Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2008, Article ID 394201, 11 pages
doi:10.1155/2008/394201

Research Article

Implementation of the Least-Squares Lattice with Order and Forgetting Factor Estimation for FPGA

Zdenek Pohl, Milan Tichy, and Jiri Kadlec

Institute of Information Theory and Automation, Pod Vodarenskou vezi 4, 182 08 Prague, Czech Republic

Correspondence should be addressed to Milan Tichy, tichy@utia.cas.cz

Received 6 February 2008; Revised 5 June 2008; Accepted 24 June 2008

Recommended by Ricardo Merched

A high performance RLS lattice filter with the estimation of an unknown order and forgetting factor of identified system was developed and implemented as a PCORE coprocessor for Xilinx EDK. The coprocessor implemented in FPGA hardware can fully exploit parallelisms in the algorithm and remove load from a microprocessor. The EDK integration allows effective programming and debugging of hardware accelerated DSP applications. The RLS lattice core extended by the order and forgetting factor estimation was implemented using the logarithmic numbers system (LNS) arithmetic. An optimal mapping of the RLS lattice onto the LNS arithmetic units found by the cyclic scheduling was used. The schedule allows us to run four independent filters in parallel on one arithmetic macro set. The coprocessor containing the RLS lattice core is highly configurable. It allows to exploit the modular structure of the RLS lattice filter and construct the pipelined serial connection of filters for even higher performance. It also allows to run independent parallel filters on the same input with different forgetting factors in order to estimate which order and exponential forgetting factor better describe the observed data. The FPGA coprocessor implementation presented in the paper is able to evaluate the RLS lattice filter of order 504 at 12 kHz input data sampling rate. For the filter of order up to 20, the probability of order and forgetting factor hypotheses can be continually estimated. It has been demonstrated that the implemented coprocessor accelerates the Microblaze solution up to 20 times. It has also been shown that the coprocessor performs up to 2.5 times faster than highly optimized solution using 50 MIPS SHARC DSP processor, while the Microblaze is capable of performing another tasks concurrently.

1. INTRODUCTION

A number of possible applicationsin *digital signal processing (DSP)* such as parameter estimation [1], echo suppression [2], or beam-forming [3] can be found for adaptive least squares filters. Their recursive form known as the *recursive least squares (RLS)* [4] is the solution of the minimum mean square error problem. The convergence rate of the RLS is far superior to that of the well-known *least mean square (LMS)* [5] algorithm and its normalized sibling NLMS.

The hardware implementation of the RLS algorithm is rather difficult due to its high computational complexity and problems with numerical stability. The complexity can be decreased by the exploitation of serial structure of the input data, typical for DSP applications. It allows to reduce asymptotic complexity of the RLS from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. One of the *fast* versions of RLS algorithm is represented by the *fast transversal filters (FTF)* [6–8]. The problems with numerical instability of the FTF lead to the development of its stabilized version [9, 10]. Motivated to develop numerically stable *fast* RLS filters, the *least-squares lattice (LSL)* filters [11, 12] and *fast QR decomposition (QR-RLS)* [13] algorithms were developed. While the numerical stability of the *fast* QR-RLS algorithm was proven analytically [14], good numerical properties of an LSL filter were found experimentally. The analysis of QR-RLS and LSL algorithms can be found in [15]. This work focuses on the LSL algorithm because of its slightly lower complexity.

The implementation of the LSL filter with error-feedback [16–18] has proven good numerical behavior. In [16], it was shown that the filter can be efficiently implemented in *field programmable gate arrays (FPGA)*. In the following text, this algorithm is referred to as the *RLS lattice*. Its computational complexity is $24N$, which can be further reduced by the utilization of parallel hardware in an FPGA.

The possibilities of efficient FPGA implementation of the RLS lattice algorithm were exhaustively investigated in [19, 20]. Similar approach to the optimization of other DSP

algorithms for FPGAs can be found in [21–23]. As shown in these works, the resulting *intellectual property (IP)* cores can outperform floating-point DSP microprocessor solutions by one order of magnitude. There also exist other RLS filter FPGA implementations such as [1, 3], where IP cores are implemented and integrated in a custom one purpose design. Another implementation of RLS filter is presented in [24], where calculations are distributed between FPGA and the NIOS microprocessor in a single chip.

Our aim is to provide a versatile highly configurable hardware RLS core for DSP applications. We focus on the implementation of a hardware coprocessor rather than a standalone IP core. Our target application scheme is to use one lattice filter and to estimate the order probability or to use more parallel lattice filters with different forgetting factors on a single channel and to estimate, which forgetting factor yields better results. We also expect that parallel lattice filters will be possible to interconnect serially and to create a high performance pipelined solution. The algorithm is integrated into the Xilinx EDK environment as a Microblaze accelerator, resulting in a versatile, easy-to-use and compact RLS lattice coprocessor, which can easily be accessed by the standard C programming and debugging.

2. PROBABILISTIC APPROACH TO SYSTEM IDENTIFICATION

In order to outline the development of the RLS lattice algorithm with order probability estimation and to describe its hardware implementation, a brief insight to the recursive least squares estimation from the probabilistic theory viewpoint is provided.

The probabilistic approach [25] to the system identification provides the link between the probabilistic theory and the least square error estimation, which allows us to extend the estimation task by the hypotheses probability estimation. In this approach to the system identification, the hypotheses probability update of an autonomous one-dimensional system [26] can be formulated as

$$p(h_n|\mathcal{D}^n) = \frac{p(y_n|\mathcal{D}^{n-1},h_n)p(h_n|\mathcal{D}^{n-1})}{\sum_{\forall h} p(y_n|\mathcal{D}^{n-1},h_n)p(h_n|\mathcal{D}^{n-1})}, \quad (1)$$

where y_n are data observed at the unknown system output at time n, the variables $\mathcal{D}^n$ and $\mathcal{D}^{n-1}$ are previously observed data y_0 through y_n and y_{n-1}, respectively. The hypothesis h_n is the ordered pair $h \in (i,\lambda)$, where $i \in \{0,\ldots,N\}$ is the unknown order and $\lambda \in \{\lambda_1,\ldots,\lambda_M\}$ is the unknown forgetting factor. The term $p(y_n|\mathcal{D}^{n-1},h_n)$ is the probabilistic description of the modeled system with order and exponential forgetting given by hypothesis h_n. This probability can be evaluated as

$$p(y_n|\mathcal{D}^{n-1},h_n) = \pi^{-1/2}\frac{\lambda\Lambda_{h,n-1}^{((\lambda\vartheta_{n-1}-i)/2+1)}}{\Lambda_{h,n}^{((\vartheta_n-i)/2+1)}}\cdot\frac{\det(\lambda\mathbb{V}_{z,h,n-1})^{1/2}}{\det(\mathbb{V}_{z,h,n})^{1/2}}\cdot\frac{\Gamma((\vartheta_n-i)/2+1)}{\Gamma((\lambda\vartheta_{n-1}-i)/2+1)}, \quad (2)$$

where Λ_h is the optimal solution error of the model for the hypothesis h. The matrix $\mathbb{V}_{z,h,n}$ is the autocorrelation matrix defined as

$$\mathbb{V}_{z,h,n} = \mathbf{z}_{h,n}\mathbf{z}_{h,n}^T, \quad (3)$$

where

$$\mathbf{z}_{h,n} = [y_{n-1}\cdots y_{n-i}]^T. \quad (4)$$

The operator Γ in (2) is the gamma function. The quantity ϑ represents the "amount of data" accumulated in matrix $\mathbb{V}_{z,h,n}$ through the estimation process. The quantity ϑ is updated as

$$\vartheta_n = \lambda\vartheta_{n-1} + 1. \quad (5)$$

It should be noted that according to [25] the relation (2) is correct only for the definition of the autoregression model inherent to the hypothesis h as a conditional probability density function (pdf)

$$p(y_n|\mathcal{D}^{n-1},\Theta_{h,n},h_n) = \frac{\omega_{h,n}}{\sqrt{2\pi}}e^{-(\omega_{h,n}/2)(y_n-\mathbf{A}_{h,n}^T\mathbf{z}_{h,n})^2}, \quad (6)$$

where the symbol $\Theta_{h,n}$ denotes parametrization of the model by the vector of unknown parameters α and of an unknown *precision measure* ω [25]

$$\Theta_{h,n} = \{\mathbf{A}_{h,n},\omega_{h,n}\},\quad \mathbf{A}_{h,n}^T = [\alpha_1\ \cdots\ \alpha_i]. \quad (7)$$

The autoregression model of an unknown system in (6) can be described as

$$y_n = \mathbf{A}_{h,n}^T\mathbf{z}_{h,n} + e_{h,n}, \quad (8)$$

where $e_{h,n}$ is the prediction error of the model inherent to the hypothesis h at time n. As shown in [25], if $e_{h,n}$ is a normally distributed random variable, the conditional pdf for the model parameters $\Theta_{h,n}$ can be written as

$$p(\Theta_{h,n}|\mathcal{D}^n,h_n) = c_{h,n}\omega_{h,n}^{\vartheta_n/2}e^{-(\omega_{h,n}/2)X_{h,n}}, \quad (9)$$

where $c_{h,n}$ is a normalizing constant and

$$X_{h,n} = \begin{bmatrix}-1\\ \mathbf{A}_{h,n}\end{bmatrix}^T\mathbb{V}_{h,n}\begin{bmatrix}-1\\ \mathbf{A}_{h,n}\cdot\end{bmatrix}. \quad (10)$$

The matrix $\mathbb{V}_{h,n}$ is the augmented autocorrelation matrix which keeps information about the shape of the conditional pdf (9). This data matrix can be updated recursively as

$$\mathbb{V}_{h,n} = \lambda\mathbb{V}_{h,n-1} + \lambda\begin{bmatrix}y_n\\ \mathbf{z}_{h,n}\end{bmatrix}\begin{bmatrix}y_n & \mathbf{z}_{h,n}^T\end{bmatrix}. \quad (11)$$

The optimal solution $\hat{\mathbf{A}}_{h,n}$ for $\mathbf{A}_{h,n}$ is located at the maximum of the conditional pdf (9). The maximum can be found by the minimization of the quadratic form (10), which can be written as

$$X_{h,n} = \Lambda_{h,n} + (\mathbf{A}_{h,n} - \hat{\mathbf{A}}_{h,n})^T\mathbb{V}_{z,h,n}(\mathbf{A}_{h,n} - \hat{\mathbf{A}}_{h,n}). \quad (12)$$

For better clarity, the subscript n will be omitted in the following text. The matrix $\mathbb{V}_h$ used in (10) can be decomposed as follows:

$$\mathbb{V}_h = \begin{bmatrix} V_y & \mathbf{V}_{z,y,h}^T \\ \mathbf{V}_{z,y,h} & \mathbb{V}_{z,h} \end{bmatrix}. \tag{13}$$

Consequently, the optimal solution can be written as

$$\begin{aligned} \hat{\mathbf{A}}_h &= \mathbb{V}_{z,h}^{-1}\mathbf{V}_{z,y,h}, \\ \omega_h &= \vartheta\Lambda_h^{-1}, \end{aligned} \tag{14}$$

where

$$\Lambda_h = V_y - \mathbf{V}_{z,y,h}^T\mathbb{V}_{z,h}^{-1}\mathbf{V}_{z,y,h}. \tag{15}$$

Recursive formulas for the direct update of the decomposed matrix $\mathbb{V}_h$ can be derived from (11).

The recursive solution for $\hat{\mathbf{A}}_h$, maximizing the pdf given by (9), is also known as the RLS algorithm [27]. It is important to note that for the estimation of $(N+1)M$ hypotheses by the Bayes formula (1), it is needed to estimate all models defined by the estimated hypotheses, which means to calculate NM-array of RLS filters.

3. HYPOTHESES ESTIMATION

Despite the possibility to implement the RLS estimation by formulas introduced in Section 2, it is more convenient to use one of the state-of-the-art RLS algorithms. As the most convenient algorithm for implementation, the recursive least-squares lattice [4] in the error-feedback form was chosen. As suggested in [17], the normalized a posteriori errors are used to reduce the complexity of the algorithm. The computational complexity of this algorithm is $24N$, where N denotes the filter order (dimension).

As mentioned above, the estimation of probability of hypotheses h by (1) requires performing one RLS estimation for each hypothesis h. Thus, the NM-array of RLS filters has to be calculated.

The most important property making the RLS lattice suitable for the hypotheses estimation is its modular structure. The RLS lattice filter consists of a cascade of identical modules. Each module implements the *order update*, which means that it is using ith order output from the preceding module and increases the order of estimation to $i+1$. Consequently, estimations of all orders up to N can be found during computations. Using this principle, the number RLS filters required for the hypotheses estimation can be reduced to M.

In our solution, the evaluation of probability estimates is divided into two stages. The first stage performs the order update, which uses the "old" probability estimates and updates them by new data. This operation is represented by the numerator of (1). In the second stage, the *normalization* of the updated order estimates is performed. The normalization is represented by the denominator of (1). The forgetting on hypotheses pdf is applied in the normalization stage. The order update can be integrated into the RLS lattice algorithm.

Initialization
$\boldsymbol{\gamma}_{-1} = \mathbf{1}, \mathbf{F}_{-1} = \mathbf{B}_{-1} = \delta\mathbf{I}$
$\boldsymbol{\psi}_{-1} = \boldsymbol{\gamma}_{-1}^f = \boldsymbol{\gamma}_{-1}^b = \boldsymbol{\kappa}_{-1} = \mathbf{b}_{-1}^a = \mathbf{0}$
$\mathbf{e1}$, $\mathbf{e2}$, $\mathbf{g}$ set using (17) and (18)
Lattice update (for each $n \geq 0$):
$\alpha_{0,n} = d_n, \eta_{0,n} = \psi_{0,n} = u_n, \gamma_{0,n} = 1$
$p_{-1,\lambda,n}^{sd} = (N+1)\varphi$
for $(i = 0; i \leq N; i = i+1)$

$\eta_{i+1,n}$	$= \eta_{i,n} - \gamma_{i+1,n-1}^f \psi_{i,n-1}$	T1,2
f	$= \gamma_{i,n-1}\eta_{i,n}$	T3
$\psi_{i+1,n}$	$= \psi_{i,n-1} - \gamma_{i+1,n-1}^b \eta_{i,n}$	T5,4
b	$= \gamma_{i,n}\psi_{i,n}$	T6
$\alpha_{i+1,n}$	$= \alpha_{i,n} - \kappa_{i+1,n-1}\psi_{i,n}$	T8,7
$\gamma_{i+1,n}^f$	$= \gamma_{i+1,n-1}^f + b_{i,n-1}^a \eta_{i+1,n}$	T10,9
$F_{i,n}$	$= \nu + \lambda F_{i,n-1} + f\eta_{i,n}$	T13,11,14,12
$B_{i,n}$	$= \nu + \lambda B_{i,n-1} + b\psi_{i,n}$	T17,15,18,16
f_i^a	$= f/F_{i,n}$	T19
$b_{i,n}^a$	$= b/B_{i,n}$	T20
$\gamma_{i+1,n}^b$	$= \gamma_{i+1,n-1}^b + f_i^a \psi_{i+1,n}$	T22,21
$\kappa_{i+1,n}$	$= \kappa_{i+1,n-1} + b_{i,n}^a \alpha_{i+1,n}$	T24,23
$\gamma_{i+1,n}$	$= \gamma_{i,n} - b_{i,n}^a b$	T26,25
a_1	$= \lambda F_{i,n-1}$	U1
a_2	$= a_1^{\mathbf{e1}_i}/F_{i,n}^{\mathbf{e2}_i}$	U2
a_3	$= \lvert\gamma_{i,n}\rvert^{1/2}$	U3
$p_{i,\lambda,n}^d$	$= p_{i,\lambda,n-1}^d \cdot a_2 \cdot a_3 \cdot \mathbf{g}_i$	U4,5,6
$p_{i,\lambda,n}^{sd}$	$= p_{i-1,\lambda,n}^{sd} + p_{i,\lambda,n}^d$	U7

end

Algorithm 1: RLS lattice algorithm with exponential forgetting and probability update evaluation. The labels `Txx` and `Ux` denote individual arithmetic operations contained in the right side of each equation.

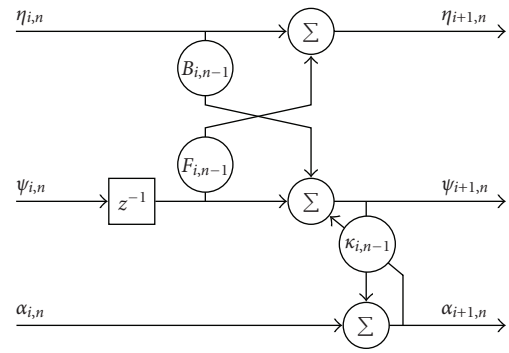

Figure 1: Data flow graph of one order update step of the algorithm.

The RLS lattice algorithm with order and forgetting factor update is summarized in Algorithm 1 . For the illustration, the update of estimates to order $i+1$ from order i is depicted in Figure 1.

The algorithm presented in Algorithm 1 is in the form, where input u_n is used to estimate desired value d_n. For identification of one-dimensional autoregression model the input must be connected as presented in Figure 2. Then, the

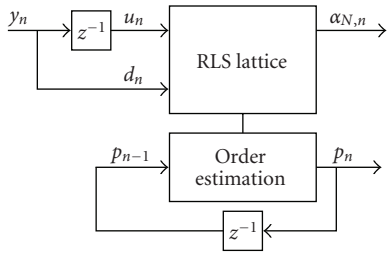

Figure 2: RLS lattice algorithm for identification of one-dimensional autoregression model.

Table 1: The parameters of the RLS lattice algorithm with exponential forgetting and probability update evaluation.

Parameters	
N	Filter order
$\lambda \in (0;1\rangle$	Forgetting factor
$\delta \quad \delta > 0, \delta \to 0$	Regularization parameter
$\nu \quad 2^{-b}(1-\lambda)$	Regularization constant
$\varphi \in \langle 0;1)$	Hypotheses forgetting factor

probabilistic approach given in Section 2 can be used for estimation of order and forgetting factor probability as also shown in the figure. The RLS lattice algorithm parameters are summarized in Table 1.

For the probability $p(h_n|\mathcal{D}^n)$, $p_{h,n} = p_{i,\lambda,n}$ will be further used as a more simple notation, where h was defined as the ordered pair (i,λ), $i \in \{0,\ldots,N\}$, $h \in \{1,\ldots,M\}$. Before the first iteration of the algorithm, initial hypotheses pdf has to be set and the look-up tables have to be initialized. The initial hypotheses pdf can be selected as

$$p_{i,\lambda,-1} = \frac{1}{(N+1)M} \quad \forall i,\lambda, \tag{16}$$

where $p_{i,\lambda,n}$ is the probability of order i and forgetting factor λ at time n. Value $n = -1$ represents the initialization step. The look-up tables are initialized for the limit value of

$$\vartheta_{\lim} = \lim_{n\to\infty} \vartheta_n = \frac{1}{1-\lambda} \tag{17}$$

as follows:

$$\begin{aligned} \mathbf{e1}_i &= \frac{\lambda\vartheta_{\lim} - i}{2} + 1 \\ \mathbf{e2}_i &= \frac{\vartheta_{\lim} - i}{2} + 1 \qquad i = 0,\ldots,N. \\ \mathbf{g}_i &= \pi^{-1/2}\frac{\Gamma(\mathbf{e2}_i)}{\Gamma(\mathbf{e1}_i)} \end{aligned} \tag{18}$$

After the values $p_{i,\lambda,n-1}$ have been updated, the normalization step has to be performed to obtain actualized probability $p_{i,\lambda,n}$. The pdf given by (1) extended with forgetting of hypotheses can be evaluated as

$$p_{i,\lambda} = \frac{p^d_{i,\lambda} + \varphi}{\sum_{\lambda=\lambda_1}^{\lambda_M}\sum_{i=0}^{N}(p^d_{i,\lambda} + \varphi)}, \tag{19}$$

where $p^d_{i,\lambda}$ is updated, but not normalized probability of order i and forgetting factor λ. The symbol φ is the forgetting factor of hypotheses.

Equation (19) can be calculated more efficiently as

$$p_{i,\lambda} = \frac{p^d_{i,\lambda} + \varphi}{\sum_{\lambda=\lambda_1}^{\lambda_M} p^{sd}_{N,\lambda}}, \tag{20}$$

where $p^{sd}_{N,\lambda}$ is a sum of updated probabilities $p^d_{i,\lambda}$ biased by the forgetting factor of hypotheses. Comparing (19) and (20), it can be shown that the sum of updated probabilities $p^d_{i,\lambda}$ can be calculated as

$$p^{sd}_{N,\lambda} = \sum_{i=0}^{N}(p^d_{i,\lambda} + \varphi) = (N+1)\varphi + \sum_{i=0}^{N} p^d_{i,\lambda}. \tag{21}$$

Then, the value of $p^{sd}_{i,\lambda}$ is calculated using the update of $p^{sd}_{i,\lambda}$ from its initial value $p^{sd}_{-1,\lambda} = (N+1)\varphi$ as shown in the right-hand side of (21). This step is labeled as operation `U7` in Algorithm 1.

Adding the order and forgetting estimation, the original RLS lattice algorithm increases its complexity to $31N$. Thus, the number of operations for maintaining $N+1$ order and M forgetting factor hypotheses is $31NM$. The normalization of updated probabilities requires $2M(N+1)+M-1$ operations. Considering these figures, we can state the complexity of the RLS lattice with the estimation of hypotheses which is $33NM + 3M - 1$ operations, provided that the division of two powers is regarded as one operation.

It is evident that the implementation of M RLS lattice estimations to test each hypothesis can be easily parallelized. For each forgetting factor hypothesis, one RLS lattice instance extended by the probability update can be evaluated in parallel. When all filters have been calculated, the normalization is performed before new data are acquired. Such an arrangement can be efficiently implemented in FPGAs.

4. FPGA IMPLEMENTATION

Hardware implementation of the RLS lattice requires ALU providing ADD, SUB, MUL, and DIV operations. For the probability estimation, POW and SQRT operations are also required. The *Logarithmic Number System (LNS)* arithmetic [28, 29] has been identified as the most convenient alternative to floating point for hardware implementation. This selection is supported by [16, 19].

Numbers in LNS are represented as fixed-point base-2 logarithms of numbers to be represented. The LNS arithmetic provides extremely effective MUL, DIV, and SQRT operations. The ADD/SUB operations are more complex and thus require more resources. For our RLS lattice implementation, the *high speed logarithmic arithmetic (HSLA)* library was used [29].

The proposed hardware provides solution to a few implementation challenges, such as

(i) conversions between fixed-point and LNS numbers;

(ii) implementation of the power function or directly of the division of powers;

(iii) mapping the algorithm to the LNS ALU efficiently and scheduling of operations;

(iv) ensuring the numerically robust behavior;

(v) implementation of the optimized RLS lattice core;

(vi) supporting integration of the core into a Microprocessor system.

In the following sections, these issues are directly addressed and their solution is presented.

4.1. Conversions

In audio DSP applications, the 16-bit two's complement integer is a typical input and output data format. The same precision was used for implementation of the input and output of the RLS lattice filter.

A conversion of an unsigned 16-bit fixed-point numbers to LNS format and vice versa, introduced in [19], was implemented. The method can be easily modified for signed integers. The integer to LNS conversion is based on the LNS addition, which can be written as

$$\log_2(X+Y) = i + \log_2(1+2^{j-i}), \tag{22}$$

where $i = \log_2|X|$ and $j = \log_2|Y|$ are the fixed-point numbers representing X and Y in LNS, respectively. The 16-bit integer input can be written as

$$Z = 2^8 z_1 + z_2, \tag{23}$$

where z_1 and z_2 are high and low parts of the integer Z, respectively. Then, the conversion of Z into the LNS can be written as

$$\log_2 Z = \log_2(2^8 z_1 + z_2) = \log_2(X+Y). \tag{24}$$

It is clear that for calculation of the LNS image of Z, the values $i = \log_2|X| = \log_2|2^8 z_1|$ and $j = \log_2|z_2|$ have to be known. The value of i and j can be tabulated as T_i and T_j, each of depth 256.

The conversion from LNS to fixed point is implemented as the binary search of the nearest lower number in T_i. The value of T_i is then subtracted from the original in the LNS domain and the search continues in T_j. The integer result is formed from addresses of found values in the tables T_i and T_j. The hardware solution of conversions using LNS addition and two tables delivers maximal conversion performance for the input and output data.

Initialization of the algorithm presented in Algorithm 1 requires constants and initial vector values in LNS. A software conversions for an FPGA soft processor were implemented using the functions provided in HSLA.

4.2. Division of powers

As mentioned in Section 2, the division of two general powers in (2) is considered as one floating-point-like operation. In the LNS arithmetic, this operation can be calculated as

$$Z = \log_2 \frac{A^{\mathbf{e1}_i}}{B^{\mathbf{e2}_i}} = a \cdot \mathbf{e1}_i - b \cdot \mathbf{e2}_i, \tag{25}$$

where $a = \log_2|A|$ and $b = \log_2|B|$ are LNS representations of A and B, respectively. The values of **e1** and **e2** are fixed-point exponent values stored in tables defined in (18).

According to (25), the division of two powers in the LNS can be implemented very efficiently as the fixed-point multiplications, which are in fact fixed-point integers representing base-2 logarithm with a fixed-point exponent. Consequently, the results are subtracted. The FPGA implementation benefits from the possibility to use integer subtraction in full precision and to truncate the result to the desired LNS width at the end.

The fixed-point exponents in tables **e1** and **e2** are stored in 16-bit unsigned fixed-point with 8 fractional bits. Such decision makes possible efficient implementation without the loss of precision. Although such representation limits the possible range of the parameter λ to $\lambda \in \langle 0.95; 0.995\rangle$ and of the parameter i to $i \in \{0; 20\}$, this range covers the most used options for the recursive identification and for the order estimation. The restriction put on the order i corresponds with the range in which the probabilistic approach to order estimation can provide reliable results as shown in [30].

4.3. ALU and scheduling

Since our implementation of the RLS lattice filter is designed for 16-bit two's complement integers used as an input and output, the 19-bit LNS arithmetic provides sufficient precision. The bit allocation within the 19-bit LNS number is as follows: 1 bit sign and 18-bit two's complement fixed-point number. Special values are reserved for zero, NaN, and Inf.

In order to exploit the maximal possible parallelism in the implementation of the RLS lattice filter, the cyclic scheduling of the RLS lattice inner loop was used [20, 31]. It was found that the addition hardware macro is utilized by less than 25%. For higher utilization of resources, four RLS lattice filter modules can share one dual-port ADD/SUB unit, that is, four independent RLS lattice filters can be implemented without using more ADD/SUB units. Other hardware macros used in the RLS lattice filter implementation are four MUL and DIV macros, one SQRT, and POW/POW macro. All hardware macros are fully pipelined.

Using the method of cyclic scheduling, four independent RLS lattice filter modules were mapped onto the LNS arithmetic units so that the schedule consists of 2 clock cycles prologue loop, 25 clock cycles main body loop, and 34 clock cycles epilogue loop. The resulting schedule is depicted in Figure 3, where the evaluation of one iteration of the RLS lattice filter is displayed. Operations for one iteration of the algorithm presented in Algorithm 1 are spread over three consecutive iterations in the hardware implementation and evaluated concurrently. Operations in Algorithm 1 were labeled `T1-26` for the RLS lattice algorithm and `U1-7` for the probability update. In Figure 3, the corresponding operations are displayed as 4-clock-cycle long operations, which consist of four consecutive calls—one for each instance of the RLS lattice module.

4.4. Numerical behavior

To guarantee the numerical stability, it is required to avoid divisions close to zero in operations labeled `T19` and `T20`.

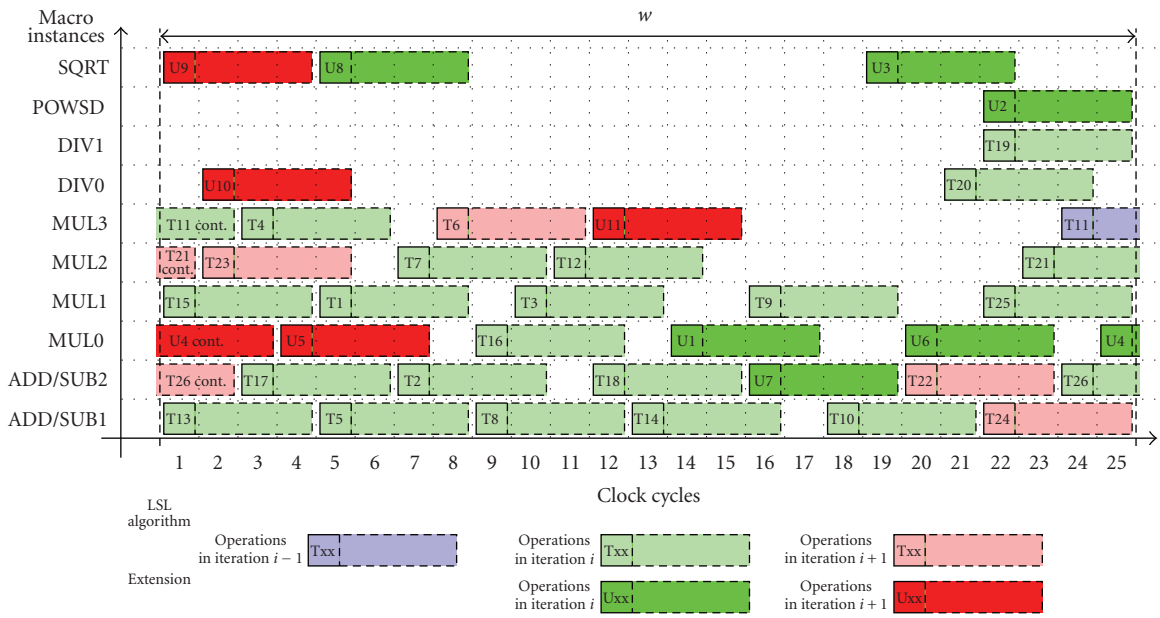

FIGURE 3: Schedule of operations of four RLS lattice algorithms with probability estimation. The operations `Txx` and `Ux` are defined in Algorithm 1.

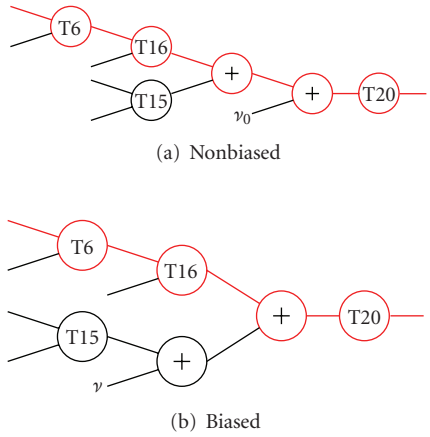

FIGURE 4: Comparison of two approaches to regularization of division in `T20`. The critical path fragment is marked in red.

The sufficient solution is to add a small positive constant ν_0 to the denominator before the divisions are evaluated [18]. Such solution provides *nonbiased* energies $B_{i,n}$ and $F_{i,n}$.

Alternatively, the constant $\nu = \nu_0(1 - \lambda)$ can be used in operations labeled `T13` and `T17` [17]. We use this version of the RLS lattice which introduces a small bias to the energies $B_{i,n}$ and $F_{i,n}$.

The "biased" version is more suitable for parallelization, rather than using the nonbiased version, because the evaluation of the energy $B_{i,n}$ lies on the critical path of the algorithm. Situation on the critical path is compared to both options in Figure 4. It can be seen that at the expense of a small bias added to the energies $F_{i,n}$ and $B_{i,n}$, the iteration time of the lattice loop is decreased from 33 to 25 clock cycles, that is, by 24%.

According to [17, 18], the constant ν equal to $2^{-b}(1 - \lambda)$ should be used. Here, b is the mantissa length to which the prediction energies are quantized and λ is the forgetting factor. For the 19-bit LNS arithmetic case, $b = 12$ was used. The worst case forgetting factor $\lambda = 0.95$ is determined by constrain introduced in Section 4.2. The relation for ν is correct under the assumption that input signal y_n is within the range $\langle -1, 1\rangle$, which is always satisfied by the input and output conversions described in Section 4.1.

4.5. Optimized RLS lattice core

The result of RLS lattice implementation is a standalone IP core. The internal variables of the core are expected to be stored in external memories. It allows more convenient integration of the core into the soft core processor.

A special memory organization, grouping the instances of one internal variable into one distributed memory element, was used. It allows us to reduce the size of multiplexer connected to the shared arithmetic unit macros, which is demonstrated in Figure 5. The experiments show that 8% of FPGA resources was saved by this approach.

4.6. Integration to a microprocessor system

Considering common solutions, where algorithm is used as an IP core in one purpose hardware design, our aim is to provide a versatile configurable RLS lattice solution. Thus, the optimized RLS lattice core was integrated as a coprocessor to the Microblaze processor system. For its development, the Xilinx system generator (XSG) was used. It is possible to create the coprocessor PCORE, which can directly be integrated to the Microblaze system. Such solution provides maximal applicability of the RLS lattice core, although at the expense of slight performance loss

against one purpose design based on the same core. The XSG schematic of the coprocessor can be seen in Figure 6. It consists of the RLS lattice core denoted *LSL*, which can also be used as a standalone core of the input and output data buffers denoted *inBuf* and *outBuf*, of the control words *mb2hc control* and *hc2mb status*, and of the communication interface denoted *Comm*.

The *Comm* block is responsible for the batch processing of input data by the RLS lattice core in the *LSL* block. It stores filter results to the output buffer. It is capable of working with the RLS lattice core containing one up to four filter instances.

The *LSL* module consists of order updating loop (see algorithm in Algorithm 1) with the input consisting of four values. After increasing the estimation to higher order, the same four data can be regarded as output. Otherwise, only the internal states are altered. As a consequence, one filter can use another filter output as its input and to continue in the computation of order updates. The solution of higher-order filter can be obtained using the pipelined solution consisting of multiple RLS lattice filter instances. That is why the *Comm* block in the coprocessor design has to be capable of reconfiguring data path between RLS lattice instances, in order to use them either as parallel four channel RLS lattice filter (each with order up to 126), or it is possible to connect up to four instances serially and to create pipelined RLS lattice solution with order up to 504, achieving 4× higher performance. It is the reason why, in Figure 6, the *fifo* block storing the intermediate results is used in the pipelined organization of the coprocessor.

Four versions of the PCORE with one, two, three, and four RLS lattice modules were implemented. Resource requirements in the Xilinx Virtex4 SX35 device are summarized in Table 2. For illustration, the floor-plan of the system with Microblaze and the PCORE with four RLS lattice modules as a coprocessor are presented in Figure 7. The area occupied by the Microblaze processor is displayed in yellow, the FSL communication interfaces in blue, batch processing control unit in purple, and the RLS lattice filter core in green color. The entire system occupies 78% of the Xilinx Virtex4 SX35 chip.

5. DYNAMIC RECONFIGURATION

The implementation of the RLS lattice as a coprocessor connected to the Microblaze makes possible to use the dynamic reconfiguration for loading and unloading the coprocessor while the microcontroller is running. The loading of the coprocessor can be initiated on demand.

The software version of the RLS lattice filter was developed. The floating-point number representation is used in the Microblaze, whereas the 19-bit LNS is used in hardware. The software conversions based on the HSLA library were used for implementation of migration between hardware and software. The conversions are based on evaluation of base-2 logarithm contained in the Microblaze glibc library. Such conversions are time consuming even if a hardware support of floating-point is included in the Microblaze.

To control the load of the processor and the power consumption, a mechanism for migration the task from the processor to coprocessor was developed. The software version of the RLS lattice runs in the Microblaze when no other tasks require to use the processor. When there is a need to use the Microblaze for other tasks, the processor is freed and the RLS lattice is run in the coprocessor.

The available run-time configurations are presented in Figure 8. One is the software solution, remaining three are the coprocessor versions containing one to four RLS lattice filters.

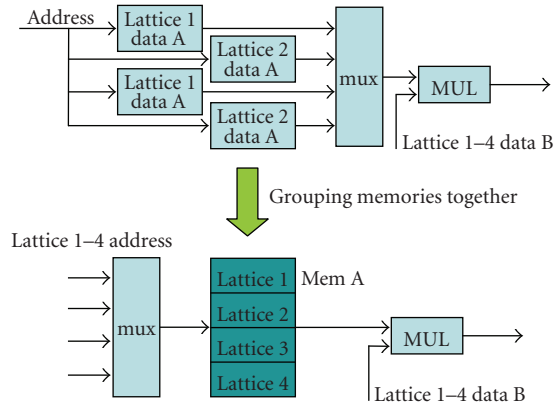

Figure 5: Reduction of multiplexers by grouping the block memories.

Table 2: RLS lattice coprocessor resources usage in the Xilinx Virtex4 SX35 device.

PCORE	1	2	3	4
Slice Flip Flops	4032	5106	6131	7242
4 input LUTs	7719	9527	11554	12357
Occupied Slices	6112	7826	9675	10270
Total 4 input LUTs	8006	9844	11905	13350
Block RAMs/DSP48	42/12			
N_{max}	126	126	126	126
M_{max}	1	2	3	4
f_{max}	44 MHz			

6. PERFORMANCE RESULTS

The performance in each run-time state was measured and the key results are summarized in Table 3. The table is divided into three parts. The first part summarizes performance of the RLS lattice filter with the probability estimation. In general, the estimation of the model order higher than 15–20 is not giving satisfactory results, as it was shown in [30]. Thus, higher order of estimation is not expected to be used. As it can be seen in the table, the performance decreases with the number of employed RLS lattice instances. The performance of hardware coprocessor is 5.5-times higher compared to the software solution, even if the clock frequency of the microprocessor is 4-times higher (this can be seen in the table by comparison of Microblaze and PCORE4 rows for $N = 16$, $M = 4$).

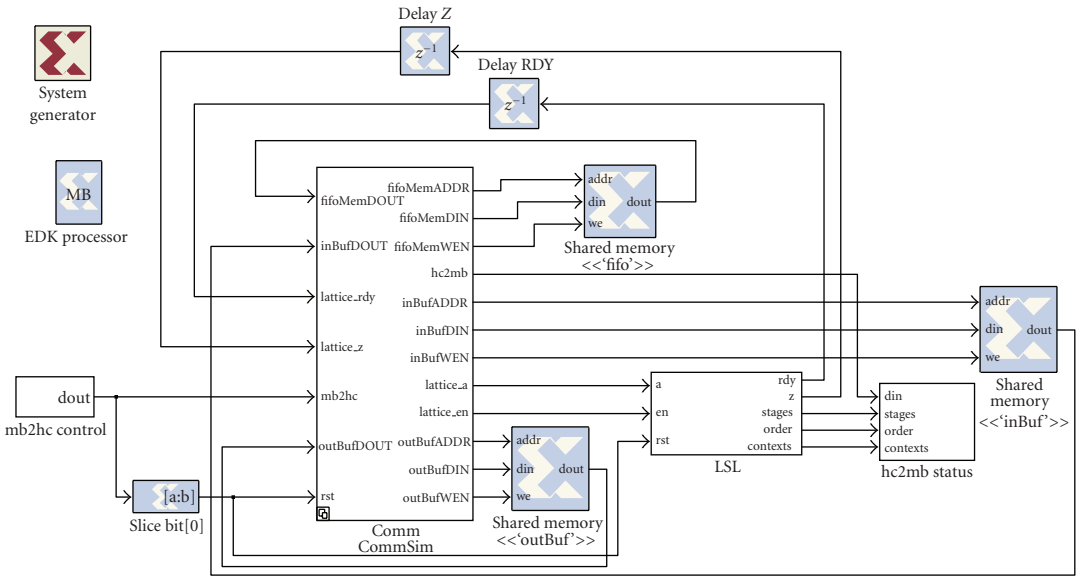

Figure 6: The RLS lattice coprocessor integration to Microblaze processor via FSL.

Table 3: Performance of the RLS lattice coprocessor: the execution time for 180 inputs is presented.

Processor	Clk [MHz]	M	N	Time (ms)	M flops
Microblaze	100	4	16	69.46	5.50
PCORE1	25	1	16	12.24	7.79
PCORE2	25	2	16	12.96	14.74
PCORE4	25	4	16	12.49	30.60
Microblaze	100	4	126	547.53	3.98
PCORE1	25	1	126	29.60	18.39
PCORE2	25	2	126	29.40	37.02
PCORE4	25	4	126	28.70	75.86
PCORE4 pipe	25	1	64	7.02	39.38
PCORE4 pipe	25	1	504	26.48	82.22

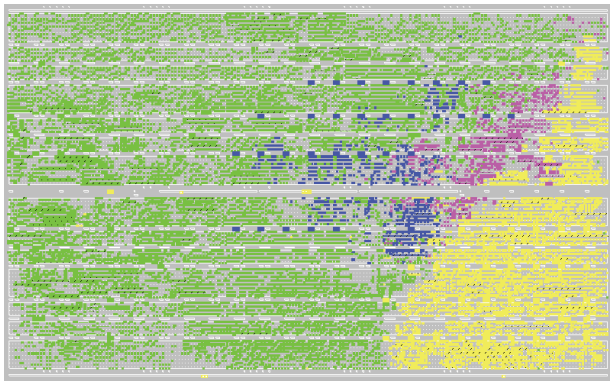

Figure 7: Floor-plan of the RLS lattice filter implementation: Microblaze processor (yellow), FSL communication interface (blue), batch processing control (in purple) and the RLS lattice core (green); the Xilinx Virtex4 SX35 usage is 78%.

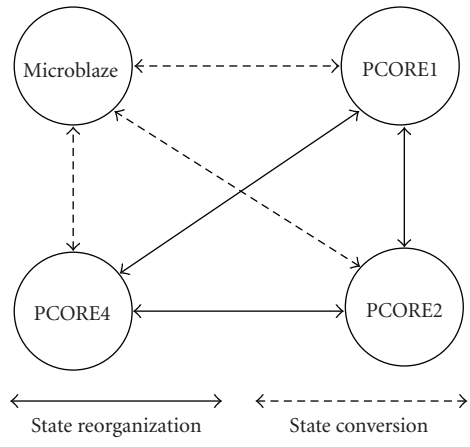

Figure 8: Reconfiguration states with switch possibilities.

The second part shows results with the probability estimation deactivated. The results for maximal supported order of 126 are presented. It can be seen that the maximal filter order exploits the best RLS lattice core optimization by

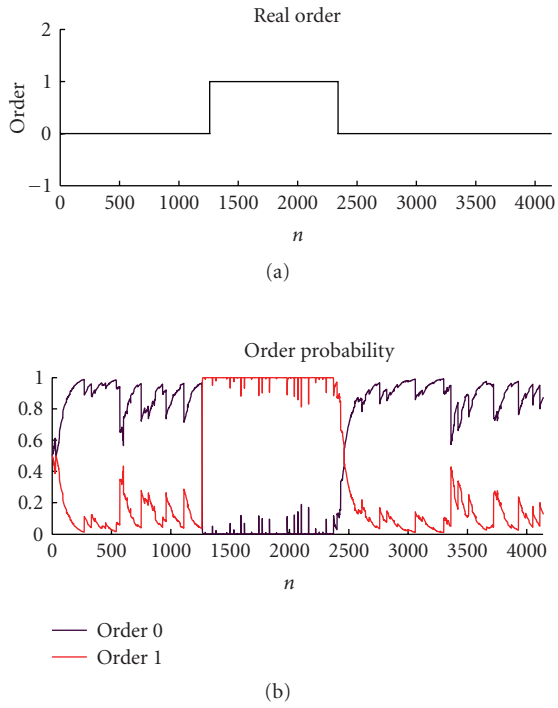

FIGURE 9: The order probability evaluation for orders 0 and 1.

cyclic scheduling. The acceleration of computation is nearly 20-times compared to the software solution running at 4-time higher clock speed.

The last part of the table shows results for the pipelined version of the RLS lattice filter. The pipelined solution uses all four RLS lattice instances, each computing 1/4 of overall estimation order. The hardware solution of order 504 is equivalent to software with $M = 4$ and $N = 126$, and the speedup of 20-times is reached again. The performance of the pipelined solution can be seen in the last row of Table 3. In the pipelined solution, the probability estimates cannot be maintained because the normalization cannot be implemented in such case.

The switch cost between run-time states was measured. The time for conversion is 411 μs/order. To change the state, local memories in the PCORE have to be reorganized. The time required for this state change was found by another experiment as 4 μs/order. Thus, we can formulate the "reorg" state and conversion times, which are $T_{\text{reorg}} = 4MN\,\mu$s and $T_{\text{conv}} = 411MN\,\mu$s, respectively.

The example of order estimation is presented in Figure 9. The upper plot shows the time when the model order was switched from 0 to 1 and back. The bottom plot shows the corresponding probability of order 0 and 1.

7. RELATED WORK

The error feedback RLS lattice filter algorithm was implemented before, which is presented in [32]. Our implementation of the lattice PCORE has similar performance. However, three more lattice filters are possible to run at the same time and hypotheses probability can be evaluated in our implementation. We have also demonstrated that four filters in coprocessor can be serialized to perform as one, 4-level pipelined filter. The performance can be 4-times better but the hypotheses estimation cannot be maintained.

There exists another RLS filter FPGA implementation presented in [33]. The performance cannot be compared since the only information provided in the paper is the clock frequency 5.3 MHz and the resource usage consisting of 2685 slices and 7 multipliers for the floating-point version and 3971 slices and 24 multipliers for the 17-bit LNS version (at 4 MHz). The PCORE implemented in our work uses 19-bit LNS and it can run at 44 MHz. In the smallest configuration, it occupies 4032 slices, 42 BRAMs, and 12 DSPs.

Our implementation is based on the algorithm presented in [17]. Its implementation for the analog devices 21061 DSP (SHARC) running at 50 MHz allows to evaluate order $N = 290$ at 8 kHz sampling rate. From Table 3 it can be extrapolated that our solution at 44 MHz can, in the case of one lattice module, operate for order $N = 168$ at 8 kHz sampling rate (1.7× slower than SHARC solution). When all four RLS lattice modules are used, the pipelined solution can theoretically reach up to $N = 753$. However, the limitation of the current implementation allows maximal value of $N = 504$ (1.7× faster than SHARC). Alternatively, the filter of order $N = 290$ can operate at the sampling rate of 20 kHz (2.5× faster than SHARC). It is important to note that our architecture allows to execute any task on the microprocessor while the RLS lattice coprocessor is busy.

8. CONCLUSIONS

The easy use and easy programming and debugging PCORE integrated in the Xilinx EDK can perform 5-times faster than the software solution for Microblaze. At the maximal order with deactivated probability estimation, the acceleration reaches up to 20 times.

The dynamic reconfiguration can be used to adapt DSP capabilities to actual demand by changing the contents of the RLS lattice coprocessor. The migration of processing from the microprocessor to HW requires $411NM$ microseconds, where N is the order and M is the number of filter instances. The time is determined mainly by the conversion from the Microblaze floating-point representation to the LNS. In the case of reconfiguration between different hardware versions, the reorganization of the internal state takes only $4NM$ microseconds. The reconfiguration controller must be designed with respect to the high cost of the migration between software and hardware solutions.

ACKNOWLEDGMENTS

This work was supported and funded by the Czech Ministry of Education, Project CAK no. 1M0567, and also by the European Commission, Project AETHER no. FP6-2004-IST-4-027611. The paper reflects only the authors view and neither the Czech Ministry of Education nor the European Commission are liable for any use that may be made of the information contained herein.

REFERENCES

[1] Z. Salcic, J. Cao, and S. K. Nguang, "A floating-point FPGA-based self-tuning regulator," *IEEE Transactions on Industrial Electronics*, vol. 53, no. 2, pp. 693–704, 2006.

[2] F. Capman, J. Boudy, and P. Lockwood, "Acoustic echo cancellation using a fast QR-RLS algorithm andmultirate schemes," in *Proceedings of the 20th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '95)*, vol. 2, pp. 969–972, Detroit, Mich, USA, May 1995.

[3] A. Nakajima, M. Kim, and H. Arai, "FPGA implementation of MMSE adaptive array antenna using RLS algorithm," in *Proceedings of the IEEE Antennas and Propagation Society International Symposium*, vol. 3A, pp. 303–306, Washington, DC, USA, July 2005.

[4] S. Haykin, *Adaptive Filter Theory*, Prentice-Hall, Upper Saddle River, NJ, USA, 4th edition, 2002.

[5] B. Widrow and S. Stearns, *Adaptive Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1985.

[6] G. Carayannis, D. Manolakis, and N. Kalouptsidis, "A fast sequential algorithm for least-squares filtering and prediction," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 31, no. 6, pp. 1394–1402, 1983.

[7] J. Cioffi and T. Kailath, "Fast, recursive-least-squares transversal filters for adaptive filtering," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 2, pp. 304–337, 1984.

[8] L. Ljung, M. Morf, and D. Falconer, "Fast calculation of gain matrices for recursive estimation schemes," *International Journal of Control*, vol. 27, no. 1, pp. 1–19, 1978.

[9] J.-L. Botto and G. V. Moustakides, "Stabilizing the fast Kalman algorithms," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 9, pp. 1342–1348, 1989.

[10] D. T. M. Slock and T. Kailath, "Numerically stable fast transversal filters for recursive least squares adaptive filtering," *IEEE Transactions on Signal Processing*, vol. 39, no. 1, pp. 92–114, 1991.

[11] D. Lee, M. Morf, and B. Friedlander, "Recursive least squares ladder estimation algorithms," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 3, pp. 627–641, 1981.

[12] F. Ling, D. Manolakis, and J. Proakis, "Numerically robust least-squares lattice-ladder algorithms with direct updating of the reflection coefficients," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 4, pp. 837–845, 1986.

[13] J. M. Cioffi, "The fast adaptive ROTOR's RLS algorithm," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 4, pp. 631–653, 1990.

[14] P. A. Regalia, "Numerical stability properties of a QR-based fast least squares algorithm," *IEEE Transactions on Signal Processing*, vol. 41, no. 6, pp. 2096–2109, 1993.

[15] P. A. Regalia and M. G. Bellanger, "On the duality between fast QR methods and lattice methods in least squares adaptive filtering," *IEEE Transactions on Signal Processing*, vol. 39, no. 4, pp. 879–891, 1991.

[16] F. Albu, J. Kadlec, C. Softley, et al., "Implementation of (normalised) RLS lattice on virtex," in *Proceedings of the 11th International Conference on Field Programmable Logic and Applications (FPL '01)*, pp. 91–100, Springer, Northern Ireland, UK, August 2001.

[17] A. H. C. Carezia, P. M. S. Burt, M. Gerken, M. D. Miranda, and M. T. M. Da Silva, "A stable and efficient DSP implementation of a LSL algorithm for acoustic echo cancelling," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '01)*, vol. 2, pp. 921–924, Salt Lake City, Utah, USA, May 2001.

[18] M. D. Miranda, M. Gerken, and M. T. M. Da Suva, "Efficient implementation of error-feedback LSL algorithm," *Electronics Letters*, vol. 35, no. 16, pp. 1308–1309, 1999.

[19] A. Hermánek, Z. Pohl, and J. Kadlec, "FPGA implementation of the adaptive lattice filter," in *Proceedings of the 13th International Conference on Field Programmable Logic and Applications (FPL '03)*, pp. 1095–1098, Springer, Lisbon, Portugal, September 2003.

[20] Z. Pohl, J. Kadlec, P. Sucha, and Z. Hanzdlek, "Performance tuning of iterative algorithms in signal processing," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '05)*, pp. 699–702, Tampere, Finland, August 2005.

[21] A. Heřmánek, J. Schier, and P. A. Regalia, "Architecture design for FPGA implementation of finite interval CMA," in *Proceedings of the European Signal Processing Conference (EUSIPCO '04)*, pp. 2039–2042, Vienna, Austria, September 2004.

[22] A. Heřmánek, J. Schier, P. Sucha, and Z. Hanzálek, "Optimization of finite interval CMA implementation for FPGA," in *Proceedings of the IEEE Workshop on Signal Processing Systems Design and Implementation (SIPS '05)*, pp. 75–80, Athens, Greece, November 2005.

[23] P. Sucha, Z. Hanzalek, A. Heřmánek, and J. Schier, "Scheduling of iterative algorithms with matrix operations for efficient FPGA design–implementation of finite interval constant modulus algorithm," *The Journal of VLSI Signal Processing*, vol. 46, no. 1, pp. 35–53, 2007.

[24] M. Karkooti, J. R. Cavallaro, and C. Dick, "FPGA implementation of matrix inversion using QRD-RLS algorithm," in *Proceedings of the 39th Asilomar Conference on Signals, Systems and Computers*, pp. 1625–1629, Pacific Grove, Calif, USA, October-November 2005.

[25] V. Peterka, "Bayesian approach to system identification," in *Trends and Progress in System Identification*, P. Eykhoff, Ed., pp. 239–304, Pergamon Press, Oxford, UK, 1981.

[26] J. Kadlec, *Probabilistic identification of regression model in fixed point*, Ph.D. thesis, UTIA CAS, Bolivar, Tenn, USA, September 1986.

[27] J. Kadlec, "Lattice feedback regularised identification," in *Proceedings of the 10th IFAC Symposium on System Identification (SYSID '93)*, pp. 277–282, Copenhagen, Denmark, 1993.

[28] J. N. Coleman, E. I. Chester, C. I. Softley, and J. Kadlec, "Arithmetic on the European logarithmic microprocessor," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 702–715, 2000.

[29] R. Matoušek, M. Tichy, Z. Pohl, J. Kadlec, C. Softley, and N. Coleman, "Logarithmic number system and floating-point arithmetics on FPGA," in *Proceedings of the 12th International Conference on Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream (FPL '02)*, vol. 2438, pp. 627–636, Springer, Montpellier, France, September 2002.

[30] J. R. Dickie and A. K. Nandi, "A comparative study of AR order selection methods," *Signal Processing*, vol. 40, no. 2-3, pp. 239–255, 1994.

[31] P. Sucha, Z. Pohl, and Z. Hanzalek, "Scheduling of iterative algorithms on FPGA with pipelined arithmetic unit," in *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '04)*, vol. 10, pp. 404–412, Toronto, Canada, May 2004.

[32] F. Albu, J. Kadlec, N. Coleman, and A. Fagan, "Pipelined implementations of the a priori error-feedback LSL algorithm using logarithmic arithmetic," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '02)*, vol. 3, pp. 2681–2684, Orlando, Fla, USA, May 2002.

[33] B. Lee and K. Lever, "Logarithmic number system and floating-point implementations of a well-conditioned RLS estimation algorithm on FPGA," in *Proceedings of the 37th Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 109–113, Pacific Grove, Calif, USA, November 2003.