Chapter 10
On Pipelined Implementations of QRD-RLS Adaptive Filters

Jun Ma and Keshab K. Parhi

Abstract This chapter discusses the pipelined systolic implementations of QR-decomposition-based recursive least-squares (QRD-RLS) adaptive filters. The *annihilation-reording look-ahead* technique is presented as an attractive technique for pipelining of Givens rotation (or CO-ordinate Rotation DIgital Computer (CORDIC)) based adaptive filters. It is an *exact* look-ahead and is based on CORDIC arithmetic, which is known to be numerically stable. The conventional look-ahead is based on multiply–add arithmetic. The annihilation-reording look-ahead technique transforms an orthogonal *sequential* adaptive filtering algorithm into an equivalent orthogonal *concurrent* one by creating additional concurrency in the algorithm. Parallelism in the transformed algorithm is explored, and different implementation styles including *pipelining, block processing*, and *incremental block processing* are presented. Their complexity are also studied and compared. The annihilation-reording look-ahead is employed to develop *fine-grain* pipelined QRD-RLS adaptive filters. Both *implicit* and *explicit* weight extraction algorithms are considered. The proposed pipelined architectures can be operated at arbitrarily high sample rate without degrading the filter convergence behavior. Stability under finite-precision arithmetic are studied and proved for the proposed architectures. The complexity of the pipelined architectures are analyzed and compared.

Jun Ma
Shanghai Jiaotong University, Shanghai – China
e-mail: majun@ic.sjtu.edu.cn

Keshab K. Parhi
University of Minnesota, Minneapolis, MN – USA
e-mail: parhi@umn.edu

J.A. Apolinário Jr. (ed.), *QRD-RLS Adaptive Filtering*,
DOI 10.1007/978-0-387-09734-3_10,

10.1 QRD-RLS Systolic Architecture

Recursive least squares (RLS) [1] based adaptive filters have wide applications in channel equalization, voiceband modem, high-definition TV (HDTV), digital audio broadcast (DAB) system, beamforming, and speech and image processing. Historically, least mean squares (LMS) based adaptive filters are preferred in practical applications due to their simplicity and ease of implementation. A limitation of LMS algorithm is that it has a very slow convergence rate. The convergence of the LMS algorithm is also very sensitive to the eigenvalue spread of the correlation matrix of the input data. In applications such as HDTV equalizer and DAB system, there are only limited number of data samples available. LMS-based equalizer may not be able to reach convergence. The convergence of the RLS algorithm is an order of magnitude faster than that of the LMS algorithm, but its complexity is an order of magnitude higher. However, with rapid advances in scaled very large scale integration (VLSI) technologies, it is possible to implement RLS adaptive filters on single chips which will make them attractive due to their rapid convergence behavior.

QR decomposition-based RLS (QRD-RLS) algorithm [1], also referred as Givens rotation or COordinate Rotation DIgital Computer (CORDIC)-based RLS algorithm in this chapter, is the most promising RLS algorithm since it possess desirable properties for VLSI implementations such as regularity, good finite word-length behavior, and can be mapped onto CORDIC arithmetic-based processors [2–5]. The QRD-RLS algorithm can be summarized as follows. The notations used in this chapter are slightly different from those used in previous chapters, e.g. Chapters 2–4. For ease of reading, their correspondences are summarized in Table 10.1. At each sample time instance n, evaluate a residual (*a posteriori*) error:

$$e(n) = y(n) - \mathbf{u}^{\mathrm{T}}(n)\,\mathbf{w}(n), \tag{10.1}$$

where $\mathbf{u}(n)$ and $y(n)$ denote the p-element vector of signal samples and the reference signal at time instance n, respectively, and $\mathbf{w}(n)$ is the p-element vector of weights which minimize the quantity

$$\begin{aligned} \xi(n) &= \| \Lambda^{1/2}(n)\mathbf{e}(n) \|^2 \\ &= \| \Lambda^{1/2}(n)\,(\mathbf{y}(n) - A(n)\mathbf{w}(n)) \|^2, \end{aligned} \tag{10.2}$$

Table 10.1 Notation correspondences between Chapter 10 and Chapters 2–4.

Notations	Chapter 10	Chapters 2–4
Time index	n	k
Input signal	$u(n)$	$x(k)$
Input vector	$\mathbf{u}(n)$	$\mathbf{x}(k)$
Input matrix	$A(n)$	$\mathbf{X}(k)$
Reference signal	$y(n)$	$d(k)$
Cholesky factor	$R(n)$	$\mathbf{U}(k)$
Cholesky factor degree	$p \times p$	$(N+1) \times (N+1)$

where $\mathbf{y}(n) = [y(1), \ldots, y(n)]^{\mathrm{T}}$ denotes the sequence of all reference signal samples obtained up to time instance n, $A(n) = [\mathbf{u}(1), \mathbf{u}(2), \cdots, \mathbf{u}(n)]^{\mathrm{T}}$ is the input data matrix, and $\Lambda(n) = \mathrm{diag}[\lambda^{n-1}, \cdots, \lambda, 1]$ is the diagonal matrix of the forgetting factors. Here, we assume that all the data are real. The extension to the complex case does not seem to have any particular difficulties. The optimum weight vector $\mathbf{w}_{ls}$ of the QRD-RLS solution can be obtained by solving the following equation:

$$R(n)\,\mathbf{w}_{ls}(n) = \mathbf{p}(n), \tag{10.3}$$

where $R(n)$ and $\mathbf{p}(n)$ are p-by-p matrix and p-by-1 vector, respectively, which are obtained by applying a QR decomposition to the weighted data matrix $\Lambda^{1/2}(n)A(n)$ and the weighted reference vector $\Lambda^{1/2}(n)\mathbf{y}(n)$, respectively. $R(n)$, which is usually referred to, in the literature [17], as the *Cholesky factor*, is chosen, in this chapter, to be an upper triangular matrix.

In practice, the QR decomposition is implemented in a recursive manner. With each incoming data sample set, a new row $\mathbf{u}^{\mathrm{T}}(n)$ is appended to the data matrix $A(n-1)$ to yield $A(n)$. An orthogonal transformation matrix $Q(n)$ is determined as products of p Givens rotation matrices to null the last row of $A(n)$. Thus the triangular matrix $R(n-1)$ gets updated to $R(n)$. The determined matrix $Q(n)$ is then used to update $\mathbf{p}(n-1)$ to $\mathbf{p}(n)$. The QR update procedure can be described by the following equation:

$$\begin{bmatrix} R(n) & \mathbf{p}(n) \\ \mathbf{0}_p^{\mathrm{T}} & \alpha(n) \end{bmatrix} = Q(n) \begin{bmatrix} \lambda^{1/2}R(n-1) & \lambda^{1/2}\mathbf{p}(n-1) \\ \mathbf{u}^{\mathrm{T}}(n) & y(n) \end{bmatrix}. \tag{10.4}$$

A systolic array-based signal flow graph (SFG) representation of the QR update procedure is shown in Figure 10.1. In this figure, cells with symbols r and p inside are the elements of the upper triangular matrix R and the vector $\mathbf{p}$, respectively, as shown in Equation (10.4). The recursive update relationship from time index $n-1$ to n, shown in Equation (10.4), is reflected by the delay element denoted by the small square cell with symbol D inside in Figure 10.1. The input data $u_i(n)$ at the top row of Figure 10.1 are the elements of input vector $\mathbf{u}(n)$ in Equation (10.4), and the reference data $y(n)$ in Figure 10.1 corresponds to the reference data $y(n)$ in Equation (10.4). In Figure 10.1, the circle and square cells denote Givens rotations or CORDIC operations with circle cells operating in *vectoring mode* and square cells operating in *rotating mode*. The functionality of the recursive update from time index $n-1$ to n is shown at the bottom of Figure 10.1. The c and s denote $\cos\theta$ and $\sin\theta$, respectively, which are chosen to annihilate $x_1(n)$. The determined rotation angle is then applied to rotate the second column vector which consists of $\lambda^{M/2}r_2(n-1)$ and $x_2(n)$, where $M = 1$ in the case of Equation (10.4). The forgetting factor λ in Equation (10.4), for clarity purpose, is omitted in the circle and square cell in Figure 10.1.

In practice, there are two types of QRD-RLS algorithms. One is *implicit* weight extraction-based RLS algorithms, which are found useful in applications such as adaptive beamforming. In these algorithms, the residual error $e(n)$ is obtained without the explicit computation of the weight vector $\mathbf{w}(n)$. A popular implicit weight

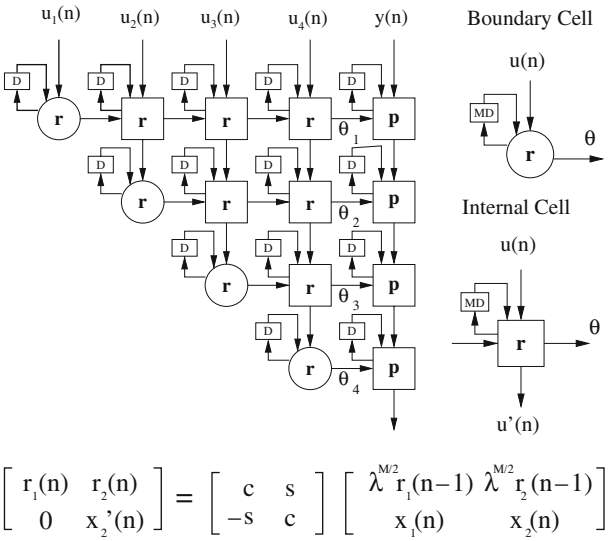


Fig. 10.1 The systolic array-based signal flow graph representation of the QR update procedure.

extraction algorithm is due to McWhirter etc. [6]. The other is *explicit* weight extraction-based RLS algorithms, which are found useful in applications such as channel equalization. One such algorithm is due to Gentleman and Kung [7], which involves a triangular update part and a linear array for triangular back-solving. The linear array part does not make use of Givens rotations, and thus cannot be efficiently combined with the triangular update part. To overcome this problem, alternative QRD-RLS algorithms with *inverse updating* have been developed to achieve explicit weight extraction and also make use of Givens rotations. A typical structure is the double-triangular type adaptive inverse QR algorithm [8, 9]. This algorithm performs a QR update in an upper triangular matrix and an inverse QR update for weight extraction in a lower triangular matrix.

One of the important ways to design efficient RLS algorithms for high-speed/low-power applications is through *pipelining* [10, 11] and *parallel processing* [12]. Both implicit and explicit weight extraction-based QRD-RLS algorithms can be easily pipelined at cell level (also referred as *coarse-grain pipelining*). However, the speed or sample rate of the algorithms is limited by the recursive operations in individual cells as shown in Figure 10.1. In many applications, such as image coding and beamforming, very high data rates would be required, and the sequential QRD-RLS algorithms may not be able to operate at such high data rate. In this chapter, we exploit the parallelism that exists in the QRD-RLS algorithm and consider pipelining at finer level such as bit or multi-bit level (also referred as *fine-grain pipelining*). Notice that apart from being used to increase speed, pipelining can also be used to reduce power dissipation in low to moderate speed applications [13].

To exploit the parallelism and increase the speed of the QRD-RLS, look-ahead techniques [14] or block processing techniques can be applied. The look-ahead

techniques and the so called STAR rotation have been used in [15] to allow fine-grain pipelining with little hardware overhead. However, this is achieved at the cost of degradation of filtering performance due to the approximations in the algorithms. Block processing was used to speed up the QRD-RLS in [16], however with large hardware overhead. Both algorithms are based on multiply–add arithmetic. If one insists on not using multiply–add arithmetic for their implementation, there is no trivial extension of the look-ahead technique to the QRD-RLS algorithm.

There are other fast QRD-RLS algorithms, which are computationally more efficient than the original algorithm [17, 18]. Square-root free forms of QRD-RLS are presented in [17–22]. A unified approach to square-root QRD-RLS algorithm is presented in [19]. A low-complexity square-root free algorithm is developed in [20]. In [21], a scaled version of the fast Givens rotation [17] is developed that prevents overflow and underflow. A division as well as square-root free algorithm has been proposed in [23]. In [18], a fast QRD-RLS algorithm based on Givens rotations was introduced. However, all these fast algorithms suffer the same pipelining difficulty as the QRD-RLS algorithm, i.e., they cannot be pipelined at fine-grain level.

In this chapter, the *annihilation-reording look-ahead* technique [24] is presented to achieve fine-grain pipelining in QRD-RLS adaptive filters. It is an exact look-ahead and based on CORDIC arithmetic. One of the nice properties of this technique is that it can transform an *orthogonal sequential* recursive DSP algorithm to an equivalent *orthogonal concurrent* one by creating additional concurrency in the algorithm. The resulting transformed algorithm possesses pipelinability, stability (if the original one is stable), and good finite word-length behavior which are attractive for VLSI implementations.

The rest of the chapter are organized as follows. The annihilation-reording look-ahead technique is presented in Section 10.2. The derivation of pipelined CORDIC-based RLS adaptive filters using the proposed look-ahead technique is presented in Section 10.3. Section 10.4 draws conclusions and briefly discusses the adaptive beamforming application. Appendix provides the derivation and proof of some key formulas presented in the chapter.

10.2 The Annihilation-Reording Look-Ahead Technique

In this section, we introduce the annihilate-reording look-ahead technique as an exact look-ahead based on CORDIC arithmetic. Similar to the traditional look-ahead, it transforms a sequential recursive algorithm to an equivalent concurrent one by creating additional parallelism in the algorithm. It is based on CORDIC arithmetic and is suitable for pipelining of Givens rotation-based adaptive filtering algorithms. The annihilation-reording look-ahead technique can be derived from two aspects. One is from the *block processing* point of view, the other is from the *iteration* point of view. The former is practical in real applications, while the latter shows the connection with the traditional look-ahead technique. During our derivation, the forgetting factor λ is omitted for clarity purpose.

This section is organized as follows. The derivations of the annihilation-reording look-ahead through block processing and iteration are presented in Subsection 10.2.1 and Subsection 10.2.2, respectively. The relationship with the conventional multiply–add look-ahead is shown in Subsection 10.2.3. Subsection 10.2.4 explores the parallelism in the proposed look-ahead transformed algorithm. Different implementation styles are then presented in Subsection 10.2.5. Finally, a lemma for stability invariance is stated and proved in Subsection 10.2.6.

10.2.1 Look-ahead through block processing

In this subsection, we derive the annihilation-reording look-ahead transformation for Givens rotation-based algorithms via block-processing formulation.

The annihilation-reording look-ahead technique can be summarized as the following two-step procedure.

1. Formulate block updating form of the recursive operations with block size equal to the pipelining level M.
2. Choose a sequence of Givens rotations to perform the updating in such a way that it first operates on the block data and then updates the recursive variables. The aim is to reduce the computational complexity of a block update inside the feedback loop to the same complexity as a single-step update.

Assume that a three-time speed up is desired for the QR update shown in Figure 10.1. Consider the block update form of the QR update procedure shown in Figure 10.2 with block size equal to the desired pipelining level 3. A sequence

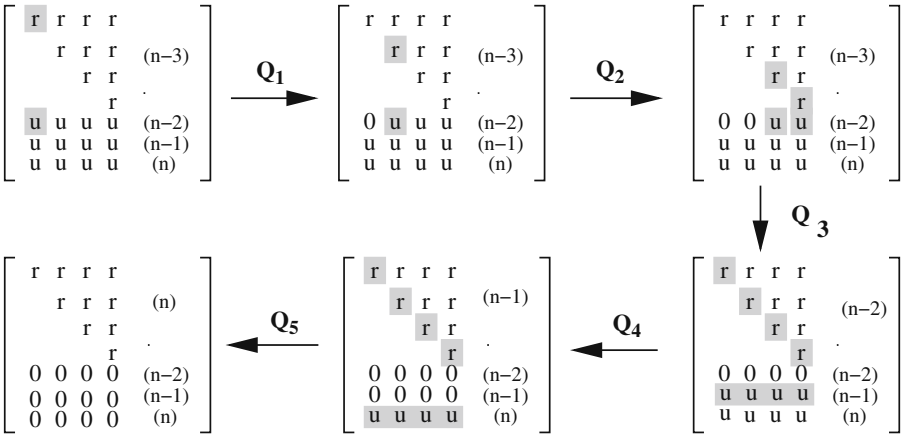


Fig. 10.2 QRD-RLS block update.

of Givens rotations is then chosen to annihilate the block data $\mathbf{u}(n-2), \mathbf{u}(n-1)$, and $\mathbf{u}(n)$, and update $R(n-3)$ to $R(n)$. In this figure, traditional sequential update operation is used. The sample data is annihilated in a row-by-row manner and the diagonal r elements are involved in each update. The SFG of a typical r element update is shown in Figure 10.3. It can be seen that the number of rotations inside the feedback loop increases linearly with the number of delay elements in the loop. Therefore, there is no net improvement in the sample or clock speed.

The annihilation-reording look-ahead technique is illustrated in Figure 10.4. In this figure, the sample data is annihilated in a column-by-column manner and the diagonal r elements are updated only at the last step. The SFG of a typical r element update is shown in Figure 10.5. It can been seen that, without increasing the loop computational complexity, the number of delay elements in the feedback loop is increased from one delay element to three delay elements. These three delay elements can then be redistributed around the loop using the *retiming* technique [25] to achieve fine-grain pipelining by three-level. The two CORDIC units outside the

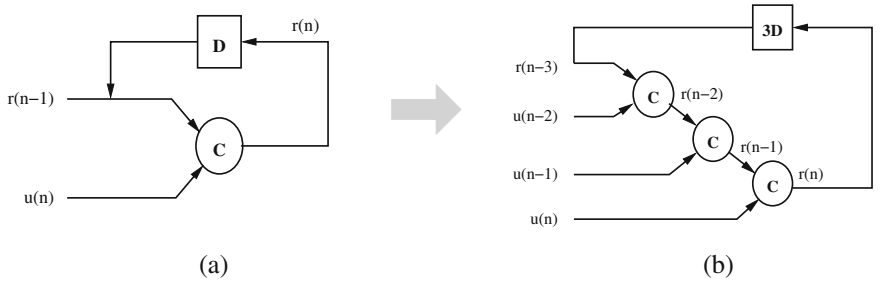


Fig. 10.3 (**a**) The sequential QR update procedure. (**b**) The block update procedure with block size 3.

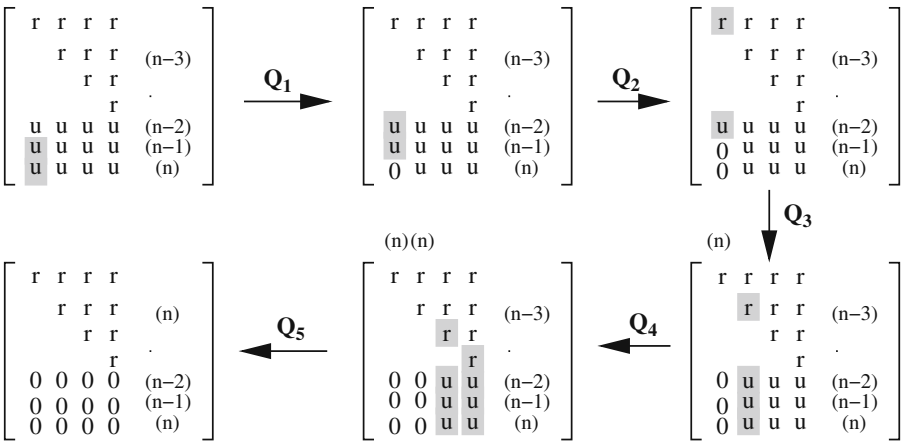


Fig. 10.4 QRD-RLS annihilation-reordering look-ahead.

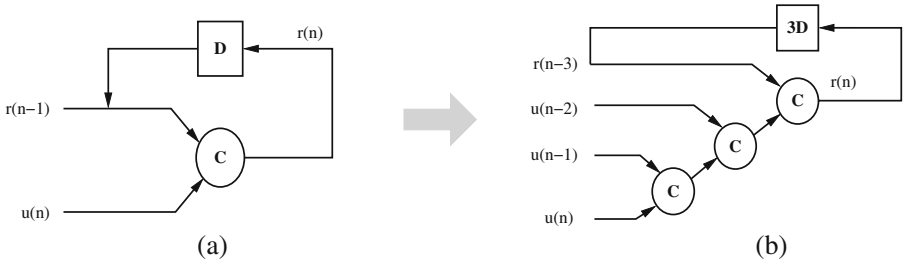


Fig. 10.5 (**a**) A sequential QR update procedure. (**b**) The three-level pipelined architecture using annihilation-reording look-ahead.

feedback loop are the computation overhead due to the look-ahead transformation. Since they are feed-forward, *cutset pipelining* [26] can be applied to speed them up. Furthermore, the overhead CORDIC units outside the loop can be arranged in a *tree* structure to explore the parallelism and reduce overall latency.

10.2.2 Look-ahead through iteration

Alternatively, the annihilation-reording look-ahead can be derived through matrix iterations. From Figure 10.1 and Equation (10.4), the basic QR recursion is given as follows:

$$\begin{bmatrix} r(n) \\ 0 \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} r(n-1) \\ u(n) \end{bmatrix}, \tag{10.5}$$

where $r(n)$ and $u(n)$ correspond to the boundary element and input data to the boundary element in Figure 10.1, respectively. A direct look-ahead by iterating Equation (10.5) two times can be performed by the following embedding procedure. Equation (10.5) can be rewritten as

$$\begin{bmatrix} r(n) \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} c_1 & 0 & s_1 \\ 0 & 1 & 0 \\ -s_1 & 0 & c_1 \end{bmatrix} \begin{bmatrix} r(n-1) \\ 0 \\ u(n) \end{bmatrix}. \tag{10.6}$$

From Equation (10.5), we also have

$$\begin{bmatrix} r(n-1) \\ 0 \\ u(n) \end{bmatrix} = \begin{bmatrix} c_2 & s_2 & 0 \\ -s_2 & c_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r(n-2) \\ u(n-1) \\ u(n) \end{bmatrix}. \tag{10.7}$$

Substituting Equation (10.7) into Equation (10.6) leads to

$$\begin{bmatrix} r(n) \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} c_1 & 0 & s_1 \\ 0 & 1 & 0 \\ -s_1 & 0 & c_1 \end{bmatrix} \begin{bmatrix} c_2 & s_2 & 0 \\ -s_2 & c_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r(n-2) \\ u(n-1) \\ u(n) \end{bmatrix}. \tag{10.8}$$

This is the one-step iterated version of Equation (10.5). Iterating (10.8) once more leads to the following two-step iterated version of (10.5).

$$\begin{bmatrix} r(n) \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} c_1 & 0 & 0 & s_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s_1 & 0 & 0 & c_1 \end{bmatrix} \begin{bmatrix} c_2 & 0 & s_2 & 0 \\ 0 & 1 & 0 & 0 \\ -s_2 & 0 & c_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_3 & s_3 & 0 & 0 \\ -s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r(n-3) \\ u(n-2) \\ u(n-1) \\ u(n) \end{bmatrix} \quad (10.9)$$

The SFG of (10.9) is shown in Figure 10.3. Notice that this transformation does not help much, since all three CORDIC operations involve updating the r element. Although the feedback loop contains three delays, the computation time in the loop is also increased by a factor of three. Therefore, the overall sample rate remains unaltered.

In order to increase the sample rates, the following transformation is considered. Notice that, in (10.9), instead of vectoring the input vector in the order of $(1,2),(1,3)$, and $(1,4)$, we could apply the Givens matrix in a different order so that the input vector is annihilated in the order of $(3,4),(2,3)$, and $(1,2)$, where notation (i,j) represents a pair of row indexes of input matrix in (10.9). For example, $(3,4)$ denotes that the Givens matrix will operate on input vector $[u(n-1),u(n)]^{\mathrm{T}}$, According to this scheme, the input samples are pre-processed and the r elements are updated only at the last step. This leads to the following three-level annihilation-reording look-ahead transformation for CORDIC-based RLS adaptive filters.

$$\begin{bmatrix} r(n) \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} c_1' & s_1' & 0 & 0 \\ -s_1' & c_1' & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_2' & s_2' & 0 \\ 0 & -s_2' & c_2' & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & c_1' & s_1' \\ 0 & 0 & -s_1' & c_1' \end{bmatrix} \begin{bmatrix} r(n-3) \\ u(n-2) \\ u(n-1) \\ u(n) \end{bmatrix}$$

The SFG of the above transformation is shown in Figure 10.5, which is the same as the one derived from the block processing point of view. Therefore, without increasing the loop computation time, we increase the number of delays in the feedback loop from one delay element to three delay elements. These three delay elements can then be redistributed around the loop to achieve pipelining by three-level.

The above derivation is similar to the traditional multiply–add look-ahead [11] procedure in the sense that both perform look-ahead through iteration. However, it can be seen that the block processing derivation in Section 10.2.1 is more simple and practical in real applications.

10.2.3 Relationship with multiply–add look-ahead

It is worth mentioning here, for the first order case, that there exists strong similarity of the transformed flow graphs between the annihilation-reording look-ahead and

the multiply–add look-ahead. Consider the first order IIR digital filter described by the following equation:

$$y(n) = a\,y(n-1) + u(n). \tag{10.10}$$

The SFG of (10.10) is shown in Figure 10.6(a). After applying the multiply–add look-ahead transformation with pipelining level 3, the resulting equation is given as

$$y(n) = a^3\,y(n-3) + a^2\,u(n-2) + a\,u(n-1) + u(n). \tag{10.11}$$

The SFG of (10.11) is shown in Figure 10.6(b). The filter sample rate can be increased by a factor of 3 after redistributing the three delay elements in the feedback loop using the *retiming* technique [25].

On the other hand, a redraw of Figure 10.5 can be carried out as in Figure 10.7. Comparing Figures 10.7 to Figure 10.6, it is seen that the two graphs are essentially the same except that the *multiply–add* units are replaced by the *CORDIC* units. Thus, both the annihilation-reording look-ahead and the multiply–add look-ahead explore the intra-iteration constraints in the recursive algorithms and create additional concurrency. The differences between the two techniques lie in that the multiply–add look-ahead is suitable for multiply–add arithmetic-based recursive digital filters where the filter coefficients are fixed. The annihilation-reording look-ahead is suitable for CORDIC arithmetic (or Givens rotation) based adaptive digital filters, where the filter coefficients are adaptive to the input data.

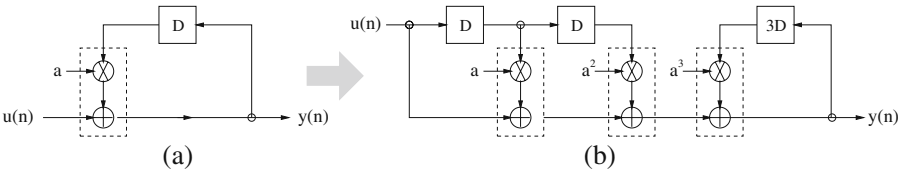


Fig. 10.6 (a) A first-order IIR digital filter. (b) The three-level pipelined architecture using multiply–add look-ahead.

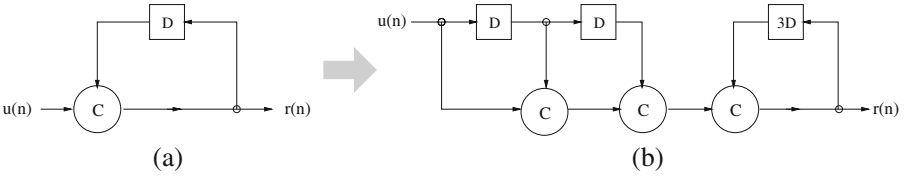


Fig. 10.7 (a) A sequential QR update procedure. (b) The three-level pipelined architecture using annihilation-reording look-ahead.

10.2.4 Parallelism in annihilation-reording look-ahead

In this subsection, we show explicitly how the annihilation-reording look-ahead technique explore the parallelism in the recursive algorithm and create the additional concurrency.

Consider the sequential QR update procedure shown in Figure 10.5(a). Its dependence graph (DG) representation is shown in Figure 10.8. In this figure, the little circle denotes CORDIC operations. The arrows denote dependency between signals. The k direction is the time increasing direction. The arrows along the k direction represent the dependency between iterations. For example, $r(n+1)$ is dependent on $r(n)$, $r(n)$ is dependent on $r(n-1)$, and so on. As we mentioned in Section 10.1, it is this kind of dependency that limits the speed of the QR update and thus limits the sample rate. The annihilation-reording look-ahead actually breaks this dependency and transforms the original DG into an equivalent DG which consists of M *independent* sub-DGs, where M is the desired pipelining level. Since these M sub-DGs are independent, they can be executed in parallel. Therefore, the M independent sub-DGs are the additional concurrency created by look-ahead transformation. For $M = 3$, the three sub-DGs: *DG-I, DG-II*, and *DG-III* are shown in Figure 10.9. Figure 10.9 is the DG representation of Figure 10.5(b). From Figure 10.9, it is seen that the computation of $r(n)$ is *not* dependent on the $r(n-1)$ anymore, instead dependent on the r element three iterations back in time which is $r(n-3)$. Similarly, $r(n+1)$ is dependent on $r(n-2)$, and $r(n+2)$ depends on $r(n-1)$. Therefore, after look-ahead transformation, the dependency between consecutive iterations are broken down into three independent operation sequences with each sequence having a dependency between every three iterations. For each sub-DGs, the dependency which is perpendicular to the k direction does not cause problem, since *index transformation* [27] (which is equivalent to the *cut-set* pipelining for SFG) can be performed to reveal these dependency. Therefore, the three independent sequences, which consist of $3^2 = 9$ independent CORDIC operations in total, are created for one iteration. In general, for pipelining level M, M^2 independent CORDIC operations are created in the algorithm for one iteration. As M increases, infinite parallelism can be created in the algorithm, thus can achieve arbitrarily high sample rate.

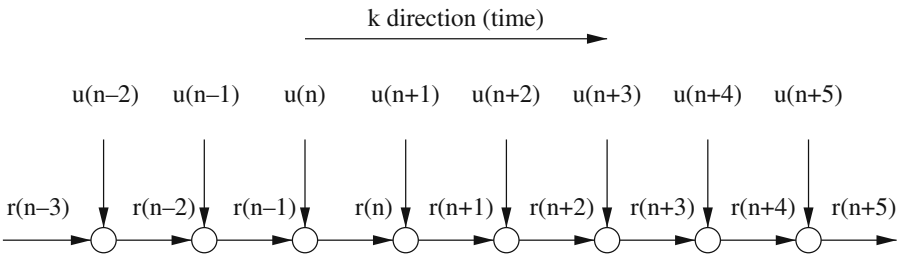


Fig. 10.8 The dependence graph of the sequential QR update procedure.

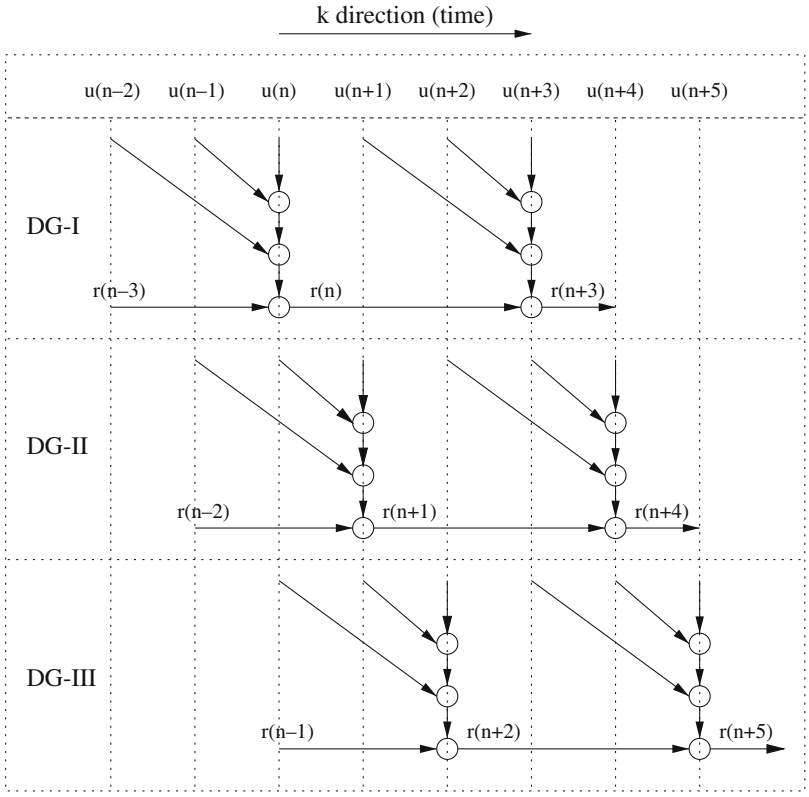


Fig. 10.9 The dependence graph of the pipelined QR update with pipelining level 3.

10.2.5 Pipelined and block processing implementations

In this subsection, we present three concurrent realizations of CORDIC-based RLS adaptive filters. They are *pipelining*, *block processing*, and *incremental block processing*.

10.2.5.1 Pipelined realization

Consider the three sub-DGs in Figure 10.9. If they are mapped along the k direction, we obtain the SFG representation. Since the three sub-DGs are independent, they can be mapped onto the same CORDIC operation resources and operated in a pipeline interleaving fashion [11]. This leads to the pipelined realization of CORDIC-based adaptive filters shown in Figure 10.10. In this figure, the input data samples are processed in the block manner through a tapped delay line as shown in Figure 10.11(a). Since consecutive block samples are shift-overlapped, thus all

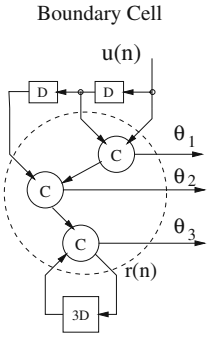


Fig. 10.10 Pipelined realization with pipelining level 3.

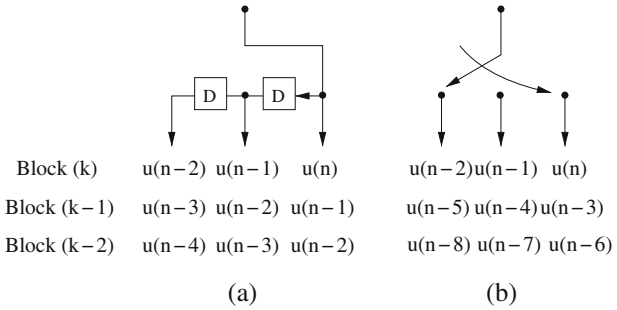


Fig. 10.11 Serial-to-parallel conversion for (**a**) Pipelining and (**b**) Block Processing.

filtering output can be obtained consecutively. The implementation complexity in terms of CORDIC units for pipelined realization is linear with respect to the pipelining level M which is $M = 3$ CORDIC units in Figure 10.10.

10.2.5.2 Block processing realization

In block processing, the three sub-DGs are mapped independently along the k direction to obtain the block processing realization shown in Figure 10.12. In block realizations, input samples are processed in the form of non-overlapping blocks to generate non-overlapping output samples. The block of multiple inputs is derived from the single serial input by using a serial-to-parallel converter at the input as shown in Figure 10.11(b), and the serial output is derived from the block of outputs by a parallel-to-serial converter at the output. The implementation complexity in terms of CORDIC units for block processing realization is quadratic with respect to the pipelining level M which is $M^2 = 3^2 = 9$ CORDIC units in Figure 10.12. To reduce complexity, incremental block processing technique can be used.

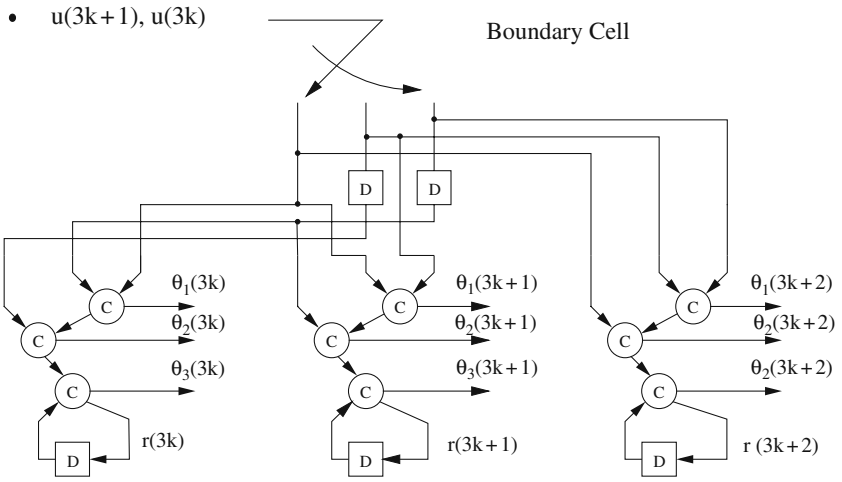


Fig. 10.12 Block processing realization with block size 3.

10.2.5.3 Incremental block processing realization

Consider the annihilation-reording look-ahead transformed DG shown in Figure 10.9. Instead of using $u(n+1), u(n), u(n-1)$, and $r(n-2)$ to obtain $r(n+1)$, $r(n+1)$ can be computed incrementally using $u(n+1)$ and $r(n)$ once $r(n)$ is available. Similarly, $r(n+2)$ can be computed incrementally using $u(n+2)$ and $r(n+1)$ once $r(n+1)$ is available. The DG of incremental block QR update is shown in Figure 10.13. Mapping the DG along the k direction gives us the SFG representation of the incremental block processing realization shown in Figure 10.14. The implementation complexity in terms of CORDIC units for incremental block processing is linear with respect to the pipelining level M which is $2M-1 = 2\times 3-1 = 5$ CORDIC units in Figure 10.14. Notice that the incremental computation parts do not contain feedback loops, thus cutset pipelining can be employed to speed them up.

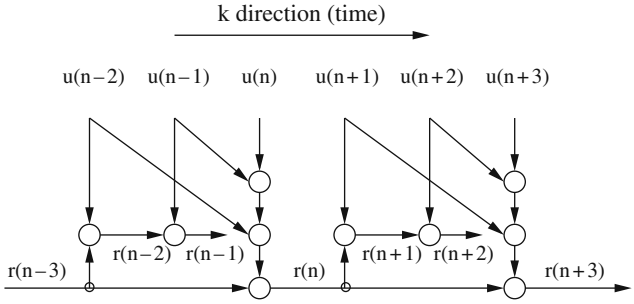


Fig. 10.13 The dependence graph of the incremental block QR update with block size 3.

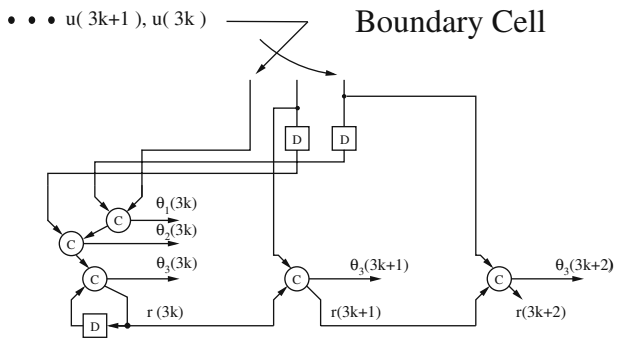


Fig. 10.14 Incremental block processing realization with block size 3.

Therefore, in terms of number of CORDIC units used in the implementation, pipelined realization is better than incremental block processing and block processing, and incremental block processing is better than block processing. In practice, the chosen of implementation styles depends on the target applications and available hardware resources.

10.2.6 Invariance of bounded input and bounded output

In this subsection, we show a property of the annihilation-reording look-ahead transformation. It will be useful in the proof of the stability of the pipelined QRD-RLS algorithm in Section 10.3.2.

Lemma 1. *Consider the compound CORDIC cell denoted by the dashed circle in Figure 10.10. Under finite-precision arithmetic, if each individual CORDIC cell is bounded input and bounded output (BIBO), then the compound CORDIC cell is also BIBO.*

Proof. Assume the pipelining level is M, from Figure 10.10, the look-ahead transformed compound CORDIC cell consists of cascade connections of M CORDIC units. Since each of them is BIBO under finite-precision arithmetic, therefore the compound cell is also BIBO.

10.3 Pipelined CORDIC-Based RLS Adaptive Filters

In this section, we apply the annihilation-reording look-ahead techniques to the CORDIC-based RLS adaptive filters and derive fine-grain pipelined topologies. We consider both algorithms with implicit weight extraction (conventional QRD-RLS) and explicit weight extraction (inverse QRD-RLS).

This section is organized as follows. The pipelined QRD-RLS with implicit weight extraction is presented in Section 10.3.1. Its stability under finite-precision arithmetic is studied and proved in Section 10.3.2. Finally, the pipelined adaptive inverse QR algorithm for explicit weight extraction is presented in Section 10.3.3.

10.3.1 Pipelined QRD-RLS with implicit weight extraction

Consider the QRD-RLS formulation given in Equations (10.1), (10.2), (10.3), and (10.4). After the triangular matrix $R(n)$ and the corresponding vector $\mathbf{p}(n)$ are generated, the optimum weight vector $\mathbf{w}(n)$ can be obtained by solving Equation (10.3). The residual error $e(n)$ is then computed as

$$e(n) = y(n) - \mathbf{u}^{\mathrm{T}}(n) R^{-1}(n) \mathbf{p}(n). \tag{10.12}$$

However, for some applications such as adaptive beamforming, this proves to be unnecessary. Since in these cases, the residual error $e(n)$ is usually the only variable interested, and it is not necessary to compute the weight vector $\mathbf{w}(n)$ explicitly. In [6], it is shown that the estimation error $e(n)$ may be obtained directly as the *product* of two variables, the angle-normalized residual $\alpha(n)$ and the likelihood factor $\gamma(n)$. $\alpha(n)$ and $\gamma(n)$ are obtained by applying the same orthogonal transformation matrix $Q(n)$ to the vector $[\mathbf{p}(n-1), y(n)]^{\mathrm{T}}$ and the pinning vector $\boldsymbol{\pi} = [0, \cdots, 0, 1]^{\mathrm{T}}$ [6]. Therefore, the adaptive RLS algorithm can be summarized as

$$\begin{bmatrix} R(n) & \mathbf{p}(n) & \mathbf{s}(n) \\ \mathbf{0}_p^{\mathrm{T}} & \alpha(n) & \gamma(n) \end{bmatrix} = Q(n) \begin{bmatrix} \lambda^{1/2} R(n-1) & \lambda^{1/2} \mathbf{p}(n-1) & \mathbf{0}_p \\ \mathbf{u}^{\mathrm{T}}(n) & y(n) & 1 \end{bmatrix}, \tag{10.13}$$

where $\mathbf{0}_p$ is the p-by-1 null vector. The SFG representation of the algorithm is shown in Figure 10.15, where problem size p is chosen to be 4. The circle and square cells in Figure 10.15 denote CORDIC operations which follow the same notations in Figure 10.1. The circle cell with letter G inside denotes a Gaussian rotation (or a linear CORDIC operation). Its functionality is shown in the figure. Notice that the converting factor cells which generate the likelihood factor γ does not contain recursive operations.

In Figure 10.15, the recursive operation in the cell limits the throughput of the input samples. To increase the sample rates, the annihilation-reording look-ahead technique is applied.

The recursive updating formula for the QRD-RLS with implicit weight extraction is given in Equation (10.13). Its block updating form with block size M is given as follows:

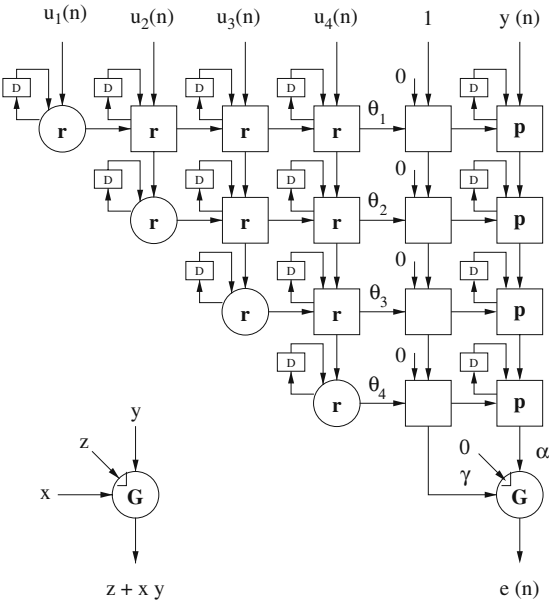


Fig. 10.15 Signal flow graph representation for RLS minimizations.

$$\begin{bmatrix} R(n) & \mathbf{p}(n) & \mathbf{s}(n) \\ O_{M\times p} & \boldsymbol{\alpha}(n) & \boldsymbol{\gamma}(n) \end{bmatrix} = Q(n) \begin{bmatrix} \lambda^{M/2}R(n-M) & \lambda^{M/2}\mathbf{p}(n-M) & \mathbf{0}_p \\ U_M(n) & \mathbf{y}_M(n) & \boldsymbol{\delta}_M \end{bmatrix}, \tag{10.14}$$

where $U_M(n)$ is an M-by-p matrix defined as

$$U_M(n) = [\mathbf{u}(n-M+1), \cdots, \mathbf{u}(n-1), \mathbf{u}(n)]^{\mathrm{T}},$$

and $\mathbf{y}_M(n)$ is an M-by-1 vector defined as

$$\mathbf{y}_M(n) = [y(n-M+1), \cdots, y(n-1), y(n)]^{\mathrm{T}}.$$

In (10.14), $O_{M\times p}$ and $\mathbf{0}_p$ denote M-by-p null matrix and p-by-1 null vector, respectively, $\boldsymbol{\alpha}(n)$ and $\boldsymbol{\gamma}(n)$ are M-by-1 vectors, and $\boldsymbol{\delta}_M$ is a M-by-1 constant vector defined as $\boldsymbol{\delta}_M = [0, \ldots, 0, 1]^{\mathrm{T}}$. The estimation error $e(n)$ can be calculated as the *inner product* of the angle-normalized residual vector $\boldsymbol{\alpha}(n)$ and the likelihood vector $\boldsymbol{\gamma}(n)$, i.e.,

$$e(n) = \boldsymbol{\alpha}^{\mathrm{T}}(n)\,\boldsymbol{\gamma}(n). \tag{10.15}$$

The proof is given in Appendix.

We now determine a sequence of Givens rotations, whose product form the orthogonal transformation matrix $Q(n)$ in (10.14), to annihilate the block input data matrix $U_M(n)$. The order of the Givens rotations is chosen such that the input data is pre-processed and block-data update is finished in the same complexity as a single-data update. This procedure was illustrated in detail in Figure 10.4. A three-level fine-grain pipelined QR update topology was shown in Figure 10.5. After applying the annihilation-reording look-ahead, the concurrent QRD-RLS algorithm can be realized using different implementation styles such as pipelining, block processing, and incremental block processing as discussed in Section 10.2.5. In the rest of the chapter, we only show the topologies for the pipelined realization. The other realizations can be derived similarly. A fine-grain pipelining implementation with pipelining level 3 of CORDIC-based QRD-RLS adaptive filter with implicit weight extraction is shown in Figure 10.16. In this figure, all cell notations follow the notations in Figure 10.15 except that they are compound versions. The internal structure of each compound cell is shown at the bottom part of Figure 10.16. Compared to Figure 10.15, the three-level pipelined architecture tripled the number of CORDIC units and communication bandwidth which is linear with respect to the pipelining level. Thus, in general, the total complexity is $\mathcal{O}[\frac{1}{2}Mp^2]$ CORDIC units per sample time, where p is the input sample size, and M is the pipelining level.

10.3.2 Stability analysis

It is generally recognized that the QR decomposition-based algorithms have good numerical properties, which means that they can perform with an acceptable manner in a short word-length environment. This is due to the fact that the algorithms consist of only orthogonal transformation which leads to inherent stability under finite-precision implementation. From Sections 10.2.1 and 10.2.2, we see that the annihilation-reording look-ahead transformation only involves orthogonal transformation and does not alternate the orthogonality of the algorithm. This implies that the pipelined algorithms also maintain the good numerical properties. Let's define the *stability* of the QRD-RLS algorithm in the sense of BIBO i.e., under finite-precision arithmetic, if the input signals are bounded, then the output residual error $e(n)$ is also bounded. We have the following result:

Theorem 1. *Under finite-precision arithmetic, given a pipelining level M, the M-level fine-grain pipelined CORDIC-based RLS adaptive filter algorithm with implicit weight extraction is stable.*

Proof. In [28], it is shown that for the sequential QRD-RLS algorithm shown in Figure 10.15, a CORDIC cell, operating with finite-precision arithmetic, constitutes a BIBO subsystem of the array. From Figure 10.16, the pipelined algorithm has the same architecture as the sequential one except that all CORDIC cells are compound versions. Therefore, by Lemma 1, a *compound* CORDIC cell, operating with

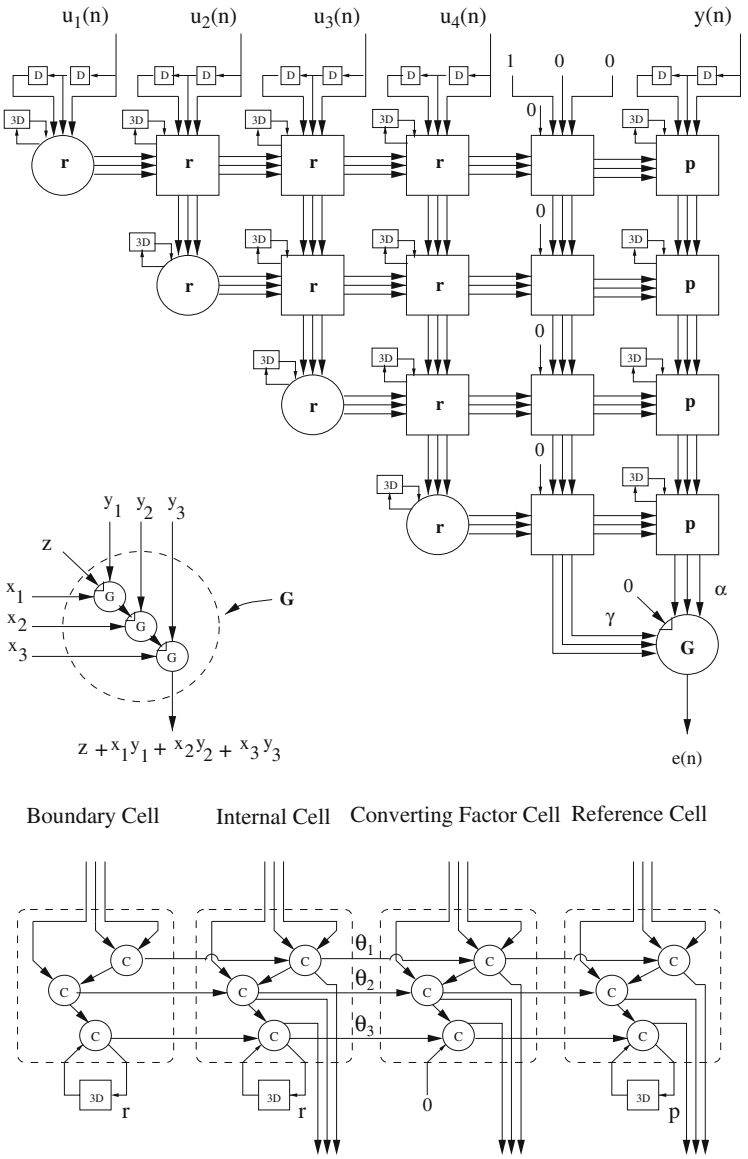


Fig. 10.16 A three-level fine-grain pipelined CORDIC-based implicit weight extraction QRD-RLS adaptive filter architecture.

finite-precision arithmetic, constitutes a BIBO subsystem of the array. Thus, if the desired response $y(n)$ and input samples $\mathbf{u}(n)$ in Figure 10.16 are bounded, the quantized value of the input of the final linear CORDIC cell is also bounded, which leads to the bounded residual error $e(n)$. This completes the proof of Theorem 1.

The stability of other CORDIC-based RLS adaptive filtering algorithms presented in this chapter can also be proved using the similar approach as in Theorem 1 and will not be repeated any further.

10.3.3 Pipelined QRD-RLS with explicit weight extraction

In applications such as channel equalization, RLS-based equalization algorithms such as, e.g., *decision-directed schemes* [29] and *orthogonalized constant modulus algorithms* [30], require the explicit availability of the filter weight vector $\mathbf{w}(n)$. The standard (Gentleman-Kung type) QRD-RLS update scheme involves two computational steps which cannot be efficiently combined on a pipelined array. To circumvent the difficulty, inverse updating-based algorithms are developed [30–32]. Here, we focus on a double-triangular type adaptive inverse QR algorithm [8].

Consider the least squares formulation given in (10.1), (10.2), (10.3), and (10.4). Define the $(p+1)$-by-$(p+1)$ upper triangular compound matrix $\tilde{R}(n)$ as

$$\tilde{R}(n) = \begin{bmatrix} \lambda^{1/2}R(n) & \lambda^{1/2}\mathbf{p}(n) \\ \mathbf{0}_p^{\mathrm{T}} & \gamma(n) \end{bmatrix},$$

where $\gamma(n)$ is a scalar factor, and $R(n), \mathbf{p}(n), \mathbf{0}_p^{\mathrm{T}}$ are defined as in Section 10.1. Using Equation (10.3), $\tilde{R}^{-1}$ is then given as

$$\begin{aligned} \tilde{R}^{-1}(n) &= \begin{bmatrix} \lambda^{-1/2}R^{-1}(n) & -R^{-1}(n)\mathbf{p}(n)/\gamma(n) \\ \mathbf{0}_p^{\mathrm{T}} & 1/\gamma(n) \end{bmatrix} \\ &= \begin{bmatrix} \lambda^{-1/2}R^{-1}(n) & -\mathbf{w}(n)/\gamma(n) \\ \mathbf{0}_p^{\mathrm{T}} & 1/\gamma(n) \end{bmatrix}. \end{aligned}$$

Notice that $\tilde{R}^{-1}$ remains upper triangular and the optimal weight vector $\mathbf{w}(n)$ is explicitly shown on the rightmost column of $\tilde{R}^{-1}$ except for a scaling factor $-1/\gamma$. Now, consider the QR update of the upper triangular compound matrix $\tilde{R}$. From (10.4), we have

$$\begin{bmatrix} \tilde{R}(n) \\ \mathbf{0}_{p+1}^{\mathrm{T}} \end{bmatrix} = \tilde{Q}(n) \begin{bmatrix} \tilde{R}(n-1) \\ \tilde{\mathbf{u}}^{\mathrm{T}}(n) \end{bmatrix}, \tag{10.16}$$

where $\tilde{\mathbf{u}}^{\mathrm{T}}(n) = [\mathbf{u}^{\mathrm{T}}(n), y(n)]$, and $\tilde{Q}(n)$ is determined as products of $(p+1)$ Givens rotation matrices to null the input sample vector $\tilde{\mathbf{u}}^{\mathrm{T}}(n)$ and update matrix $\tilde{R}$. Extending the $(p+2)$-by-$(p+1)$ matrix on the right-hand-side of (10.16) to the $(p+2)$-by-$(p+2)$ square matrix by adding an extra column vector $[\mathbf{0}_{p+1}^{\mathrm{T}}, 1]^{\mathrm{T}}$ to its right leads to

$$\begin{bmatrix} \tilde{R}(n) & \mathbf{v}(n) \\ \mathbf{0}_{p+1}^{\mathrm{T}} & d(n) \end{bmatrix} = \tilde{Q}(n) \begin{bmatrix} \tilde{R}(n-1) & \mathbf{0}_{p+1} \\ \tilde{\mathbf{u}}^{\mathrm{T}}(n) & 1 \end{bmatrix}, \tag{10.17}$$

where vector $\mathbf{v}(n)$ and scalar $d(n)$ correspond to the QR update of vector $\mathbf{0}_{p+1}$ and scalar 1. Inverting the matrix on both sides of Equation (10.17) (the matrix is

non-singular since $\tilde{R}$ is non-singular) and noticing that $Q^{-1} = Q^{\mathrm{T}}$ lead to

$$\begin{bmatrix} \tilde{R}^{-1}(n) & \mathbf{v}'(n) \\ \mathbf{0}_{p+1}^{\mathrm{T}} & d'(n) \end{bmatrix} = \begin{bmatrix} \tilde{R}^{-1}(n-1) & \mathbf{0}_{p+1} \\ -\tilde{\mathbf{u}}^{\mathrm{T}}(n)\tilde{R}^{-1}(n-1) & 1 \end{bmatrix} \tilde{Q}^{\mathrm{T}}(n). \tag{10.18}$$

Taking the transposition on both sides of (10.18), we obtain

$$\begin{bmatrix} \tilde{R}^{-T}(n) & \mathbf{0}_{p+1} \\ \mathbf{v}'^{\mathrm{T}}(n) & d'(n) \end{bmatrix} = \tilde{Q}(n) \begin{bmatrix} \tilde{R}^{-T}(n-1) & -\tilde{R}^{-T}(n-1)\tilde{\mathbf{u}}(n) \\ \mathbf{0}_{p+1}^{\mathrm{T}} & 1 \end{bmatrix}.$$

Thus, we have the following inverse updating formula

$$\begin{bmatrix} \tilde{R}^{-T}(n) \\ \mathbf{v}'^{\mathrm{T}}(n) \end{bmatrix} = \tilde{Q}(n) \begin{bmatrix} \tilde{R}^{-T}(n-1) \\ \mathbf{0}_{p+1}^{\mathrm{T}} \end{bmatrix}. \tag{10.19}$$

Notice that the orthogonal transformation matrix $\tilde{Q}(n)$, which updates the upper triangular matrix $\tilde{R}$ in (10.16), also updates the lower triangular matrix $\tilde{R}^{-T}$ in (10.19). Thus, the double-triangular adaptive inverse QR algorithm can be summarized as follows:

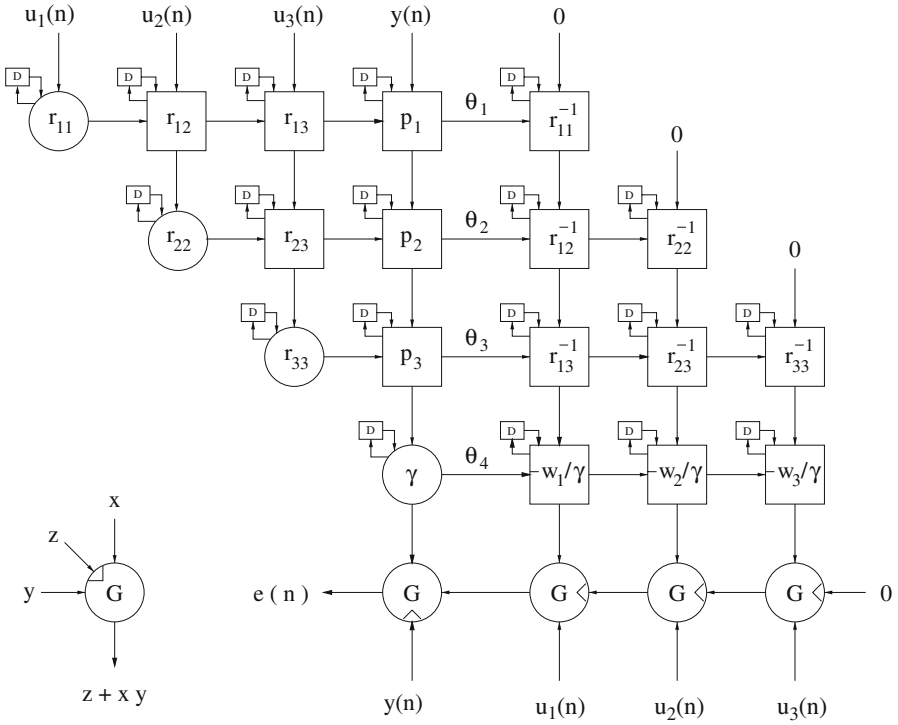
$$\begin{bmatrix} \tilde{R}(n) & \tilde{R}^{-T}(n) \\ \mathbf{0}_{p+1}^{\mathrm{T}} & \mathbf{v}'^{\mathrm{T}}(n) \end{bmatrix} = \tilde{Q}(n) \begin{bmatrix} \tilde{R}(n-1) & \tilde{R}^{-T}(n-1) \\ \tilde{\mathbf{u}}^{\mathrm{T}}(n) & \mathbf{0}_{p+1}^{\mathrm{T}} \end{bmatrix}. \tag{10.20}$$

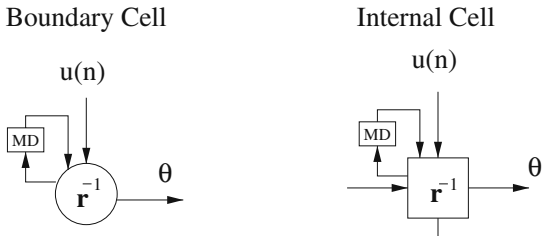
The important point lies in noticing that the scaled weight vector $-\mathbf{w}/\gamma$ *explicitly* sits on the bottom row of lower triangular matrix $\tilde{R}^{-T}$ or the rightmost column of upper triangular matrix $\tilde{R}^{-1}$ as shown before. Therefore, we could achieve parallel weight extraction by taking out the last row elements of $\tilde{R}^{-T}(n)$ and multiply them by the scaling factor $\gamma(n)$ sitting on the lower right corner of upper triangular matrix $\tilde{R}(n)$.

An efficient SFG representation of the CORDIC-based double-triangular adaptive inverse QR algorithm is shown in Figure 10.17. In this figure, the notation follows the ones in Figure 10.1. The operation for updating r^{-1} elements is shown at the bottom part of Figure 10.17. The residual error $e(n)$ is computed according to (10.1) as shown in the figure.

The element on the lower right corner of lower triangular matrix $\tilde{R}^{-T}$ contains value $1/\gamma(n)$ and is not shown in the figure. This algorithm has complexity $\mathcal{O}[p^2]$ Givens rotations per sample period, where p is the size of the array.

From Figure 10.17, we see that the double-triangular adaptive inverse QR algorithm can be easily pipelined at cell level after applying *cut-set* pipelining. The speed or sample rates of the algorithm's implementation is, however, limited by recursive operations in each individual cell as described algebraically by (10.20). We now apply the annihilation-reording look-ahead technique to derive concurrent





$$\begin{bmatrix} r_1^{-1}(n) & r_2^{-1}(n) \\ 0 & u_2'(n) \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} \lambda^{-M/2} r_1^{-1}(n-1) & \lambda^{-M/2} r_2^{-1}(n-1) \\ u_1(n) & u_2(n) \end{bmatrix}$$

Fig. 10.17 Signal flow graph representation of double-triangular type adaptive inverse QR algorithm.

adaptive inverse QR algorithm for high-speed CORDIC-based parallel RLS weight extraction.

The block updating form with block size M of the sequential updating Equation (10.20) is given as follows:

$$\begin{bmatrix} \tilde{R}(n) & \tilde{R}^{-T}(n) \\ 0_{M\times(p+1)} & V(n) \end{bmatrix} = \tilde{Q}(n) \begin{bmatrix} \tilde{R}(n-M) & \tilde{R}^{-T}(n-M) \\ \tilde{U}_M^{\mathrm{T}}(n) & 0_{M\times(p+1)} \end{bmatrix}, \tag{10.21}$$

where $\tilde{U}_M^{\mathrm{T}}(n)$ is a M-by-$(p+1)$ matrix defined as

$$\tilde{U}_M^{\mathrm{T}}(n) = \left[\tilde{\mathbf{u}}(n-M+1) \cdots \tilde{\mathbf{u}}(n-1)\ \tilde{\mathbf{u}}(n) \right]^{\mathrm{T}},$$

$O_{M\times(p+1)}$ denotes a M-by-$(p+1)$ null matrix, and $V(n)$ is a M-by-$(p+1)$ matrix.

The derivation of (10.21) essentially follows the algebraic manipulation in (10.16), (10.17), (10.18), (10.19), and (10.20) provided that we start from the following block update equation

$$\begin{bmatrix} \tilde{R}(n) \\ 0_{M\times(p+1)} \end{bmatrix} = \tilde{Q}(n) \begin{bmatrix} \tilde{R}(n-M) \\ \tilde{U}_M(n) \end{bmatrix}. \tag{10.22}$$

Notice that the $\tilde{Q}(n)$ matrix in (10.22) is different from the $\tilde{Q}(n)$ in (10.16), though we use the same notation here.

Apply the annihilation-reording procedure described in Figure 10.4; we obtain the concurrent architecture shown in Figure 10.5. A complete three-level fine-grain pipelined topology for CORDIC-based QRD-RLS with explicit parallel weight extraction is shown in Figure 10.18. In this figure, all cell notations follow the notations in Figure 10.17 except that some of them are compound versions. The internal structure of each compound cell is shown at the bottom part of Figure 10.18. Compared to Figure 10.17, the number of CORDIC units and communication bandwidth are tripled which is linear with respect to the pipelining level. In general, the total complexity for pipelined realization of CORDIC-based QRD-RLS with explicit weight extraction is $\mathcal{O}[Mp^2]$, where M is the pipelining level and p is the size of input samples.

10.4 Conclusion

In this chapter, the annihilation-reording look-ahead technique is presented to achieve fine-grain pipelining for CORDIC-based RLS adaptive filtering algorithms. It is an exact look-ahead and based on CORDIC arithmetic. The look-ahead transformation can be derived from either the block processing or the iteration point of view, while the former is simpler and more practical and the latter shows the

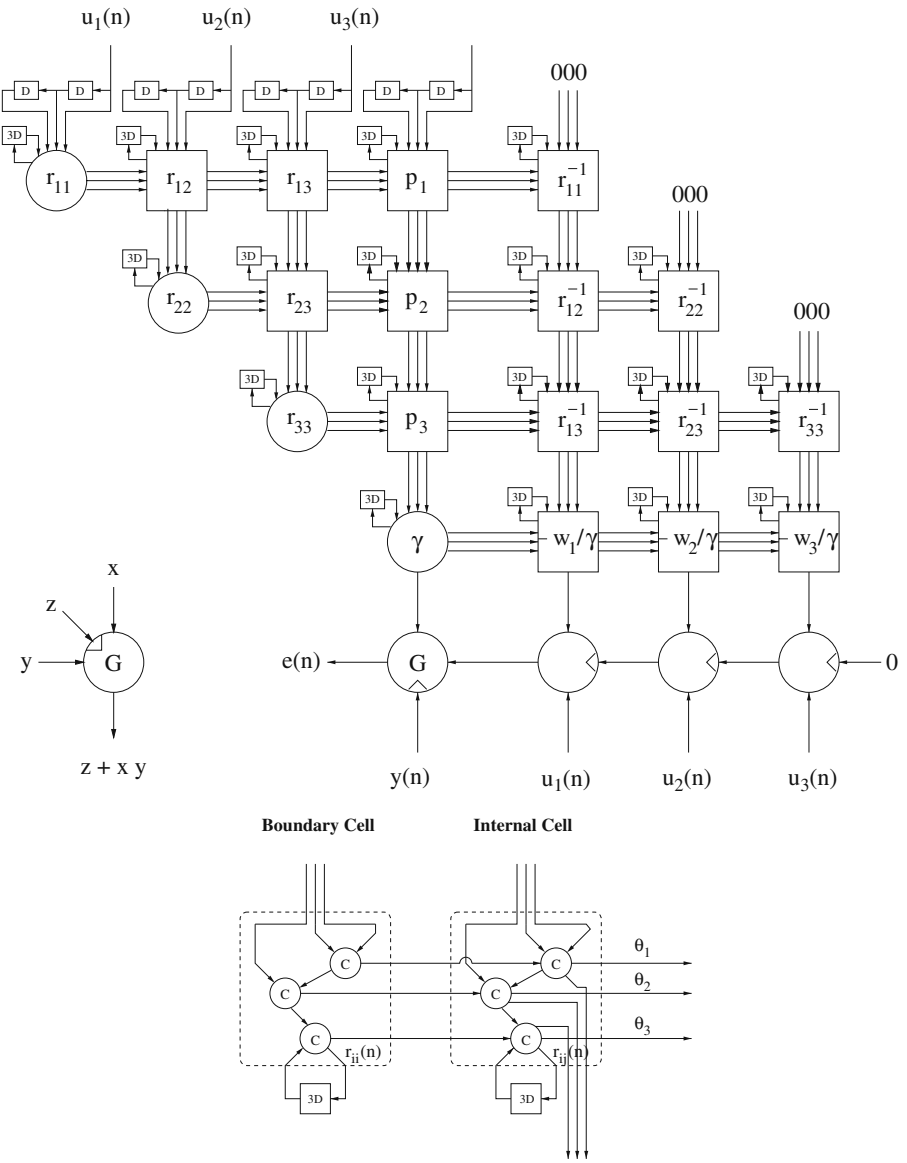


Fig. 10.18 A three–level fine-grain pipelined topology of CORDIC-based double-triangular adaptive inverse QR algorithm.

connection with the traditional multiply–add look-ahead technique. The exploration of the parallelism in the annihilation-reording look-ahead transformation leads to three implementation styles namely pipelining, block processing, and incremental block processing. The implementation complexity in terms of CORDIC units for pipelined realization is the least, and the one for the block processing is the most.

CORDIC-based RLS filters with implicit weight extraction are found useful in applications such as adaptive beamforming, and the ones with explicit weight extraction are found useful in applications such as channel equalization. The application of proposed look-ahead technique to these RLS filters lead to fine-grain pipelined topologies which can be operated at arbitrarily high sample rate. The pipelined algorithms maintain the orthogonality and the stability under finite-precision arithmetic.

QRD-RLS adaptive filters can also be used for adaptive beamforming applications, e.g., the linearly constrained minimum variance (LCMV) adaptive beamformer [33], which will be briefly discussed here. The details can be found in [24]. The LCMV adaptive beamforming is a constrained least squares minimization problem. Solving the constrained minimization problem directly leads to the QRD-MVDR beamforming realization [9]. An alternative is the unconstrained reformulation which leads to the QR decomposition-based generalized sidelobe canceler (GSC) realization [34, 35]. Both MVDR and GSC beamformer can be realized using CORDIC arithmetic. The application of the annihilation-reording look-ahead technique to these adaptive beamforming algorithms leads to fine-grain pipelined topologies [24]. Furthermore, they consist of only Givens rotations which can be mapped onto CORDIC arithmetic-based processors [5].

The implementation complexity in terms of CORDIC units for various RLS-based algorithms and implementation styles are shown in Table 10.2. The implementation complexity of QRD-MVDR and QRD-GSC is obtained from reference [24]. From the table, we see that the pipelining level M is a dimension variable in the complexity expressions for all algorithms and implementation styles. The pipelined and incremental block processing realizations require a linear increasing CORDIC units with an increasing factor of M for the pipelining and a factor of $2M-1$ for the incremental block processing. The complexity factor for the block processing is M^2 which is quadratic with respect to the pipelining level. The adaptive inverse QR algorithm requires approximately two times of CORDIC units as the QRD-RLS algorithm since an extra lower triangular matrix is needed to extract the weight vector. The MVDR topology outperforms GSC topology in terms of complexity by employing a constraint *post-processor* rather than a constraint *pre-processor* for the GSC realization.

Table 10.2 The implementation complexity in terms of CORDIC units for various RLS-based algorithms and implementation styles.

Implementation styles	QRD-RLS	Inverse QR	QRD-MVDR	QRD-GSC
Pipelining	$\frac{1}{2}Mp^2$	Mp^2	$M(\frac{1}{2}p^2+Kp)$	$MK(\frac{1}{2}p^2+p)$
Incremental block	$\frac{1}{2}(2M-1)p^2$	$(2M-1)p^2$	$(2M-1)(\frac{1}{2}p^2+Kp)$	$(2M-1)K(\frac{1}{2}p^2+p)$
Block processing	$\frac{1}{2}M^2p^2$	M^2p^2	$M^2(\frac{1}{2}p^2+Kp)$	$M^2K(\frac{1}{2}p^2+p)$

Appendix

In this appendix, we derive Equations (10.14) and (10.15). At time instance $(n-M)$, apply the QR decomposition to the weighted data matrix $\Lambda^{1/2}(n-M)A(n-M)$ and the reference vector $\mathbf{y}(n)$ as follows:

$$Q(n-M)\Lambda^{1/2}(n-M)\left[A(n-M)\ \mathbf{y}(n-M)\right] = \begin{bmatrix} R(n-M) & \mathbf{p}(n-M) \\ O & \mathbf{v}(n-M) \end{bmatrix}, \quad (10.23)$$

where $R(n-M)$ is p-by-p upper triangular matrix, $\mathbf{p}(n-M)$ and $\mathbf{v}(n-M)$ are p-by-1 and $(n-M-p)$-by-1 vectors, respectively. At time n, the new inputs $U_M(n)$ and $\mathbf{y}_M(n)$ become available processing, we have

$$\left[A(n)\ \mathbf{y}(n)\right] = \begin{bmatrix} A(n-M) & \mathbf{y}(n-M) \\ U_M^{\mathrm{T}}(n) & \mathbf{y}_M(n) \end{bmatrix}. \quad (10.24)$$

Define

$$\bar{\Lambda}^{1/2}(n) = \begin{bmatrix} \lambda^{M/2}\Lambda^{1/2}(n-M) & \\ & I_M \end{bmatrix}, \text{ and} \quad (10.25)$$

$$\bar{Q}(n-M) = \begin{bmatrix} Q(n-M) & \\ & I_M \end{bmatrix}. \quad (10.26)$$

Then

$$\bar{Q}(n-M)\bar{\Lambda}^{1/2}(n)\left[A(n)\ \mathbf{y}(n)\right] = \begin{bmatrix} \lambda^{M/2}R(n-M) & \lambda^{M/2}\mathbf{p}(n-M) \\ O & \lambda^{M/2}\mathbf{v}(n-M) \\ U_M^{\mathrm{T}}(n) & \mathbf{y}_M(n) \end{bmatrix}. \quad (10.27)$$

Notice that here we choose $\bar{\Lambda}^{1/2}(n)$ instead of $\Lambda^{1/2}(n)$. $\Lambda^{1/2}(n)$ differs from $\bar{\Lambda}^{1/2}(n)$ in replacing I_M by $\Lambda^{1/2}(M)$. Using $\Lambda^{1/2}(n)$ will lead to extra operations of the input data. Conversely, using $\bar{\Lambda}^{1/2}(n)$ will not and also not affect the algorithm's convergence behavior due to the existence of $\lambda^{M/2}$ in $\bar{\Lambda}^{1/2}(n)$.

Apply the orthogonal matrix $Q(n)$ which consists of a sequence of Givens rotations to annihilate the input data $U_M(n)$ using $\lambda^{M/2}R(n-M)$ in (10.27), we have

$$\begin{bmatrix} R(n) & \mathbf{p}(n) \\ O & \mathbf{v}(n) \\ O_{M\times N} & \boldsymbol{\alpha}(n) \end{bmatrix} = Q(n) \begin{bmatrix} \lambda^{M/2}R(n-M) & \lambda^{M/2}\mathbf{p}(n-M) \\ O & \lambda^{M/2}\mathbf{v}(n-M) \\ U_M(n) & \mathbf{y}_M(n) \end{bmatrix}, \quad (10.28)$$

which derives the first two columns in (10.14). Next we prove Equation (10.15) and justify the third column in (10.14). From (10.2), we have

$$\mathbf{e}(n-M) = \mathbf{y}(n-M) - A(n-M)\mathbf{w}(n-M). \quad (10.29)$$

By (10.23), we then have

$$Q(n-M)\Lambda^{1/2}(n-M)\mathbf{e}(n-M) = \begin{bmatrix} \mathbf{p}(n-M) \\ \mathbf{v}(n-M) \end{bmatrix} - \begin{bmatrix} R(n-M) \\ O \end{bmatrix} \mathbf{w}(n-M). \tag{10.30}$$

Let

$$\begin{aligned} \boldsymbol{\varepsilon}(n-M) &= Q^{1/2}(n-M)\Lambda^{1/2}(n-M)\mathbf{e}(n-M) \\ \bar{\mathbf{e}}_M(n) &= \mathbf{y}_M(n) - U_M^{\mathrm{T}}(n)\mathbf{w}(n-M) \\ \mathbf{e}_M(n) &= \mathbf{y}_M(n) - U_M^{\mathrm{T}}(n)\mathbf{w}(n), \end{aligned} \tag{10.31}$$

where $\boldsymbol{\varepsilon}(n-M), \bar{\mathbf{e}}_M(n)$, and $\mathbf{e}_M(n)$ are M-by-1 vectors. By (10.1), it is seen that the last element in $\mathbf{e}_M(n)$ is the desired residual error $e(n)$ at time instance n. From (10.30) and (10.31), we obtain

$$\begin{bmatrix} \lambda^{M/2}\boldsymbol{\varepsilon}(n-M) \\ \bar{e}_M(n) \end{bmatrix} = \begin{bmatrix} \lambda^{M/2}\mathbf{p}(n-M) \\ \lambda^{M/2}\mathbf{v}(n-M) \\ \mathbf{y}_M(n) \end{bmatrix} - \begin{bmatrix} \lambda^{M/2}R(n-M) \\ O \\ U_M^{\mathrm{T}}(n) \end{bmatrix} \mathbf{w}(n-M). \tag{10.32}$$

Apply the orthogonal matrix $Q(n)$ to both sides of (10.32) and use (10.28) resulting

$$Q(n)\begin{bmatrix} \lambda^{M/2}\boldsymbol{\varepsilon}(n-M) \\ \mathbf{e}_M(n) \end{bmatrix} = \begin{bmatrix} \mathbf{p}(n) \\ \mathbf{v}(n) \\ \boldsymbol{\alpha}(n) \end{bmatrix} - \begin{bmatrix} R(n) \\ O \\ O \end{bmatrix} \mathbf{w}(n). \tag{10.33}$$

Notice that, after annihilating the input block data $U_M^{\mathrm{T}}(n)$, the weight vector $\mathbf{w}(n-M)$ has been updated to $\mathbf{w}(n)$. Correspondingly, the residual error $\bar{\mathbf{e}}_M(n)$ becomes $\mathbf{e}_M(n)$ as defined in (10.31). Now, moving $Q(n)$ to the right-hand-side of (10.33), and noticing that $Q(n)$ is orthogonal, we obtain

$$\begin{bmatrix} \lambda^{M/2}\boldsymbol{\varepsilon}(n-M) \\ \mathbf{e}_M(n) \end{bmatrix} = Q^{\mathrm{T}}(n)\begin{bmatrix} \mathbf{p}(n) - R(n)\mathbf{w}(n) \\ \mathbf{v}(n) \\ \boldsymbol{\alpha}(n) \end{bmatrix}. \tag{10.34}$$

Since the optimum weight vector $\mathbf{w}(n)$ satisfies $\mathbf{p}(n) - R(n)\mathbf{w}(n) = \mathbf{0}_p$, (10.34) reduces to

$$\begin{bmatrix} \lambda^{M/2}\boldsymbol{\varepsilon}(n-M) \\ \mathbf{e}_M(n) \end{bmatrix} = Q^{\mathrm{T}}(n)\begin{bmatrix} \mathbf{0}_p \\ \mathbf{v}(n) \\ \boldsymbol{\alpha}(n) \end{bmatrix}. \tag{10.35}$$

Noticing that the last element of $\mathbf{e}_M(n)$ is $e(n)$, we have

$$\begin{aligned} e(n) &= \begin{bmatrix} \mathbf{0}_{n-M}^{\mathrm{T}} & \delta_M^{\mathrm{T}} \end{bmatrix} Q^{\mathrm{T}}(n)\begin{bmatrix} \mathbf{0}_p \\ \mathbf{v}(n) \\ \boldsymbol{\alpha}(n) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{0}_{n-M}^{\mathrm{T}} & \delta_M^{\mathrm{T}} \end{bmatrix} Q^{\mathrm{T}}(n)\begin{bmatrix} O_{(n-M)\times M} \\ I_M \end{bmatrix} \boldsymbol{\alpha}(n). \end{aligned} \tag{10.36}$$

The second equality is due to the fact that $Q^{\mathrm{T}}(n)$ only makes use of elements $\mathbf{0}_p$ and $\boldsymbol{\alpha}(n)$ in the vector $[\mathbf{0}_p^{\mathrm{T}}, \mathbf{v}^{\mathrm{T}}(n), \boldsymbol{\alpha}^{\mathrm{T}}(n)]^{\mathrm{T}}$. Taking the transpose on both sides of (10.36), and noticing that $e(n)$ is a scalar, then

$$\begin{aligned} e(n) &= \boldsymbol{\alpha}^{\mathrm{T}}(n) \left[O_{M\times(n-M)} \; I_M \right] \left(Q(n) \begin{bmatrix} O_{n-M} \\ \delta_M \end{bmatrix} \right) \\ &= \boldsymbol{\alpha}^{\mathrm{T}}(n) \left[O_{M\times(n-M)} \; I_M \right] \begin{bmatrix} \mathbf{s}(n) \\ \boldsymbol{\gamma}(n) \end{bmatrix} \\ &= \boldsymbol{\alpha}^{\mathrm{T}}(n) \, \boldsymbol{\gamma}(n). \end{aligned} \tag{10.37}$$

The second equality justify the third column in (10.14). This completes the derivation of (10.14) and (10.15).

References

1. S. Haykin, Adaptive Filter Theory. 2nd edition Prentice-Hall, Englewood Cliffs, NJ, USA (1991)
2. J. E. Volder, The CORDIC trigonometric computing technique. IRE Transactions on Electronic Computers, vol. EC-8, no. 3, pp. 330–334 (September 1959)
3. Y. H. Hu, Cordic-based VLSI architectures for digital signal processing. IEEE Signal Processing Magazine, no. 3, vol. 9, pp. 16–35 (July 1992)
4. G.J. Hekstra and E. F. Deprettere, Floating point CORDIC. 11th Symposium on Computer Arithmetic, Windsor, Canada, pp. 130–137 (June 1993)
5. E. Rijpkema, G. Hekstra, E. Deprettere, and J. Ma, A strategy for determining a Jacobi specific dataflow processor. IEEE International Conference on Application-Specific Systems, Architectures and Processors, Zurich, Switzerland, pp. 53–64 (July 1997)
6. J. G. McWhirter, Recursive least-squares minimization using a systolic array. Proc. SPIE: Real Time Signal Processing VI, vol. 431, pp. 105–112 (January 1983)
7. W. M. Gentleman and H. T. Kung, Matrix triangularization by systolic arrays. Proceedings SPIE: Real-Time Signal Processing IV, vol. 298, pp. 298–303 (January 1981)
8. T. J. Shepherd, J. G. McWhirter, and J. E.Hudson, Parallel weight extraction from a systolic adaptive beamformer. Mathematics in Signal Processing II (J.G. McWhirter ed.), pp. 775–790, Clarendon Press, Oxford (1990)
9. J. G. McWhirter and T. J. Shepherd, Systolic array processor for MVDR beamforming. IEE Proceedings, vol. 136, pp. 75–80 (April 1989)
10. K. K. Parhi, Algorithm transformation techniques for concurrent processors. Proceedings of the IEEE, vol. 77, pp. 1879–1895 (December 1989)
11. K. K. Parhi and D. G. Messerschmitt, Pipeline interleaving and parallelism in recursive digital filters–Part I: Pipelining using scattered look-ahead and decomposition. IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37, pp. 1118–1134 (July 1989)
12. K. K. Parhi and D. G. Messerschmitt, Pipeline interleaving and parallelism in recursive digital filters–Part II: Pipelined incremental block filtering. IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37, pp. 1099–1117 (July 1989)
13. A. P. Chandrakasan, S. Sheng, and R. W. Broderson, Low-power CMOS digital design. IEEE Journal on Solid-State Circuits, vol. 27, pp. 473–484 (April 1992)
14. K. K. Parhi, VLSI Digital Signal Processing Systems, Design and Implementation. John Wiley & Sons, New York, NY, USA (1999)
15. K. J. Raghunath and K. K. Parhi, Pipelined RLS adaptive filtering using Scaled Tangent Rotations (STAR). IEEE Transactions on Signal Processing, vol. 40, pp. 2591–2604 (October 1996)
16. T. H. Y. Meng, E. A. Lee, and D. G. Messerschmitt, Least-squares computation at arbitrarily high speeds, International Conference on Acoustics, Speech, and Signal Processing, ICASSP'1987, Dallas, USA, pp. 1398–1401 (April 1987)

17. G. H. Golub and C. F. V. Loan, Matrix Computation. Johns Hopkins University Press, Baltimore, MD, USA (1989)
18. J. M. Cioffi, The fast adaptive ROTOR RLS algorithm. IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 38, pp. 631–653 (April 1990)
19. S. F. Hsieh, K. J. R. Liu, and K. Yao, A unified square-root-free Givens rotation approach for QRD-based recursive least squares estimation. IEEE Transactions on Signal Processing, vol. 41, pp. 1405–1409 (March 1993)
20. S. Hammarling, A note on modifications to Givens plane rotation. IMA Journal of Applied Mathematics, vol. 13, no. 2, pp. 215–218 (1974)
21. J. L. Barlow and I. C. F. Ipsen, Scaled Givens rotations for solution of linear least-squares problems on systolic arrays. SIAM Journal on Scientific and Statistical Computing, vol. 13, no. 5, pp. 716–733 (September 1987)
22. J. Götze and U. Schwiegelshohn, A square-root and division free Givens rotation for solving least-squares problems on systolic arrays. SIAM Journal on Scientific and Statistical Computing, vol. 12, no. 4, pp. 800–807 (July 1991)
23. E. Franrzeskakis and K. J. R. Liu, A class of square-root and division free algorithms and architectures for QRD-based adaptive signal processing. IEEE Transactions on Signal Processing, vol. 42, pp. 2455–2469 (September 1994)
24. J. Ma, K. K. Parhi, and E. F. Deprettere, Annihilation reordering lookahead pipelined CORDIC based RLS adaptive filters and their application to adaptive beamforming. IEEE Transactions on Signal Processing, vol. 48, pp. 2414–2431 (August 2000)
25. C. E. Leiserson, F. Rose, and J. Saxe, Optimizing synchronous circuitry by retiming. 3rd Caltech Conference on VLSI, Pasadena, USA, pp. 87–116 (March 1983)
26. K. K. Parhi, High-level algorithm and architecture transformations for DSP synthesis. Journal of VLSI Signal Processing, vol. 9, pp. 121–143 January 1995
27. E. F. Deprettere, P. Held, and P. Wielage, Model and methods for regular array design. International Journal of High Speed Electronics; Special issue on Massively Parallel Computing–Part II, vol. 4(2), pp. 133–201 (1993)
28. H. Leung and S. Haykin, Stability of recursive QRD-LS algorithms using finite-precision systolic array implementation. IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37, no. 5, pp. 760–763 (May 1989)
29. M. Moonen and E. F. Deprettere, A fully pipelined RLS-based array for channel equalization. Journal of VLSI Signal Processing, vol. 14, pp. 67–74 (October 1996)
30. R. Gooch and J. Lundell, The CM array: an adaptive beamformer for constant modulus signals. International Conference on Acoustics, Speech, and Signal Processing, ICASSP'1986, Tokyo, Japan, pp. 2523–2526 (April 1986)
31. C.-T. Pan and R. J. Plemmons, Least squares modifications with inverse factorizations: parallel implications. Journal of Computational and Applied Mathematics, vol. 27, pp. 109–127 (September 1989)
32. S. T. Alexander and A. L. Ghirnikar, A method for recursive least squares filtering based upon an inverse QR decomposition. IEEE Transactions on Signal Processing, vol. SP-41, no. 1, pp. 20–30 (January 1993)
33. O. L. Frost III, An algorithm for linearly constrained adaptive array processing. Proceedings of the IEEE, vol. 60, no. 8, pp. 926–935 (August 1972)
34. R. L. Hanson and C. L. Lawson, Extensions and applications of the Householder algorithm for solving linear least squares problems. AMS Mathematics of Computation, vol. 23, no. 108, pp. 787-812 (October 1969)
35. S. P. Applebaum and D. J. Chapman, Adaptive arrays with main beam constraints. IEEE Transactions on Antennas and Propagation, vol. AP-24, no. 5, pp. 650–662 (September 1976)