Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 49(63), Fascicola 2, 2004

QRD-LSL Algorithm Suitable for Implementation on D.S.P.

Andrei Alexandru Enescu, Constantin Paleologu, Silviu Ciochină[1]

Abstract – This paper deals with modifications brought to the QRD-LSL algorithm presented in [5], in order to implement it on Digital Signal Processor (*DSP*). It is necessary to choose a powerful processor, with a parallel architecture that allows several instructions to be executed simultaneously. This is the main reason for choosing Motorola *SC140* processor. In addition, we have structured the algorithm in a way to allow a high complexity algorithm to be run in real-time in a specific application. The paper presents the main features of this *DSP* and then makes comparisons between the original algorithm, described in [1] and the improved version with low complexity.

Keywords: **adaptive lattice algorithm, least squares, digital signal processor**

I. INTRODUCTION

It has been shown in [2], [3] that *QRD-LSL* algorithm has great performances when used in echo canceller configuration. However, in the original version, the complexity is quite large, making the algorithm impossible to be implemented in real-time. In [1], a modified version of the algorithm is presented. A comparison between the two algorithms is presented in Table 1:

Table 1.

Adaptive algorithm	QRD-LSL	MQRD-LSL
multiplications	25M+11	22M+10
divisions	4M+2	4M+2
additions/ subtractions	8M+3	8M+3
square-root operations	4M+2	0

It is to be noticed that the square root operations require a large computing time, as they must be approximated by another technique, e. g. Taylor series. Either way, the computing time increases significantly and the application area becomes restricted due to a reduced sampling frequency. Thus, the main goal is to minimize the number of instructions within the implemented algorithm.

Another issue is the fixed-point representation. Since fractional representation is used (i.e. for a given number of bits, B, the range for any fractional variable is $[-1;1-2^{-B}]$), care must be taken when arithmetic operations are made, in order to avoid overflow. An overview on the dynamics of the algorithm variables will prove that scaling is needed in the implementation process, because some variables are greater than unit value.

Moreover, this paper presents the implementation „tricks" used to simplify the algorithm in order to implement it on a digital signal processor.

II. AN OVERVIEW OF SC140 ARCHITECTURE

As we have chosen to implement the algorithm on *SC140*, it is necessary to describe the features of this processor first. The specific features of this architecture, described in [6] are the following:

- High level abstraction of the Application Software
  - Applications development in C language
  - Hardware supported integer and fractional data
- Scalable performance
  - 4 *ALU*s (Arithmetic logic Units) and 2 *AAU*s (Address Arithmetic Units)
  - 4 *MMACS* (million multiply and accumulate operations per second) for each megahertz of clock frequency
- High Code Density for Minimized Cost
  - 16-bit wide instruction encoding

The core important features are:

- Up to 10 *RISC MIPS* for each megahertz of clock frequency
- A true (16*16) + 40→40-bit *MAC* unit in each *ALU*

[1] Facultatea de Electronică şi Telecomunicaţii, Catedra de Telecomunicaţii Bd. I. Maniu 1-3, Bucureşti, e-mail: {aenescu, pale, silviu}@comm.pub.ro

- A true 40-bit parallel barrel shifter in each *ALU*
- 16 x 40-bit data registers for fractional and integer data operand storage
- 16 x 32-bit address registers (8 can be used as 32-bit base address registers)
- 4 address offset registers and 4 modulo address registers
- Unified data and program memory space (Harvard architecture)
- Byte addressable data memory

However, the main feature that we have already mentioned is the C compiler and the ability to convert *C* source code into assembly code. The complexity of *QRD-LSL* algorithm is quite large and therefore the need for flexibility is important, since programming in *C* code is much easier than implementing the algorithm direct in assembly code. The *C* compiler supports *ANSI C* standard and also intrinsic functions for *ITU/ETSI* primitives. Assembly code integration is also possible, which optimizes supplementary the code.
The block diagram of *SC140* core, as presented in [2], is described in Fig. 1.

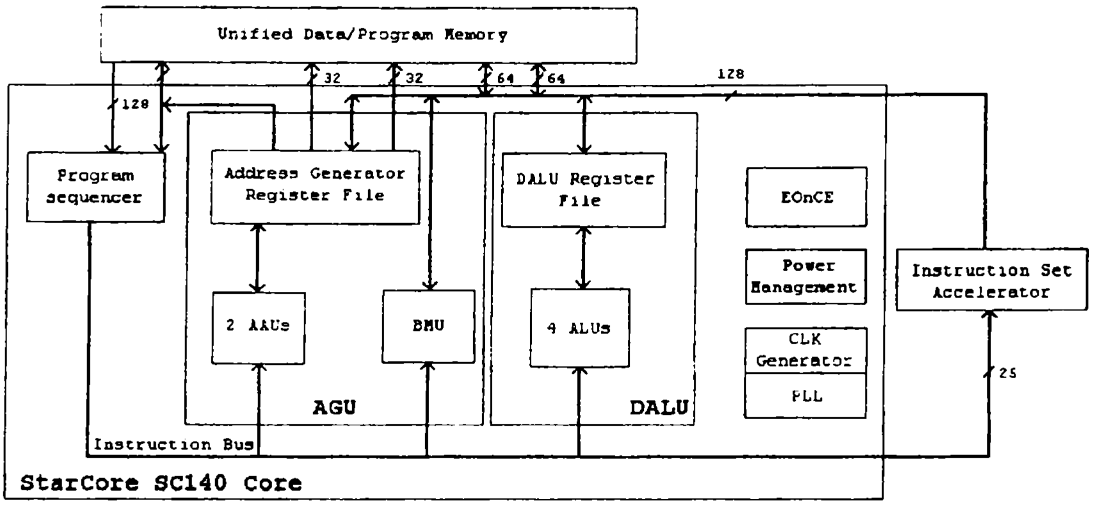

Fig 1. Block Diagram of SC140 Core

In Fig. 1, we present the SC140 core, including:

- *DALU* (Data Arithmetic Logic unit)
  - A register file of 16 x 40-bit registers
  - 4 parallel *ALU*s (each one containing a *MAC* unit and a *BFU*- bit-field unit)
  - 8 data bus shifter/limiters
- *AGU* (Address Generation Unit)
  - 2 *AAU*s, accessing 16 address registers, 4 offset registers, 4 modulo registers

III. AN ANALYZE OF *QRD-LSL* IN *DSP* IMPLEMENTATION CONTEXT

In [1]. a version of *QRD-LSL* algorithm is presented, after eliminating complex operations, such as square roots. The algorithm is presented in Table 2:

Table 2.

```
1. Initialization
For order $m = 1,2,\ldots,M$
        $\bar{\pi}_{f,m-1}(0) = \bar{\pi}_{b,m-1}(0) = 0$, $\bar{p}_m(0) = 0$
        $B_{m-1}(0) = \delta$, $F_{m-1}(0) = \delta$
        // $\delta$ is a small positive constant
end

2. Computations
For time $n$ compute
    $\bar{\varepsilon}_{f,0}(n) = \bar{\varepsilon}_{b,0}(n) = x(n)$
    // $x(n)$ is the input at time $n$
    $\bar{\varepsilon}_0(n) = d(n)$
    // $d(n)$ is the desired response at time n
    $\bar{\gamma}_0(n) = 1$
    $\beta_{b,0}(n-1) = 1$, $\beta_{f,0}(n-1) = 1$
    For order $m = 1,2,\ldots,M$
        $B_{m-1}(n-1) = \lambda B_{m-1}(n-2) +$
                $+ \beta_{b,m-1}(n-1)\left|\bar{\varepsilon}_{b,m-1}(n-1)\right|^2$
        $\bar{c}_{b,m-1}(n-1) = \dfrac{\lambda B_{m-1}(n-2)}{B_{m-1}(n-1)}$
        $\bar{s}_{b,m-1}(n-1) = \beta_{b,m-1}(n-1)\dfrac{\bar{\varepsilon}^{*}_{b,m-1}(n-1)}{B_{m-1}(n-1)}$
        $\bar{\varepsilon}_{f,m}(n) = \bar{\varepsilon}_{f,m-1}(n) - \bar{\varepsilon}_{b,m-1}(n-1)\bar{\pi}^{\bullet}_{f,m-1}(n-1)$
        $\bar{\pi}^{\bullet}_{f,m-1}(n) = \bar{c}_{b,m-1}(n-1)\vec{\pi}^{\bullet}_{f,m-1}(n-1) + \bar{s}_{b,m-1}(n-1)\bar{\varepsilon}_{f,m-1}(n)$
        $\bar{\gamma}_m(.. -1, = \bar{c}_{b,m-1}(n-1,\bar{\gamma}_{m-1}(n-1)$
        $F_{m-1}(n) = \lambda F_{m-1}(n-1) + \beta_{f,m-1}(n-1)\left|\bar{\varepsilon}_{f,m-1}(n)\right|^2$
        $\bar{c}_{f,m-1}(n) = \frac{\lambda F_{m-1}(n-1)}{F_{m-1}(n)}$
        $\bar{s}_{f,m-1}(n) = \beta_{f,m-1}(n-1)\frac{\vec{\varepsilon}^{\bullet}_{f,m-1}(n)}{F_{m-1}(n)}$
        $\bar{\varepsilon}_{b,m}(n) = \bar{\varepsilon}_{b,m-1}(n-1) - \bar{\varepsilon}_{f,m-1}(n)\bar{\pi}^{\bullet}_{b,m-1}(n-1)$
        $\bar{\pi}^{\bullet}_{b,m-1}\ n\ = \bar{c}_{f,m-1}(n\ \bar{\pi}^{\bullet}_{b,m-1}\ n-1\ +$
                $+ \bar{s}_{f,m-1}(n)\bar{\varepsilon}_{b,m-1}(n-1)$
        $\bar{\varepsilon}_{m+1}(n) = \bar{\varepsilon}_m(n) - \bar{\varepsilon}_{b,m}(n)\bar{p}^{\bullet}_m(n-1)$
        $\vec{p}^{\bullet}_m(n) = \bar{c}_{b,m-1}(n-1)\vec{p}^{\bullet}_m(n-1) + \bar{s}_{b,m-1}(n-1)\bar{\varepsilon}_m(n)$

        $\beta_{b,m}(n-1) = \bar{c}_{b,m-1}(n-1)\beta_{b,m-1}(n-1)$
        $\beta_{f,m}(n) = \bar{c}_{f,m-1}(n)\beta_{f,m-1}(n-1)$
    end
    $B_M(n) = \lambda B_M(n-1) + \beta_{b,M}(n)\left|\bar{\varepsilon}_{b,M}(n)\right|^2$
    $\bar{c}_{b,M}(n) = \frac{\lambda B_M(n-1)}{B_M(n)}$
    $\bar{s}_{b,M}(n) = \beta_{b,M}(n)\frac{\vec{\varepsilon}^{\bullet}_{b,M}(n)}{B_M(n)}$
    $\bar{\varepsilon}_{M+1}(n) = \bar{\varepsilon}_M(n) - \bar{\varepsilon}_{b,M}(n)\bar{p}^{\bullet}_M(n-1)$
    $\vec{p}^{\bullet}_M(n) = \bar{c}_{b,M}(n)\bar{p}^{\bullet}_M(n-1) + \bar{s}_{b,M}(n)\bar{\varepsilon}_M(n)$
    $\bar{\gamma}_{M+1}(n) = \bar{c}_{b,M}(n)\bar{\gamma}_M(n)$
    $e_{M+1}(n) = \bar{\gamma}_{M+1}(n)\bar{\varepsilon}_{M+1}(n)$
end
```

We focus on the echo cancelling configuration, since it is demonstrated in [5] that this algorithm proves good performances in double-talk configuration.

Because the standard input signals, the learning curve and other technical details can be found in [5], we only show the dynamic range of some variables during convergence process. Assuming a css_st standard signal at the far end and a sinusoid of normalized frequency 0.1 at the near end and considering an echo path with a length of the impulse response of 64, we present the evolution of the cost functions on both forward and backward prediction branches.

It can be easily seen that, during convergence, the samples increase to a very large value, much greater than 1. This observation is valid, especially for the first cell, as we can see from Fig. 2. Also it is to be noticed that a similar evolution have the samples for the forward prediction cost function.

At the price of losing from finite precision, some of the bits used in quantization can be used for scaling, regarded in binary arithmetic as a simply right-shifting by the same number of bits.

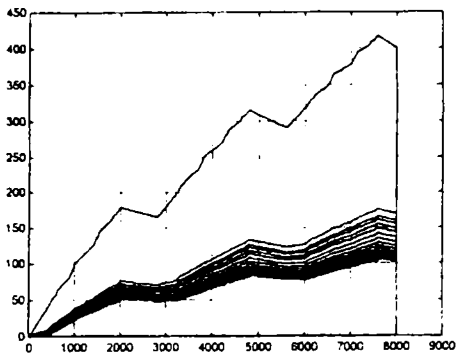

Fig. 2. Evolution of backward cost functions for the 64 cells of the structure

Then, the formula used for actualization of this cost function becomes:

$$\bar{B}_{m-1}(n-1) = \lambda\bar{B}_{m-1}(n-2) + \beta_{b,m-1}(n-1)\left|\bar{\varepsilon}_{b,m-1}(n-1)\right|^2 2^{-b} \qquad (1)$$

In (1):

$$\bar{\mathbf{B}}(n) = \mathbf{B}(n)2^{-b} \qquad (2)$$

Then, the variables that also depend on $B(n)$ are actualized as follows:

$$\bar{c}_{b,m-1}(n-1) = \frac{\lambda\bar{B}_{m-1}(n-2)}{\bar{B}_{m-1}(n-1)} \qquad (3)$$

$$\bar{s}_{b,m-1}(n-1) = \beta_{b,m-1}(n-1)\frac{\bar{\varepsilon}^{\bullet}_{b,m-1}(n-1)}{\bar{B}_{m-1}(n-1)}2^{-b} \qquad (4)$$

The same modifications stand also for the forward prediction part with the proper index replacements. If the number of bits used in scaling is properly chosen, then there is no overflow.

Even though an asymptotic limit for the cost functions has not yet been found, an upper bound for them can be deduced. Let us denote that:

$$l_m = \lim_{n\to\infty} B_m(n) \qquad (5)$$

From (1), assuming that the last term is smaller than 1 in a correct fractional representation, we get:

$$l_m \le \lambda l_m + 1$$

and therefore:

$$l_m \leq \frac{1}{1-\lambda} \qquad (6)$$

As shown in [3], a small residual error in an echo cancelling configuration is achieved with a *RLS* adaptive algorithm by setting the forgetting factor as closer to 1 as possible. All the same, if we set λ to the maximum possible represented number on short format of 16 bits, i.e. $1\text{-}2^{-15}$, then l_m is limited by 2^{15}. The number of bits used for scaling should be $log_2\ (l_m)=15$, which is unacceptable, since it is exactly the precision used for a short format variable. Thus, a trade-off is required between echo canceller's theoretical performances and the precision used for cost functions.
In order to test echo canceller's performances, the algorithm has been implemented using Code Warror *C* Compiler for *SC140*. A simulation on the evaluation board was run, with a sinusoid of a normalized frequency 0.05 as near-end signal and a scaling of 10 bits. The signals are described in Fig. 3:

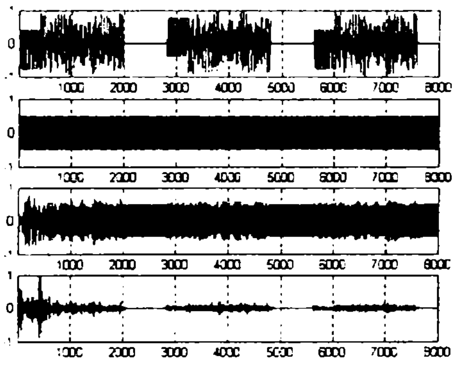

Fig. 3. Far-end signal. Near-end signal. Output signal. Residual error.

IV. OPTIMIZING TECHNIQUES USED FOR *QRD-LSL*

In this paragraph, we evaluate the number of cycles needed by the algorithm per iteration. The goal is to minimize this number, in order to lower the computational time per iteration under the sampling time of the *CODEC*.
If we take advantage of the fact that the structure is symmetrical, because of the similarities between the forward prediction structure and the backward prediction structure, then we can use two identical blocks for each lattice cell, thus we can call twice a function in *C* language during one iteration.
A behavioral description of the block is given in Table 3.

Table 3

```
$[J_{m-1}, \beta_m, \varepsilon_m, e_m, \pi_{m-1}, p_{m-1}] =$
$= prediction(J_{m-1}, \beta_{m-1}, \bar{\varepsilon}_{m-1}, \varepsilon_{m-1},$
        $e_{m-1}, \pi_{m-1}, p_{m-1})$
$aux = \lambda J_{m-1} + \beta_{m-1}\left|\bar{\varepsilon}_{m-1}\right|^2 2^{-b}$
$c_{m-1} = \frac{\lambda J_{m-1}}{aux}$
$J_{m-1} = aux$
$\beta_m = c_{m-1}\beta_{m-1}$
$s_{m-1} = 2^{-b}\frac{\beta_{m-1}\bar{\varepsilon}}{J_{m-1}}$
$\varepsilon_m = \varepsilon_{m-1} - \bar{\varepsilon}_{m-1}\pi_{m-1}$
$\pi_m = c_{m-1}\pi_{m-1} + s_{m-1}\varepsilon_{m-1}$
$if\ flag$
    $e_m = e_{m-1} - \bar{\varepsilon}_{m-1}p_{m-1}$
    $p_{m-1} = c_{m-1}p_{m-1} + s_{m-1}e_{m-1}$
$end$
```

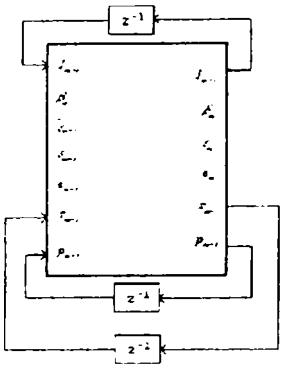

Fig. 5. Block diagram for one cell *prediction*

The filtering part is included in backward prediction part and is performed if a flag is set. This flag is set before the backward prediction and reset before the forward prediction. Then, iteration is described in Table 4.

Table 4.

```
$\gamma = 1 - \lambda$
$for\ m = 0, M$
    $flag = 1;$ // backward prediction
    $[B_{m-1}, \beta_{b,m}, \varepsilon_{f,m}, e_m, \pi_{f,m-1}, p_{m-1}] =$
    $= prediction(B_{m-1}, \beta_{b,m-1}, \bar{\varepsilon}_{m-1}, \varepsilon_{f,m-1},$
            $e_{m-1}, \pi_{f,m-1}, p_{m-1})$
    $flag = 0;$ // forward prediction
    $[F_{m-1}, \beta_{f,m}, \varepsilon_{b,m}, e_m, \pi_{b,m-1}, p_{m-1}] =$
    $= prediction(F_{m-1}, \beta_{f,m-1}, \varepsilon_{f,m-1}, \bar{\varepsilon}_{m-1},$
            $e_{m-1}, \pi_{b,m-1}, p_{m-1})$
    $\bar{\varepsilon} = \varepsilon_b$
```

end

Another optimization technique, accomplished using this procedure is that all the transformations are made in-place, regardless of the iteration (i.e. moment of time), saving a large amount of memory. Choosing an appropriate level of optimization from the *C* compiler, Code Warrior (0-3), makes further optimization. As well, the proper use of intrinsic functions from *C* compiler can further reduce the number of cycles.

A very good approximation on computing time per iteration shows that it is proportional to adaptive filter's order:

$$t_c \approx \alpha M \qquad (7)$$

We shall refer to α from now on as *proportionality constant*. During implementation on StarCore, the evolution of this constant was most relevant and it is described in Fig. 6.

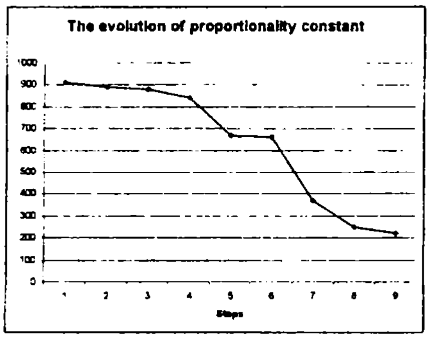

Fig. 6. Evolution of proportionality constant

In Fig. 6, on X axis, optimization steps are represented in time. The optimization process also included taking advantages of the parallel architecture and can be found in [2].

We describe the stages from Fig. 2:

1-3. Intrinsec optimizations from procedure *prediction* (use auxiliary value for *J*, rearrange the cos factor computation)

4. Modularization by using the procedure *prediction*

5-6. Levels 0-1 of optimization

7. Use in-place transformation

8-9. Levels 2-3 of optimization

V. REMARKS

A modified version of *QRD-LSL* algorithm, suitable for implementation on *DSP* has been presented, along with the description of the architecture of the StarCore, the *DSP* used for implementation. Because this algorithm is known to be complex and because it needs to run in real time, it is necessary to furthermore reduce the number of cycles. The next step is to write special assembly routines within the program, replacing the complex operations.

REFERENCES

[1] C. Paleologu, S. Ciochina, A. A. Enescu, "Modified versions of QRD-LSL Adaptive Algorithm with Lower Computational Complexity", Rev. Roum. Sci. Techn. - Electrotechn. et Energ., vol. 46, no.3, 2001.

[2] C. Paleologu, S. Ciochină, A.A. Enescu, „A Network Echo Canceller Based on a SRF QRD-LSL Adaptive Algorithm Implemented on Motorola StarCore SC140 DSP", IEEE Int. Conf. ICT 2004, Fortaleza, Brasil, 2004

[3] S. Ciochina, C. Paleologu, "On the Performances of QRD-LSL Adaptive Algorithm in Echo Cancelling Configuration", Proc. IEEE ICT 2001, Bucharest, Romania, vol.1, 2001, pp. 563-567.

[4] S. Haykin, *Adaptive Filter Theory*, Third Edition, Prentice Hall International, Inc. Englewood Cliffs, 1996.

[5] C. Paleologu, S. Ciochina, A. Enescu, "A Simplified QRD-LSL Algorithm in Echo Cancelling Configuration", Proc. IEEE ICT 2002, Beijing, China, vol.1, 2001, pp. 563-567.

[6] "SC140 DSP Core Reference Manual", Revised 1, 6/2000

[7] St. Gay, "An Efficient, Fast Converging Adaptive Filter for Network Echo Cancellation", Proc. Asilomar, Pacific Grove, CA, Nov. 1998, pp 394-398.

[8] Ph. Regalia, "Numerical Stability Properties of a QR-Based Fast Least Squares Algorithm", IEEE Trans. Signal Processing, vol. 41, no. 6, June 1993, pp 2096-2109.

[9] M. Harteneck, R. W. Stewart, J. G. McWhirter, I.K. Proudler, "Algorithmic Engineering Applied to the QR-RLS Adaptive Algorithms", Proc. 4th International Conference on Math. Signal Proc., Warwick, U.K. 1996.

[10] Regalia P., Bellanger G. "On the Duality Between Fast QR Methods and Lattice Methods in Least Squares Adaptive Filtering", IEEE Trans. Signal Processing, vol. 39, no. 8, April 1991, pp. 879-891.

[11] Ciochină S., Negrescu C., *Adaptive Systems*, Ed. Tehnică, Bucharest, 1999.

[12] Liu J. "A Novel Adaptation Scheme in the NLMS Algorithm for Echo Cancellation", IEEE Signal Processing Letters, vol. 8, no. 1, January 2001, pp. 20-22.

[13] J.G. Proakis, C. M. Rader, F. Ling, C. L. Nikias, *Advanced Digital Signal Processing Algorithms*, Macmillan Publishing Company, 1992.

[14] W.M. Gentleman, "Least Squares Computations by Givens Transformations without Square-Roots", J. Inst. Math. Its Appl., vol. 12, 1973, pp. 329-336.

[15] ITU-T Recommendation G.168, Digital Network Echo Cancellers, 2000, Draft 3.

[16] ITU-T Recommendation G.711, pulse code modulation (PCM) of voice frequencies, CCITT-Blue Book, Volume III, Fasc. III. 4, pp. 175-184.

[17] A. Andronache, C. Anghel, S. Pop, "A Novel Adaptation Scheme in the NLMS Algorithm for Digital Network Echo Canceller Implemented on Motorola StarCore SC140", Int. Conference COMM 2002, Dec. 2002, Bucharest, Romania