & 0
\end{bmatrix}
\]