Pipelined RLS Adaptive Architecture Using Relaxed Givens Rotations (RGR)

Lan-Da Van and Chih-Hong Chang*

Chip Implementation Center (CIC), National Science Council,
No. 1, Prosperity Rd. 1, Science-Based Industrial Park, Hsinchu, Taiwan, R.O.C.
*AT CHIP CORPORATION, Taipei, Taiwan, R.O.C.
E-mail: ldvan@cic.edu.tw

Abstract

In this paper, we focus on developing a new relaxed Givens rotations (RGR)-RLS algorithm and the corresponding RGR-RLS systolic array. The resulting algorithm and architecture possess fine-grain pipelining, nearly the same convergence as the QRD-RLS, good robustness for λ, and square-root free computation with a little area overhead.

I. Introduction

Recursive least squares (RLS) based adaptive digital filters have wide applications in adaptive equalization [1], beamforming and image processing. Historically, the gradient descent algorithms such as the least-mean-square (LMS) and delay LMS (DLMS) [2] algorithms are very cost-effective but unfortunately they are not suitable for all applications. The incurred major problem based on the LMS/DLMS algorithm is the slow convergence rate for a broad dynamic range signal environment. The convergence of the RLS algorithm is faster than that of the LMS and DLMS algorithms, but its computational complexity higher than the latter is an order of magnitude. The QR decomposition (QRD)-RLS algorithm [3-4] using triangularization process is the most promising RLS algorithm since it is known to have good numerical properties and can be mapped to a coarse-grain pipelining systolic array. The QRD-RLS algorithm is, hence, very suited to VLSI implementation.

The critical period of the QRD-RLS algorithm is limited by the operation time in the recursive loop of the individual cells. In many applications such as equalization and image restoration, very high throughput would be desired, and the QRD-RLS algorithm may not be capable of operating at such high throughput. In order to overcome this drawback, some research has provided several schemes as follows [5, 6]. However, these algorithms also have the same fine-grain pipelining difficulty as the QRD-RLS algorithm. Apart from being used to increase speed, fine-grain pipelining can also be used to reduce power dissipation in low to moderate speed applications. To increase the speed of the QRD-RLS algorithm, the look-ahead technique leading to fine-grain pipelining can be used. However, using look-ahead in the QRD-RLS algorithm results in large hardware overhead. Consequently, this technique is not practical for the QRD-RLS algorithm. Recently, the STAR-RLS algorithm [7] solves the fine-grain pipelining difficulty; however, it diverges for small value of the forgetting factor. It is known that the smaller value of the forgetting factor results in faster convergence than the larger one. In [8], CORDIC-based QRD-RLS needs ROM/RAM that consumes a large area to implement the architecture. Based on these unsolved problems, we are motivated to provide a new algorithm and architecture. This paper is organized as follows. We propose the RGR-RLS and Pipelined RGR-RLS (PRGR-RLS) algorithms as well as architectures in Sections II and III, respectively. In Section IV, comparisons and simulation results are discussed. Conclusion is given in the last section.

II. RGR-RLS Algorithm and Architecture

We are given a time series of inputs $x(1)$, $x(2)$, …, $x(n)$, and we want to estimate some desired signal $d(i)$ based on a weighted sum of present sample and a few of the past samples. All the data are assumed to be real. In order to efficiently approach the optimal solution based on least square criterion, the QRD-RLS algorithm using the Givens rotations can be applied, where a Givens rotation is defined as

$$\mathbf{G}(n) = \begin{bmatrix} c(n) & s(n) \\ -s(n) & c(n) \end{bmatrix}, \tag{1}$$

The QRD-RLS algorithm at each boundary cell in [1] has the following equation

$$r(n) = c(n)\lambda^{1/2} r(n-1) + s(n)x(n)\ , \tag{2a}$$

$$= \sqrt{\lambda r^2(n-1) + x^2(n)}\ , \tag{2b}$$

where $c(n)$ is to represent $\cos\theta$ that is a function of iteration number n in the triangularization process and $s(n)$ has a similar definition. At the internal cell, in addition to Eq. (2a), another equation can be easily obtained as

$$b(n) = -s(n)\lambda^{1/2} r(n-1) + c(n)x(n)\ , \tag{3}$$

where $b(n)$ denotes the output of the internal cell. In order to bypass square root and achieve fine-grain pipelining, we take some approaches to relax the Givens rotation. Therefore, the resulting RGR is assumed in the form:

$$\mathbf{G}_R(n) = \begin{bmatrix} c_1(n) & s_1(n) \\ -s_2(n) & c_2(n) \end{bmatrix}. \tag{4}$$

Note that $\mathbf{G}_R(n)$ is very close to $\mathbf{G}(n)$ after the verification of the adaptive equalization experiment. In Eq. (2b), there are two cases to be investigated.

Case 1: $\lambda^{1/2}|r(n-1)| \geq |x(n)|$

In this case, using the product relaxation, Eq. (2b) can be modified as

$$r(n) = \lambda^{1/2}|r(n-1)|(1 + \frac{x^2(n)}{\lambda r^2(n-1)})^{1/2}$$

$$\cong \lambda^{1/2}|r(n-1)| + \frac{1}{2} \times \frac{x^2(n)}{\lambda^{1/2}|r(n-1)|}\ . \tag{5}$$

At the steady-state of Eq. (2b), we see that $r(n)$ and $r(n-1)$ have the same positive sign at the boundary cell. Thus, Eq. (5) can be modified as

$$r(n) \cong \lambda^{1/2} r(n-1) + \frac{1}{2} \times \frac{x^2(n)}{\lambda^{1/2} r(n-1)}\ . \tag{6}$$

Corresponding to Eq. (2a), we can obtain the relaxed functions $c_1(n)$ and $s_1(n)$, respectively, as

$$c_1(n) = 1, \tag{7a}$$

$$s_1(n) = \frac{x(n)}{2\lambda^{1/2} r(n-1)}. \tag{7b}$$

The definition of $c(n)$ and $s(n)$ are relaxed and then defined, respectively, as

$$c(n) = \frac{\lambda^{1/2} r(n-1)}{r(n)} = \frac{\lambda^{1/2} r(n-1)}{\lambda^{1/2} r(n-1) + \frac{1}{2} \times \frac{x^2(n)}{\lambda^{1/2} r(n-1)}}$$

$$\cong 1 = c_2(n), \tag{8a}$$

$$s(n) = \frac{x(n)}{r(n)} \cong \frac{x(n)}{\lambda^{1/2} r(n-1) + \frac{1}{2} \times \frac{x^2(n)}{\lambda^{1/2} r(n-1)}}$$

$$\cong \frac{x(n)}{\lambda^{1/2} r(n-1)} = s_2(n). \tag{8b}$$

Applying Eqs. (8a) and (8b), $b(n)$ can be rewritten as

$$b(n) = -s(n)\lambda^{1/2} r(n-1) + c(n)x(n)$$

$$\cong -s_2(n)\lambda^{1/2} r(n-1) + c_2(n)x(n). \tag{9}$$

From Eqs. (7a), (7b), (8a) and (8b), we can obtain the first RGR as

$$\mathbf{G}_R(n) = \begin{bmatrix} 1 & \frac{x(n)}{2\lambda^{1/2} r(n-1)} \\ -\frac{x(n)}{\lambda^{1/2} r(n-1)} & 1 \end{bmatrix}. \tag{10}$$

Case 2: $\lambda^{1/2}|r(n-1)| < |x(n)|$

In this case, because $\lambda^{1/2}|r(n-1)| < |x(n)|$, we can directly obtain the relationship $\lambda r^2(n-1) < x^2(n)$. In similar fashion, Eq. (2b) can be modified as

$$r(n) \cong |x(n)| + \frac{\lambda r^2(n-1)}{2|x(n)|}. \tag{11}$$

Referring to the discussion of [9], the second RGR can be obtained as

$$\mathbf{G}_R(n) = \begin{bmatrix} \frac{\lambda^{1/2}}{2} & sign(x(n)) \\ -sign(x(n)) & \frac{\lambda^{1/2} r(n-1)}{|x(n)|} \end{bmatrix}. \tag{12}$$

Eqs. (10) and (12) are our proposed RGR. The RGR-RLS algorithm can be mapped to RGR-RLS architecture as shown in Fig. 1, where the boundary and internal cells are depicted in Figs. 2(a) and 2(b), respectively. In Figs. 2(a) and 2(b), the symbols MUX, $\overset{\text{SR}}{\Rightarrow}$, $\overset{\text{SR}}{\Leftarrow}$, and $|\bullet|$ denote a multiplexer, shift one-bit right register, shift one-bit left register, and absolute value of $\bullet$, respectively. When $\rho = 0$ and $\rho = 1$, MUX selects upper and lower signals, respectively, where ρ is the selection signal.

III. Pipelined RGR-RLS (PRGR-RLS) Algorithm and Architecture

We go further to debate the fine-grain pipelining issue for the proposed boundary and internal cells in the RGR-RLS architecture as shown in Fig. 2. Using D_1-step look-ahead technique, the recursive loop can be written as

$$r(n) = c_1(n)\lambda^{1/2} r(n-1) + s_1(n)x(n)$$

$$= \begin{cases} \lambda^{D_1/2} r(n-D_1) + \sum_{i=0}^{D_1-1} \lambda^{i/2} s_1(n-i)x(n-i) & ,\text{if } \lambda^{1/2}|r(n-1)| \geq |x(n)| \\ (\frac{\lambda}{2})^{D_1} r(n-D_1) + \sum_{i=0}^{D_1-1} (\frac{\lambda}{2})^i s_1(n-i)x(n-i) & ,\text{otherwise} \end{cases} \tag{13}$$

where

$$s_1(n) = \begin{cases} \frac{x(n)}{2\lambda^{1/2} r(n-1)} & ,\text{if } \lambda^{1/2}|r(n-1)| \geq |x(n)| \\ sign(x(n)) & ,\text{otherwise} \end{cases} \tag{14}$$

Eq. (13) cannot be completely pipelined since the second term in Eq. (13) cannot be pipelined. Thus, we apply the delay relaxation to Eqs. (13) and (14) as follows

$$r(n) = \begin{cases} \lambda^{D_1/2} r(n-D_1) + \sum_{i=0}^{D_1-1} \lambda^{i/2} s_1(n-D_2-i)x(n-D_2-i) & ,\text{if } \lambda^{1/2}|r(n-D_1)| \geq |x(n-D_2)| \\ (\frac{\lambda}{2})^{D_1} r(n-D_1) + \sum_{i=0}^{D_1-1} (\frac{\lambda}{2})^i s_1(n-D_2-i)x(n-D_2-i) & ,\text{otherwise} \end{cases} \tag{15}$$

where

$$s_1(n-D_2) = \begin{cases} \frac{x(n-D_2)}{2\lambda^{1/2} r(n-D_2)} & ,\text{if } \lambda^{1/2}|r(n-D_1)| \geq |x(n-D_2)| \\ sign(x(n-D_2)) & ,\text{otherwise} \end{cases} \tag{16}$$

In order to match timing sequence, the relation between D_1 and D_2 can be restricted as

$$D_2 = D_1. \tag{17}$$

In similar fashion, Eq. (3) can be written as

$$b(n) = -s_2(n-D_1)\lambda^{1/2} r(n-D_1) + c_2(n-D_1)x(n-D_1), \tag{18}$$

where

$$s_2(n-D_1) = \begin{cases} \frac{x(n-D_1)}{\lambda^{1/2} r(n-D_1)} & ,\text{if } \lambda^{1/2}|r(n-D_1)| \geq |x(n-D_1)| \\ sign(x(n-D_1)) & ,\text{otherwise} \end{cases} \tag{19a}$$

$$c_2(n-D_1) = \begin{cases} 1 & ,\text{if } \lambda^{1/2}|r(n-D_1)| \geq |x(n-D_1)| \\ \frac{\lambda^{1/2} r(n-D_1)}{|x(n-D_1)|} & ,\text{otherwise} \end{cases} \tag{19b}$$

Therefore, Eqs. (15) to (19b) are called as the PRGR-RLS algorithm, which can be mapped onto the same array as shown in Fig. 1. The boundary and internal cells for PRGR-RLS architecture are depicted in Figs. 3(a) and 3(b), respectively.

IV. Comparison and Simulation Results

In this section, first, we compare several architectures including the QRD-RLS, STAR-RLS, CORDIC-based RLS, and our proposed architecture in terms of critical period, orthogonality, robustness for λ, and need for memory as well as square-root. In Table I, we can see that our proposed architecture possesses fine-grain pipelining, nearly orthogonality that is verified by simulation, good robustness for λ, and square-root free with a little area overhead among the existing architectures [3, 7, 8]. Therefore, the relaxed Givens rotations result in an efficient architecture.

Next, we verify that the convergence of our proposed RGR-RLS algorithm is close to that of the QRD-RLS algorithm for adaptive equalization application. The simulation environment of the adaptive equalization is set to the same as described in [1], where the amount of amplitude distortion produced by the channel is set to 3.3. For fair comparisons, we simulate three cases: $\lambda = 0.2$, $\lambda = 0.5$, and $\lambda = 0.9$. For small value of λ such as $\lambda = 0.2$, the learning curves as shown in Fig. 4 with 100 runs show that the STAR-RLS cannot converge, but our proposed RGR-RLS architecture has similar convergence performance to the QRD-RLS architecture. For the second case $\lambda = 0.5$, this is the theoretical maximum relaxation error as proved in [9]. From the simulation result as shown in Fig. 5, it can be clearly seen that the RGR-RLS architecture still maintain low relaxation error since our convergence performance is close to that of the STAR and QRD-RLS architectures. As compared with the LMS (for $\mu = 0.027$), QRD-RLS, and STAR-RLS architectures, the simulation result as shown in Fig. 6 reveals that our pipelined architectures ($D_1 = 2$ and $D_1 = 5$) have better convergence performance than that of the LMS algorithm. From Figs. 4, 5, and 6, the proposed RGR-RLS algorithm has better orthogonality and robustness than the STAR-RLS algorithm.

V. Conclusion

A new look at the Givens rotation is developed in this paper. Using the relaxed Givens rotations, the resulting algorithm and architecture possess fine-grain pipelining, nearly the same convergence as the QRD-RLS, good robustness for λ, and square-root free computation with a little area overhead.

References

[1] S. Haykin, *Adaptive Filter Theory*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[2] L. D. Van and W. S. Feng, "An efficient systolic architecture for the DLMS adaptive filter and its applications," *IEEE Trans. Circuits Syst., II,* vol. 48, pp. 359-366, Apr. 2001.

[3] W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic arrays," in *Proc. SPIE: Real Time Signal Process. IV*, 1981, pp. 298-303.

[4] J. G. McWhirter, "Recursive least-squares minimization using a systolic array," in *Proc. SPIE: Real Time Signal Process. VI*, vol. 431, 1983, pp. 105-112.

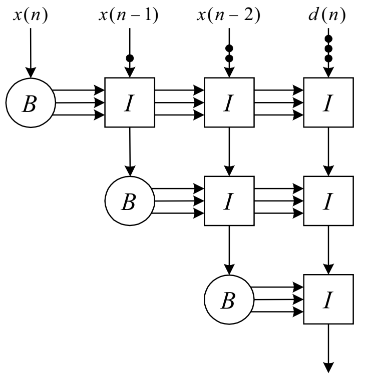
[5] S. F. Hsieh, K. J. R. Liu, and K. Yao, "A unified square-root-free Givens rotation approach for QRD-based recursive least squares estimation," *IEEE Trans. Signal Processing*, vol. 41, pp. 1405-1409, Mar. 1993.

[6] E. Frantzeskakis and K. J. R. Liu, "A class of square-root and division free algorithms and architectures for QRD-based adaptive signal processing," *IEEE Trans. Signal Processing*, vol. 42, pp. 2455-2469, Sept. 1994.

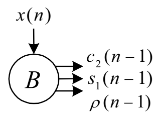
[7] K. J. Raghunath and K. K. Parhi, "Pipelined RLS adaptive filtering using scaled tangent rotations (STAR)," *IEEE Trans. Signal Processing*, vol. 40, pp. 2591-2604, Oct. 1996.

[8] J. Ma, K. K. Parhi, and E. F. Deprettere, "Annihilation-reordering look-ahead pipelined CORDIC-based RLS adaptive filters and their application to adaptive beamforming," *IEEE Trans. Signal Processing*, vol. 48, pp. 2414-2431, Aug. 2000.

[9] L. D. Van, *Design of Efficient VLSI Architectures: Multiplier, 2-D Digital Filter, and Adaptive Digital Filter*, Ph. D. dissertation, Dept. of the Electrical Engineering, National Taiwan University, Taipei, Taiwan, R.O.C., 2001.



Boundary Cell



If $\lambda^{1/2}|r(n-1)| \ge |x(n)|$

$$c_1(n) = 1;\ \ s_1(n) = \frac{x(n)}{2\lambda^{1/2} r(n-1)};$$

$$c_2(n) = 1;\ \ s_2(n) = \frac{x(n)}{\lambda^{1/2} r(n-1)};\ \ \rho(n) = 0$$

else

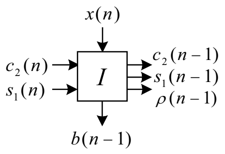
$$c_1(n) = \frac{\lambda^{1/2}}{2};\ \ \ s_1(n) = sign(x(n));$$

$$c_2(n) = \frac{\lambda^{1/2} r(n-1)}{|x(n)|};\ \ s_2(n) = sign(x(n));\ \rho(n) = 1$$

end

$$r(n) = c_1(n)\lambda^{1/2} r(n-1) + s_1(n)x(n)$$

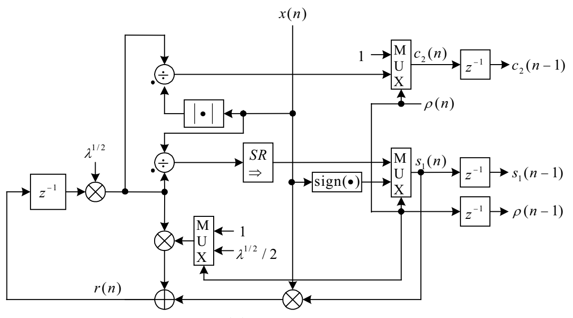
Internal Cell



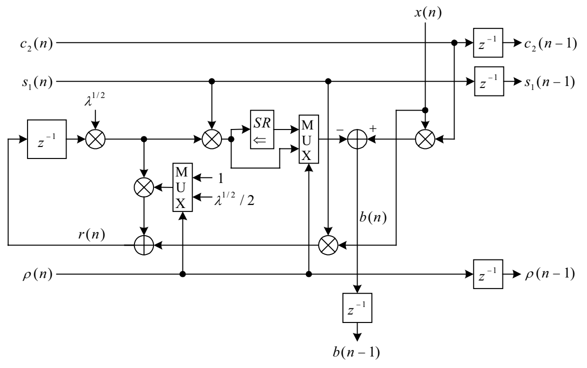
$$r(n) = c_1(n)\lambda^{1/2} r(n-1) + s_1(n)x(n)$$

$$b(n) = -s_2(n)\lambda^{1/2} r(n-1) + c_2(n)x(n)$$

Fig. 1. Systolic array for the RGR-RLS algorithm.

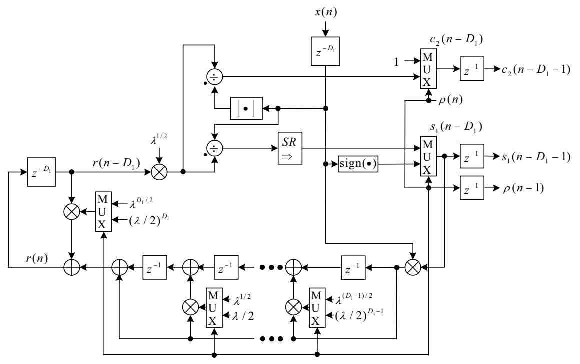


(a)

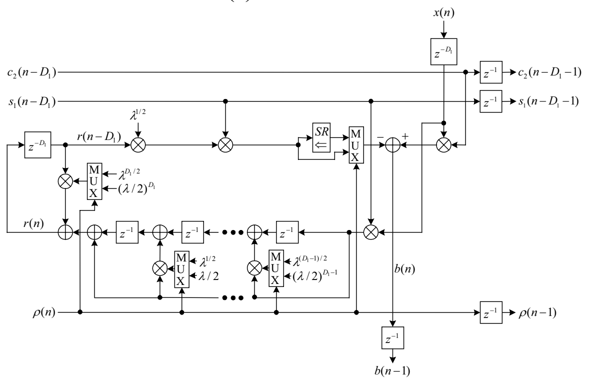


(b)

Fig. 2. (a) Boundary cell and (b) internal cell of the RGR-RLS architecture.



(a)



(b)

Fig. 3. (a) Boundary cell and (b) internal cell of the PRGR-RLS architecture.

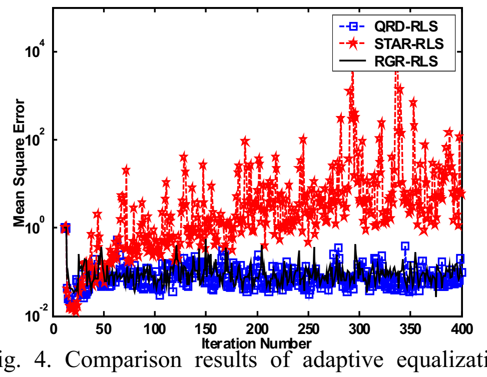


Fig. 4. Comparison results of adaptive equalization using QRD-RLS, STAR-RLS, RGR-RLS architectures for $\lambda = 0.2$.

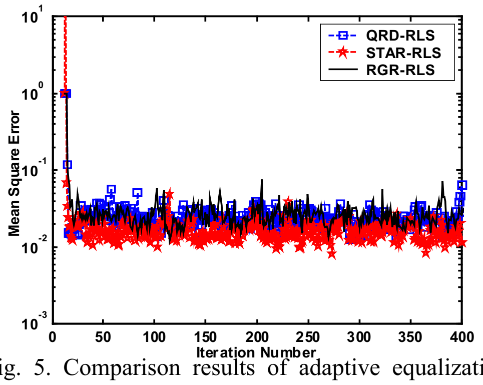


Fig. 5. Comparison results of adaptive equalization using QRD-RLS, STAR-RLS, RGR-RLS architectures for $\lambda = 0.5$.

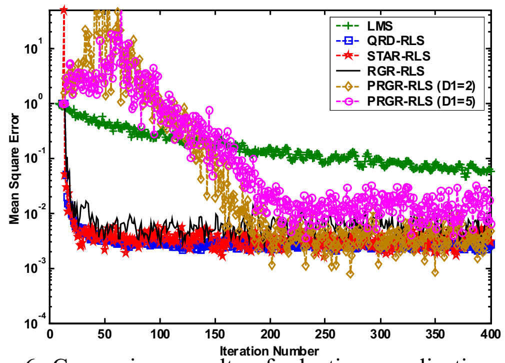


Fig. 6. Comparison results of adaptive equalization using LMS, QRD-RLS, STAR-RLS, RGR-RLS, and PRGR-RLS ($D_1 = 2$ and $D_1 = 5$) architectures for $\lambda = 0.9$.

Table I: Comparison Results among Different Architectures

	QRD-RLS	PSTAR-RLS	CORDIC-Based RLS	Our Work
Critical Period	$2T_m + T_{sq} + T_a$	$\frac{T_m + T_d + 2T_a + T_{sw}}{D_1}$	$\frac{\left(\text{Critical Period in the Word Level}\right)}{D_1}$	$\frac{3T_m + T_d + 2T_a + T_{sw}}{D_1}$
Orthogonality	Exact	Approximate	Exact	Approximate
Robustness for λ	Good	Diverges for small λ	Good	Good
Need for Memory	No	No	Yes	No
Square Root	Not Free	Free	Free	Free