An Efficient Regular Matrix Inversion Circuit Architecture for MIMO Processing

Isabelle LaRoche and Sébastien Roy

Department of Electrical and Computer Engineering,
Laval University, Québec, Qc, G1K 7P4, Canada
Email:{ilaroche, sebasroy}@gel.ulaval.ca

***Abstract*— A novel circuit architecture and algorithm is presented for the efficient implementation of a matrix inversion unit. The division-free algorithm yields a scaled version of the inverse and the scaling factor. Based on the Sherman-Morrison formula, the proposed architecture is characterized by regular, locally-connected arrays of processing units and simple iterative processing. It is especially well-suited for covariance matrices, or any other matrix which can be constructed from rank-one updates of an initial matrix whose inverse is known. While it constitutes an ideal solution for antenna array MMSE (minimum mean-square error) processing, it can also be generalized to many other applications with little effort. Implementation results of a heavily pipelined matrix inverter on a Xilinx Virtex-II FPGA are presented, including cost in logic slices and maximum clock frequency. The cost / complexity of the proposed solution is comparable to, and in many cases better than, known alternatives.**

I. Introduction

Matrix inversion is a common operation in many signal processing problems, including most block adaptation methods used in adaptive communication systems.

In previous efforts [1], [2], implementation in hardware is often non-trivial because of the need for complex operators, e.g. division or square root. Many proposed architectures rely on some form of factorization such as Cholesky [3] or QR [4], [5], [6]. The latter class is of particular interest, and includes many variants which exploit Givens rotations [7] for efficient hardware-based QR decomposition. In particular, the implementation described in [5] exploits squared Givens rotations (SGR) [8] to avoid the square root operator, as was originally suggested in [9]. However, a number of divisions is still required. The triangular locally-connected array of processing units used therein yields directly a weight vector for array processing where the necessary matrix inversion is implicit. This approach is characterized by a very complex series of events and timing requirements, and therefore requires a substantial design effort to emulate.

In [3], the use of the Sherman-Morrison formula was employed to construct an inversion circuit which inverts a series of matrices where two successive matrices differ only by a small perturbation. The approach proposed herein uses a modified form of the Sherman-Morrison formula to invert a covariance matrix from scratch, while avoiding divisions thanks to appropriate scaling.

There is some evidence that methods which involve recurrence, such as Sherman-Morrison, iterative methods, and various forms of Givens rotations-based inversion, typically provide better numerical robustness and precision than conventional direct inversion methods [6]. Simulations have shown that this is true for the proposed method. Furthermore, a means of dynamic scaling control was devised to maximize the precision of the result when integer arithmetic is used.

II. Optimal combining using Minimum Mean Square Error criteria

Figure 1 shows a typical $M \times M$ Multi-Input Multi-Output (MIMO) system. In a Layered Space-Time (LST) scheme [10], each transmitting antenna's signal is at some point estimated by passing through a linear weight-and-sum structure (spatial filter), where the optimal weight vector can be chosen according to the zero-forcing (ZF) or the minimum mean-square error (MMSE) criterion.

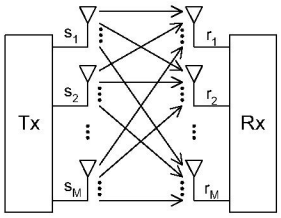

Fig. 1. An $M \times M$ MIMO system

In the presence of interference, the optimal method is MMSE combining, which requires knowledge of the received signal autocorrelation matrix to compute the weights:

$$\mathbf{w} = \alpha \mathbf{R}_{xx}^{-1} \mathbf{c}_0^*, \tag{1}$$

where α is a constant, $\mathbf{c}_0^*$ is the desired channel vector and $\mathbf{R}_{xx}^{-1}$ is the autocorrelation matrix.

Assuming the noise and interfering signals are uncorrelated, the autocorrelation matrix is given by

$$\mathbf{R}_{xx} = \sigma^2 \mathbf{I} + \sum_{i=1}^{M} \mathbf{c_i} \mathbf{c_i^H}, \tag{2}$$

where σ^2 is the noise power, $\mathbf{I}$ is the identity matrix, $\mathbf{c}_i$ and $\mathbf{c}_i^H$ are the channel vector and transposed-conjugate channel vector, respectively, of the i^{th} transmitted signal.

The LST receiver architecture which constitutes the motivation for the present work was designed in such a way that estimates of the channel vector $\mathbf{c}_i$ are successively obtained, and are used to construct the $\mathbf{R}_{xx}$ matrix for each layer. This structure given in (2) can therefore be exploited to perform inversion.

III. Sherman-Morrison formula

The Sherman-Morrison formula [11], being a special case of the matrix inversion lemma, allows the easy computation of the inverse of a slightly-modified matrix A given that the inverse of the original matrix is known. The perturbation must take the form of a rank-1 update, e.g. $\mathbf{uv}^H$, where $\mathbf{u}$ and $\mathbf{v}$ are vectors of appropriate dimensions.

Given A^{-1}, the Sherman-Morrison formula is

$$(\mathbf{A^{-1}} + \mathbf{uv}^H)^{-1} = \mathbf{A^{-1}} - \frac{(\mathbf{A^{-1}uv}^H\mathbf{A^{-1}})}{1 + \mathbf{v}^H\mathbf{A^{-1}u}}. \quad (3)$$

By examining (2), it can easily be seen that (3) can be directly applied to the autocorrelation matrix inversion problem. Using a different notation, (3) becomes:

$$\mathbf{R}_i^{-1} = (\mathbf{R}_{i-1} + \mathbf{c}_i\mathbf{c}_i^H)^{-1} = \mathbf{R_{i-1}^{-1}} - \frac{(\mathbf{R_{i-1}^{-1}c}_i\mathbf{c}_i^H\mathbf{R_{i-1}^{-1}})}{1 + \mathbf{c}_i^H\mathbf{R}_{i-1}^{-1}\mathbf{c}_i}, \quad (4)$$

where $\mathbf{R}_i = \sigma^2\mathbf{I} + \sum_{k=1}^{i}\mathbf{c}_k\mathbf{c}_k^H$, so that $\mathbf{R}_M = \mathbf{R}_{xx}$, and $\mathbf{R}_0 = \sigma^2\mathbf{I}$.

From a hardware perspective, equation (4) is problematic because of the presence of division. By introducing appropriate scaling, the division can be translated into a multiplication, thus avoiding this difficulty. The recurrent relation then becomes

$$\begin{aligned}\mathscr{R}_i^{-1} &= (\alpha_{i-1} + \mathbf{c}_i^H\mathscr{R}_{i-1}^{-1}\mathbf{c}_i)\left(\mathscr{R}_{i-1} + \frac{\mathbf{c}_i\mathbf{c}_i^H}{\alpha_{i-1}}\right)^{-1} \\ &= \mathscr{R}_{i-1}^{-1}(\alpha_{i-1} + \mathbf{c}_i^H\mathscr{R}_{i-1}^{-1}\mathbf{c}_i) - \\ &\quad (\mathscr{R}_{i-1}^{-1}\mathbf{c}_i\mathbf{c}_i^H\mathscr{R}_{i-1}^{-1}), \end{aligned} \quad (5)$$

where $\mathscr{R}_i^{-1} = \alpha_i\mathbf{R}_i^{-1}$ and the scaling factor α can be expressed as:

$$\alpha_{i+1} = \alpha_i(\alpha_i + \mathbf{c}_{i+1}^H\mathscr{R}_i^{-1}\mathbf{c}_{i+1}), \quad (6)$$

with $\alpha_0 = 1$.

IV. System architecture

A. System description

Shown in Figure 2, the overall system architecture directly implements the modified Sherman-Morrison formula (5). Each box corresponds to a specific hardware unit, the main ones being detailed in section V. At the system level, it is essentially a dataflow architecture with some amount of pipelining. Individual arithmetic units performing matrix and/or vector operations are regular, locally-connected arrays.

Memory blocks and a high number of multipliers are required, as well as other basic logic elements. In part because of these requirements, the Xilinx Virtex-II family of FPGAs seemed a natural target platform since it provides distributed hard-wired 18×18 multipliers and RAM blocks, which free up the generic logic for the other required components (adders, flip-flops, counters, etc). Simulation of the system was performed using Matlab-generated test vectors in the Modelsim XE testing environment.

V. Module description

This section gives an in-depth view of the important modules in the implementation.

A. Matrix-vector multiplication

Shown in Figure 3, this unit computes the product $\mathbf{c}_i^H\mathscr{R}_{i-1}^{-1}$. Multiplexers are needed in order to select either the initial matrix, $\sigma^2\mathbf{I}$, or the result of the previous iteration. These signals are input at the top of a linear M-cell systolic array comprised of multiply-accumulate cells. The current channel vector is conjugated and input from the left of the array. The resulting vector is fed into a series of AND gates to ensure that only the final and valid result, not the intermediary value, is sent to the next modules.

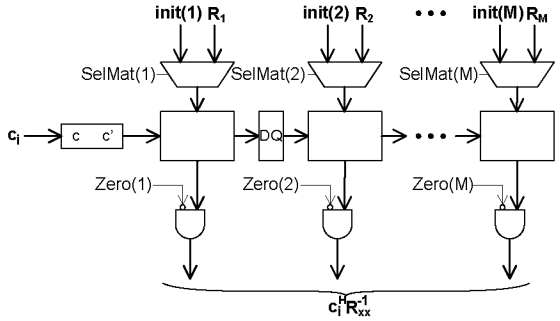

Fig. 3. Matrix-vector multiplication

B. Matrix-matrix multiplication

Figure 4 details the matrix multiplier which computes $\mathscr{R}_{i-1}^{-1}\mathbf{c}_1\mathbf{c}_1^H\mathscr{R}_{i-1}^{-1}$. Since $\mathscr{R}_{i-1}^{-1}\mathbf{c}_1 = (\mathbf{c}_1^H\mathscr{R}_{i-1}^{-1})^H$, both inputs of the unit come from the matrix-vector multiplier, one branch being fed to a conjugate unit beforehand (see Figure 2). The unit is comprised of a $M \times M$-cell systolic array. Each cell has the same structure as the one described above, with the addition of a vertical passthrough link to propagate the top inputs downwards. The results are multiplexed onto M output lines in accordance with a specific structure.

C. Vector-vector multiplication

This unit implements $\mathbf{c}_i^H\mathbf{R}_{i-1}^{-1}\mathbf{c}_i$. The result of the matrix-multiplication unit first has to be transposed before being multiplied by the channel coefficient vector. This is performed by a multiplexer which effectively acts as a parallel-to-serial converter. Both vectors are fed to a single cell.

D. Scaling factor addition

Figure 5 shows the implementation of $\alpha_{i-1} + \mathbf{c}_i^H\mathscr{R}_{i-1}^{-1}\mathbf{c}_i$ and (6). A flip-flop is used as a memory element to store the value of α. This value is added to the vector-vector multiplication result. The new α is computed and stored into the memory element.

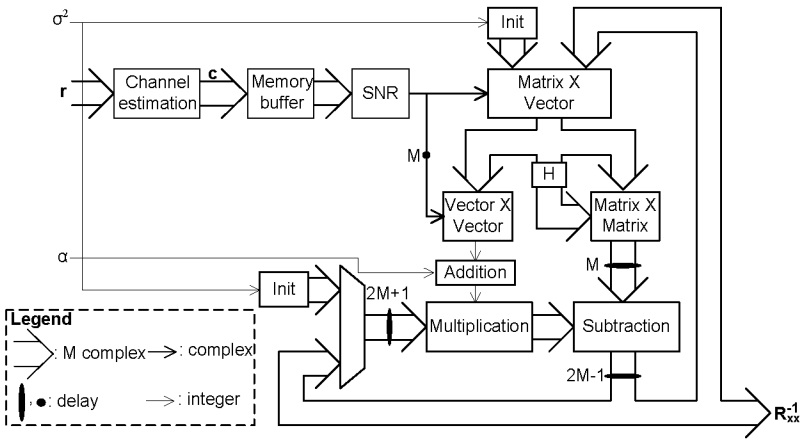

Fig. 2. General system architecture

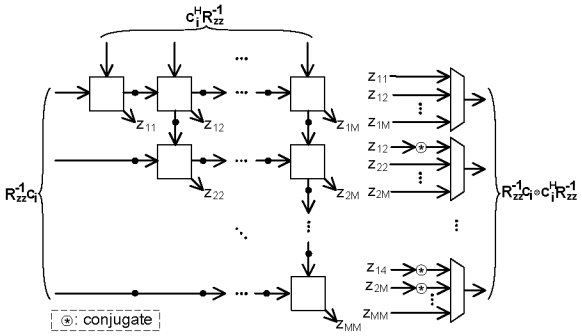

Fig. 4. Matrix-matrix multiplication

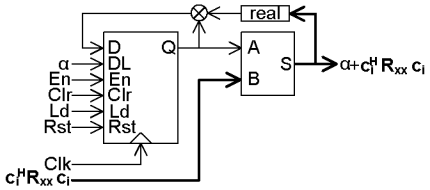

Fig. 5. Scaling factor addition

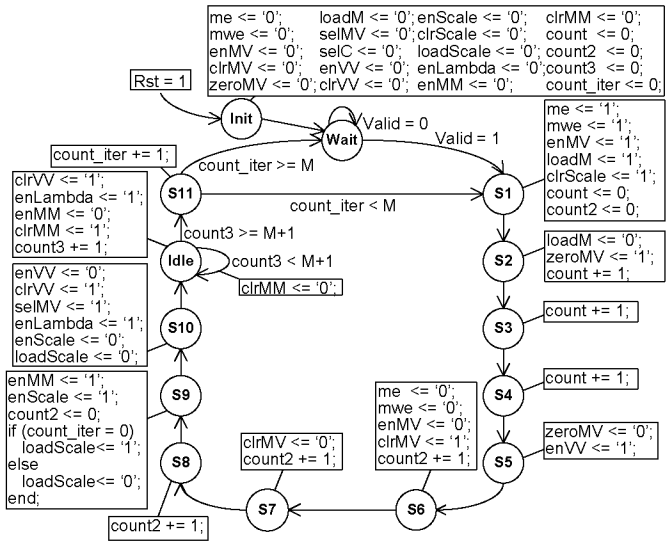

Fig. 6. State Machine

VI. Sequencer

Figure 6 shows the diagram of the state machine generating all the control signals required to ensure proper sequencing of the system.

Because of the layered structure, the circuitry is used more than once to compute the inverse and many units need to be turned on and off at specific times. This is crucial for the units containing processing cell arrays. Also, the accumulated values from a previous iteration must be cleared before starting the next computation. Enable (*enMV*, *enMM* and *enVV*) and clear (*clrMV*, *clrMM* and *clrVV*) signals are thus needed to control the array cells. In the linear and square arrays, the cells must be controlled by individual enable and clear signals. To achieve this simply, while maintaining the generic, scalable quality of the design, a single enable and a single clear signal are generated by the FSM. A sequence of flip-flops provides various delayed versions of these master signals.

The generation of the initial matrix also requires some control circuitry. First, M flip-flops are loaded with the σ^2. The load signal *loadM* generated by the FSM is for the first flip-flop and since this value is exclusively on the diagonal, a delay of 2 clock cycles is introduced between *loadM* and the other flip-flops.

The system contains many multiplexers whose select signals are generated by the FSM. The multiplexer for the channel coefficient vectors switches inputs at each iteration, so that its select signal,*selC*, is tied to a counter, *count_iter* which counts the number of iterations completed so far. For the input multiplexers of the matrix-vector multiplication unit, the switch between the two inputs must not occur at the same time for all M multiplexers because each line is delayed one clock cycle from the previous. Thus, the select signal generated by the FSM, *selMV*, is used directly by the first mux and delayed for the other muxes. *selMV* is also delayed and used by the

multiplexer for the matrix input of the multiplication unit. The input multiplexer of the vector-vector multiplication unit must change at every clock cycle for a total of M switches, so it is tied to counter *count2* controlled by the FSM. The output multiplexers for the matrix-matrix multiplication must also switch between lines at every clock cycles for a total of M switches. A series a flip-flops delay *selMV* by the appropriate number of cycles and the end signal is used by the first mux. Again, not every mux must be switched at the same time, so flip-flops are used as a mean to delay the select signal for the remaining muxes.

The scaling factor addition contains a memory element that must be controlled by the FSM. Three signals — *enScale*, *loadScale* and *clrScale* — are used to properly control this unit. Particularities of these signals are that *loadScale* is tied to one for a single clock cycle during the entire inverting process and *clrScale* is also tied to one for a single clock cycle, at the end of the M iterations before the next inversion starts.

Specific to the matrix-vector multiplication unit is a control signal fed to the output AND gates called *zeroMV*. Again, the one generated by the FSM is delayed before being fed to the other AND gates.

VII. Implementation results

The implemented system was synthesised using Xilinx XST on a Virtex-II XC2V6000. Preliminary logic resource cost results are shown in Table I. The maximum clock frequency is currently 108.3 MHz and avenues have been identified for further optimization.

	Logic Slices	4-input LUTs	18 X 18 Multipliers
Matrix-vector multiplication	765/33792 2%	1412/67584 2%	16/144 11%
Matrix-matrix multiplication	3108/33792 9%	5850/67584 8%	64/144 44%
Vector-vector multiplication	187/33792 0%	350/67584 0%	4/144 2%
Multiplication	780/33792 2%	376/67584 0%	16/144 11%
Total	4446/33792 13%	7805/67584 11%	101/144 70%

TABLE I
Implementation results

A single matrix inversion requires $4M^2$ clock cycles, exploiting $M^2 + M$ simple processing cells. In comparison, Edman's solution [2] is a linear array of $2M$ processors with $O(2P(M^2 + M - 1))$ time units being required to complete an inversion, where P is the degree of pipelining with each processing cell. A high degree of pipelining is mandatory (the author proposes $P = 5$) to obtain a decent clock frequency because of the high latency of the required division operator. The very best methods [1] which also require $M^2 + M$ processors (typically as two triangular arrays of $\frac{M^2+M}{2}$ elements) manage to compute the inverse in linear time. However, these methods necessarily involve division and/or square root operations and both the individual processing cells and the sequencer are much more complex than with the proposed method. Furthermore, the $O(M^2)$ completion time of the proposed technique is due to the systolic-style dataflow (which simplifies routing and control signal requirements). Linear time is achievable in principle by using pipelining between iterations and between successive inversions.

VIII. Conclusion

A division and square-root free matrix inversion unit was proposed and designed for incorporation into a layered space-time MIMO receiver. It is implemented using integer arithmetic and handles complex-valued matrices. It requires that the input matrix be expressible as a sum of rank-1 updates, a condition automatically met in a LST receiver since individual channel estimates are available. It was implemented in a Virtex-II FPGA with the inclusion of a dynamic scaling circuit (using simple shifts) to improve precision and avoid underflow/overflow. Further optimisations are planned, as well as a detailed precision analysis.

Acknowledgment

This work was supported by the National Sciences and Engineering Research Council of Canada (NSERC) under its Discovery Grant Program. The support of the Canadian Microelectronics Corporation (CMC), under its System-On-Chip Research Network (SOCRN) program, is also gratefully acknowledged.

References

[1] A. El-Amawy, "A systolic architecture for fast dense matrix inversion," *IEEE Transactions On Computers*, vol. 38, no. 3, pp. 449–455, May 1989.

[2] F. Edman and V. Öwall, "A scalable pipelined complex valued matrix inversion architecture," in *IEEE International Symposium on Circuits and Systems*, Kobe, Japan, May 2005.

[3] M. Ylinen, A. Burian, and J. Takala, "Updating matrix inverse in fixed-point representation: Direct versus iterative methods," in *IEEE International Symposium on System-on-Chip*, Tampere, Finland, November 2003.

[4] Z. Liu, J. V. McCanny, G. Lightbody, and R. Walke, "Generic soc qr array processor for adaptive beamforming," *IEEE Transactions On Circuits and Systems - II: Analog and Digital*, vol. 50, no. 4, pp. 169–175, Avril 2003.

[5] G. Lightbody, R. Woods, and R. Walke, "Design of parameterizable silicon intellectual property core for qr-based rls filtering," *IEEE Transactions On Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 4, pp. 659–678, Août 2003.

[6] T. Shepherd and J. McWhirter, "Systolic Adaptive Beamforming," in *Radar Array Processing*, S. Haykin, J. Litva, and T. Shepherd, Eds. Springer-Verlag, 1993.

[7] G. H. Golub and C. F. V. Loan, *Matrix Computation*. John Hopkins University Press, 1991, second Edition.

[8] R. Döhler, "Squared givens rotation," *IMA Journal of Numerical Analysis*, vol. 11, pp. 1–5, 1991.

[9] W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic arrays," *SPIE Real-Time Signal Processing IV*, vol. 298, pp. 19–26, 1981.

[10] G. J. Foschini, "Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas," *Bell Labs Technical Journal*, pp. 41–59, Fall 1996.

[11] E. W. Weisstein, *Sherman-Morrison Formula*, From MathWorld–A Wolfram Web Resource. [Online]. Available: http://mathworld.wolfram.com/Sherman-MorrisonFormula.html