ARCHITECTURE DESIGN FOR FPGA IMPLEMENTATION OF FINITE INTERVAL CMA

Antonín Heřmánek[1,2], *Jan Schier*[2], *Phillip Regalia*[1]

[1]Institute National des Télécommunications/GET
Dept. Communication, Images and Information Processing, Evry, France
[2]Institute of Information Theory and Automation
Academy of Sciences of the Czech Republic, Prague, Czech Republic

ABSTRACT

In the paper, we present the architecture design of the Finite Interval Constant Modulus Algorithm (FI-CMA) for FPGA implementation. For floating point calculations required in the algorithm we use the library based on the Logarithmic Number System (LNS). In the design, the resource reuse and minimization of the total latency is emphasized.

1. INTRODUCTION

In modern digital communication systems an estimator of transmited symbols represents one of the critical parts of the receiver. The estimator consists typically of an equalizer and of a decision device. Recent systems (such as GSM system) use well known methods based on training sequences, where a part of signal is known and repeated. The equalizer is based on matching its output to the reference signal, by adapting its parameters to minimize some criterion (typically MSE). Unfortunately the training sequence consumes a considerable part of the overall message (approx. 25% in GSM). For this reason, much research effort has been devoted to blind deconvolution algorithms, i.e., algorithms with no training sequence. Perhaps the most popular blind algorithm is the Constant Modulus Algorithm (CMA), originally proposed by Godard [2].

The FPGA represents the technology with massive fine tuned parallelism and high data throughput. As target device for the algorithm implementation we have chosen the XCV2000E device. This device offers both enough on-chip dual-port block RAMs, which are necessary to store the intermediate results, and sufficient amount of logic.

The paper is organised as follows: in the next section the system model and the optimisation criterion are discussed. In Section 3, the Finite Interval CMA algorithm is briefly reviewed. In Section 4, numerical issues are considered. Section 5 presents the proposed design architecture and finally in the Section 6, the work is concluded.

2. CONSTANT MODULUS ALGORITHM

In this section the system model and CM criterion are briefly presented. In the following text all vectors are assumed to be in column orientation.

Let $\{s_n\}$ be the symbol sequence to be transmited. It is assumed to be independent and identically distributed, and adhering to a constant modulus (CM) constellation, such as PSK. The data symbols are transmitted over a Single-Input Multiple-Output (SIMO) discrete channel with impulse response matrix $\mathbf{H}$, assumed to have finite length. The received signal has the form:

$$\mathbf{u}_n = \mathbf{H}\mathbf{s}_n + \mathbf{b}_n \tag{1}$$

where $\mathbf{s}_n = [s_n\ s_{n-1}\ \dots\ s_{n-M}]^T$ collects the M most recent input symbols, $\mathbf{b}_n$ is a background noise vector, $\mathbf{H}$ is the $P \times M_c$ channel impulse response matrix, and P is the number of antennas or the oversampling factor. An equalizer is viewed as a linear combiner of order M and its output can be written in the form:

$$y_n = \sum_{k=0}^{M} \mathbf{g}_k^T \mathbf{u}_{n-k} = \mathbf{g}^T \mathbf{U}_n \tag{2}$$

where n is the discrete baud rate time and $\mathbf{g}$ and $\mathbf{U}$ are defined as:

$$\mathbf{g} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_M \end{bmatrix}, \qquad \mathbf{U}_n = \begin{bmatrix} \mathbf{u}_n \\ \mathbf{u}_{n-1} \\ \vdots \\ \mathbf{u}_{n-M} \end{bmatrix}$$

The CMA algorithm tries to minimize a cost function defined by the constant modulus (CM) criterion which penalizes deviation in the magnitude of the equalizer output from a fixed value. This criterion has the form:

$$J_{CMA}(\mathbf{g}) = \frac{1}{4} E\left[\left(|y_n|^2 - \gamma\right)^2\right] \tag{3}$$

where $E[\cdot]$ is expectation operator and γ is a constant chosen as a function of the source alphabet.

The main advantage of this criterion is that the resulting gradient descent algorithm is very similar to the well known LMS algorithm. The relation between the LMS and gradient CMA algorithm have been discussed in many papers. It was shown that the CMA's cost surface can be directly related to LMS and that the LMS convergence rate expressions can be used to provide the limits of CMA tracking capabilities. In addition, the relatively slow convergence of CMA ($\approx 10^4$ iterations), as well as its dependence on the initialization and on the step-size parameter, are recognized drawbacks.

3. FINITE INTERVAL CMA

The FI-CMA is a windowed version of (3) where a time-window operator is applied to the received data (i.e., the expectation operator is replaced by summation over finite data interval) and its cost function has the form:

$$J(\mathbf{g}) = \sum_{n=1}^{N} \left(|y_n|^2 - 1 \right)^2 = \sum_{n=1}^{N} \left(\left| \mathbf{g}^T \mathbf{U}_n \right|^2 - 1 \right)^2 \quad (4)$$

where the constant γ is replaced by 1 without loss of generality because its value does not change the position of the local extrema points. In [5] it was shown that local extrema of (4) coincide with the local extrema of the function:

$$\mathbf{F}(\mathbf{g}) = \frac{\sum_{n=1}^{N} y_n^4}{(\sum_{n=1}^{N} y_n^2)^2} \quad (5)$$

From the equation (2), N successive equalizer outputs can be directly rewritten in matrix form as:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{U}_1^T \\ \mathbf{U}_2^T \\ \vdots \\ \mathbf{U}_N^T \end{bmatrix}}_{\mathcal{U}} \mathbf{g} = \mathbf{Q}\mathbf{R}\mathbf{g} = \mathbf{Q}\mathbf{w} \quad (6)$$

where the QR-decomposition of matrix $\mathcal{U}$ is used to obtain an orthonormal matrix $\mathbf{Q}$.

The optimal equalizer coefficients are reached by the following iterative procedure:

$$\begin{array}{rcl} \mathbf{v}_{i+1} & = & \mathbf{w}_i - \mu \mathbf{Q}^T \mathbf{y}^3 / \mathbf{F}_i \\ \mathbf{w}_{i+1} & = & \mathbf{v}_{i+1} / \|\mathbf{v}_{i+1}\| \end{array} \quad (7)$$

where i is the iteration counter. In [6], the optimal step-size has been derived of the form $\mu_i = \frac{1}{3-F_i}$. With that setting of μ and with the oversampling factor 2, the procedure (7) convergs typically after 10 iterations.

The QR-decompositon is calculated using Givens rotations as follows. Let $u_{i,k}$ be the i^{th} element of vector $\mathbf{U}_k$. Let $\mathbf{R}_k$ be a triangular matrix obtained by triangularization of sub-matrix $\mathcal{U}_k$ (first k rows of matrix $\mathcal{U}$) and $\mathbf{Q}_k$ is its corresponding orthonormal matrix. Then matrices $\mathbf{Q}_{k+1}$ and $\mathbf{R}_{k+1}$ can be calculated as follows:

$$\begin{bmatrix} \mathbf{R_{k+1}} \\ \mathbf{0} \end{bmatrix} = G_N G_{N-1} \dots G_1 \begin{bmatrix} \mathbf{R}_k \\ \mathbf{u}_{k+1} \end{bmatrix} \quad (8)$$

$$\left[\begin{array}{c|c} \mathbf{Q}_{k+1} & \mathbf{q}_k \end{array} \right] = \left[\begin{array}{c|c} \mathbf{Q}_k & 0 \\ \hline 0 & 1 \end{array} \right] G_1^T \dots G_{N-1}^T G_N^T \quad (9)$$

where the matrix G_i eliminates $u_{i,k}$ and is defined as follows:

$$G_i = \begin{bmatrix} c_i & & s_i & \\ & I & & \\ -s_i & & c_i & \\ & & & I \end{bmatrix}, \quad I = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}$$

The sine and cosine parameters are computed using the following formulae:

$$c_i = \frac{r_{i,i}}{\sqrt{r_{i,i}^2 + u_{i,k}^2}} \qquad s_i = \frac{u_{i,k}}{\sqrt{r_{i,i}^2 + u_{i,k}^2}} \quad (10)$$

Note: The calculation of $\mathbf{R}$ and $\mathbf{Q}$ for $k = 1 \dots N$ is provided in the similar way (see [3] for more details).

4. NUMERICAL REQUIREMENTS

To summarize, the FI-CMA algorithms consist of two successive parts: the QR-decomposition of input data matrix and the iterative procedure (7). The dynamic range of data and intermediate results in both these parts requires that floating point arithmetic is used for arithmetic operations. That is, however, rather costly for an FPGA implementation. In our work, we use the Logarithmic Number System (LNS) arithmetic.

4.1 Logarithmic arithmetic

This arithmetic is based on logarithmic numeric representation of floating-point numbers. The logarithmic equivalent of a number – a two's complement fixed-point value equal to $\log_2 |x|$ – is mapped to an integer. Current versions of the library offer 19- and 32-bit wordlengths. The number consists of an integer part, which is always 8 bits long, and of a fractional part, the size of which depends on the selected data precision.

Multiplication and division then transform to a fixed-point addition and subtraction, and the square-root operation becomes a simple bit-shift.

To implement addition and subtraction, the logarithmic function $\log_2(1 \pm 2^{b-a})$ has to evaluated. This function is evaluated using a first-order Taylor-series approximation with look-up tables. While the size of these tables often represents substantial problem, in our solution, they are kept small by using an error correction mechanism and a range-shift algorithm [1].

The logarithmic operations were implemented in the High-Speed Logarithmic Arithmetic (HSLA) library. They are fully pipelined, addition and subtraction have each 8 clock cycles latency, other operations have 1 clock cycle latency. In order to utilize the look-up tables, which are implemented in dual-port Block RAMs, efficiently, the ADD/SUB unit has been implemented as a twin unit. For more details see [1, 4].

The resource utilization figures of the HSLA operations are summarized in Table 1.

Parameters (19/32-bit, XCV2000E-6)			
Op.	**Lat.**	**SLICEs**	**BRAMs**
ADD/SUB	8	8/13%	3/70%
MUL	1	1%	0%
DIV	1	1%	0%
SQRT	1	1%	0%

Table 1: Resource utilization of the HSLA cores

4.2 Performance comparison

In this section we compare the performance of our 32-bit and 19-bit LNS implementations with conventional

32-bit floating point. The comparison is based on the computer simulations using double- and single precision floating point arithmetic and 32- and 19-bit bit-exact LNS functions.

As a reference model, we have chosen the implementation using the IEEE double precision arithmetic. As a performance comparison measure we have used the variance of the error signal $e = (y_i - \hat{y}_i)$.

We define the signal-to-noise ratio caused by the round-off error as:

$$SNR = -10 \log_{10} \frac{\sigma^2}{\sigma_e^2} \tag{11}$$

where σ^2 is the variance of the output y, and σ_e^2 is the variance of the corresponding error signal.

The values were measured for $N = 500, M = 4, P = 2$, for the following configurations (note that SNR increases with N due to accumulation of errors in the QR decomposition):

- Floating point 32 bit and LNS 32 bit: SNR=−90 dB;
- QR 64 bit, iterative equalizer update 19 bit – to test performance degradation of the update itself: SNR = −56 dB;
- All computations in 19 bit LNS, to test the total performance degradation: SNR = −20 dB.

Although the SNR drops in the last case, the ISI eye remains open. Following the results of experiments not included in the paper, the 19-bit implementation should be sufficient.

5. DESIGN ARCHITECTURE

To implement the algorithm, the following computational units are needed:

1. The *QR-processor* which consist of:
   - the *diagonal processor* of the matrix $\mathbf{R}$, computing the rotation sine and cosine values;
   - the *off-diagonal processor*, which rotates rows of the matrix $\mathbf{R}$;
   - the *column processor*, which rotates the columns of the matrix $\mathbf{Q}$.

   Note that in contrast to the usual design of the QR-update, we compute explicitly the first N columns of the $\mathbf{Q}$ matrix.
2. The *Equalizer update processor*, which consists of the following parts:
   - the *matrix-vector multiplier* for $\mathbf{y} = \mathbf{Q}\mathbf{w}$;
   - the *parallel processor* for calculation of $\hat{\mathbf{w}} = \mathbf{Q^T y^3}$ and $\mathbf{F}_i = \sum \mathbf{y}^4$;
   - the *parameter update* processor for $\mathbf{v} = \mathbf{w} - \mu\hat{\mathbf{w}}/\mathbf{F}_i$ and $\mathbf{w} = \mathbf{v}/\|\mathbf{v}\|$.

For the overall implementation architecture design we have to keep in mind the following factors: matrix dimensions, restrictions on parallel access to the device memories (dual ported RAMs) and limited number of the LNS adders that can be fit into a single device.

For the matrix dimensions, let N and M be the equalizer order and the data block size, respectively. Then, $N \ll M$. Typical values are $N = 10 \ldots 20$ and $M = 250 \ldots 1000$. It follows that while $\mathbf{R}$ is a triangular $(N \times N)$ matrix, $\mathbf{Q}$ is an $(N \times M)$ matrix, i.e., it has many more rows than columns.

Because of the complexity of the design (the number of "processors" to be implemented), we have decided to use the 19-bit precision, to be able to use more adders while achieving acceptable precision. We assume that the data block is stored in some external FIFO memory. During each update step, only the data vector $\mathbf{u}_n$ is read which means that only P 19-bit values are transfered (P is an oversampling factor). The input data vector $\mathbf{U}_n$ is composed/stored in an internal circular buffer which is used to prepare the input for the QR processor.

5.1 QR-processor architecture

The $\mathbf{R}$ matrix update procedure is implemented using one diagonal and one off-diagonal processor. While the off-diagonal cell depends on the output of the diagonal cell for the same input data, the rotation of i-th row of the matrix $\mathbf{R}$ can be fully pipelined and can be computed in parallel with the update of $\mathbf{Q}$. Because of this data dependency, the diagonal and off-diagonal processor can share a single twin-adder. Data flows in both cells are shown in Figure 1 and 2.

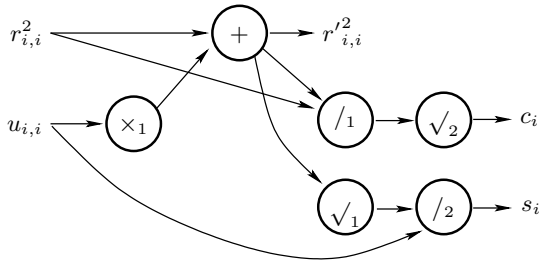

Figure 1: Data flow of diagonal cell

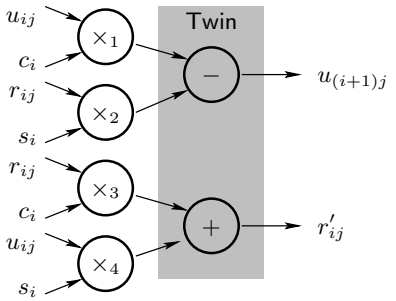

Figure 2: Data flow of $\mathbf{R}$ off-diagonal and $\mathbf{Q}$ column cell

Since the number of rows in $\mathbf{Q}$ increases with every new data vector in a single block, the computational complexity of this matrix grows linearly from the begining to the end of the block. (Note the complexity of update of matrix $\mathbf{R}$ remains constant.) The intermediate results are stored in two dual-port RAMs: one for the elements of the $\mathbf{Q}$ matrix, the other for the rightmost column of the composed matrix in equation (9). The data flow in the column processor is the same as in the off-diagonal processor of $\mathbf{R}$ (Figure 2).

The implementation probably cannot be extended to employ more processors, because of the above mentioned

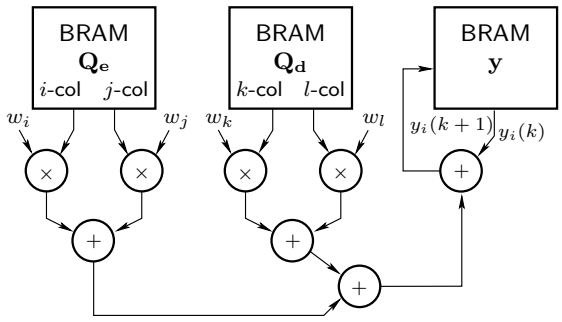

Figure 3: Matrix-vector multiplier

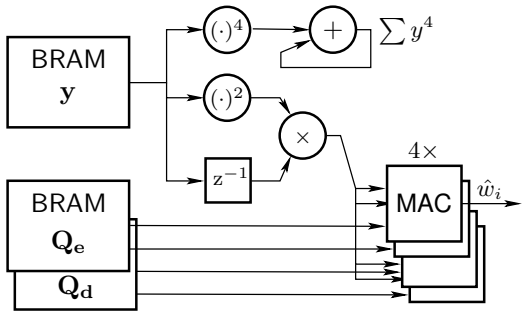

Figure 4: Parallel processor architecture

limitations (only two ports in each RAM and a growing number of rows).

5.2 Equalizer update procedure

The equalizer output (6) is computed using the vector-matrix multiplier. When the multiplication is evaluated in the usual way, i.e., row-wise with respect to matrix $\mathbf{Q}$, the efficiency of the resource utilization will be rather small. This is due to the relatively high latency of the LNS adder (and, consequently, the latency of the LNS Multiply-and-Accumulate unit) with respect to the row length (the number of elements in the MAC operation). To improve the efficiency, we propose to compute the-multiplication in a column-wise form: all elements of i^{th} column will be multiplied by the i-th element of the coefficient vector $\mathbf{w}$. The result will be stored in temporal memory. Using the dual-port Block RAMs, we are able to address two columns of $\mathbf{Q}$ at the same time. Since we do not accumulate (just multiply and add), there will be no dummy cycles. When partitioning the $\mathbf{Q}$ matrix to two Block RAMs, for example to $\mathbf{Q}_e$ and $\mathbf{Q}_d$ with even and odd columns of $\mathbf{Q}$, we may employ two multiply-add units in parallel. As a result, all adders that were used for the QR-decomposition will be utilized. The structure of the unit is shown in Figure 3

The *parallel processor* computes the value of the criterion function which is simply $\sum y_i^4$ and the matrix-vector multiplication $\mathbf{Q}^T\mathbf{y}^3$. Since M is much higher than the latency of the MAC unit, it can be used efficiently here. With the $\mathbf{Q}$ matrix partitioned into two separate block RAMs, we may use up to four MAC units and reuse all the available hardware. The structure of the processor is shown in Figure 4. It has to be noted that with the LNS arithmetic, computation of the powers y^3 and y^4 is cheap on the amount of logic and can be calculated in a single cycle.

The structure of the *parameter update* processor is simple: it performs mainly computation of the step size $\mu_i = \frac{1}{3-\mathbf{F}_i}$, division and multiplication of a vector by a constant and a second norm calculation. Again, thanks to the properties of the LNS arithmetic, the computation of multiplication/division as well as the square- and fourth root are inexpensive. The architecture of the unit is straigforward.

6. CONCLUSIONS

An architecture for an FPGA implementation of the FI-CMA algorithm has been proposed in this paper. This algorithm represents complex and computationally intensive advanced DSP algorithm, which requires floating-point arithmetic operations including higher order power and root operations. The properties of the Logarithmic Numbering System have been used with advantage: multiplication and power/root operations with very low latency and low resource utilisation. The aim of the design was that the addition core, which consumes most resources of all operations (for the necessary look-up tables) is reused as much as possible. Also the memory partitioning was considered so as to allow higher parallelization of computations and to keep the total latency as small as possible, to achieve maximum performance. The resulting design will fit in a single XCV2000E device.

REFERENCES

[1] J. N. Coleman, E. I. Chester, C. I. Softley, and J. Kadlec. Arithmetic on the european logarithmic microprocessor. *IEEE Transactions on Computers*, 49(7):702–715, 2000.

[2] D. N. Godard. Self-recovering equalization and carrier tracking in two-dimensional data communication systems. *IEEE Trans. Communications*, 28:1867–1875, November 1980.

[3] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996.

[4] R. Matoušek, M. Tichý, Z. Pohl, J. Kadlec, and C. Softley. Logarithmic number system and floating-point arithmetics on FPGA. In M. Glesner, P. Zipf, and M. Renovell, editors, *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, volume 2438 of *Lecture Notes in Computer Science*, pages 627–636, Berlin, 2002. Springer.

[5] P. A. Regalia. A finite interval constant modulus algorithm. In *Proc. International Conference on Acoustics, Speech, and Signal Processing(ICASSP-2002)*, volume III, pages 2285–2288, Orlando, FL, May 13-17 2002.

[6] P. A. Regalia and E. Kofidis. Monotonic convergence of fixed-point algorithms for ICA. *IEEE Trans. Neural Networks*, 14(4):943–949, July 2003.