Efficient Complex Matrix Inversion for MIMO Software Defined Radio

Johan Eilert, Di Wu and Dake Liu
Department of Electrical Engineering, Linköping University, Sweden
Email: {je,diwu,dake}@isy.liu.se

***Abstract*— Complex matrix inversion is a very computationally demanding operation in advanced multi-antenna wireless communications. Traditionally, systolic array-based QR decomposition (QRD) is used to invert large matrices. However, the matrices involved in MIMO baseband processing in mobile handsets are generally small which means QRD is not necessarily efficient. In this paper, a new method is proposed using programmable hardware units which not only achieves higher performance but also consumes less silicon area. Furthermore, the hardware can be reused for many other operations such as complex matrix multiplication, filtering, correlation and FFT/IFFT.**

I. Introduction

Multi-antenna or multi-in and multi-out (MIMO) schemes have been discussed widely as a technology to greatly enhance the performance of wireless communications. However, they usually require massive computing power. Among the most intensive tasks, complex matrix inversion is one of the most complex and MIPS demanding part. As semiconductor process paves its way to deep-submicron (90–65 nm) and antenna technology advances, MIMO enhanced mobile handsets will enable high-speed wireless links in the near future. However, as these physical limits still exist, it is most probable that the number of antennas in mobile handsets will be limited within four in the near future. Actually, most of the MIMO related papers recently published are dealing with 4×4 or even smaller antenna matrices.

There have been various MIMO signaling schemes such as space-time coding, spatial multiplexing and beamforming exploiting different freedoms of multi-antenna system. Most of these algorithms require complex matrix inversion, e.g. the Zero-forcing (ZF) and minimum mean-squared error (MMSE) decoding. In order to achieve the best trade-off between multiplexing and diversity gain, different MIMO signaling schemes need to be adopted according to circumstances such as the channel model. Software-defined radio (SDR) is the concept of accommodating heterogeneous radio using a tunable RF front-end and a programmable baseband processor. This paper presents how to use an application specific instruction processor (ASIP) for SDR to efficiently implement complex matrix inversion for MIMO communications.

II. Matrix Inversion in MIMO

In a MIMO system, for the general transmission function

$$r = \mathbf{H}s + n$$

the channel estimator estimates the channel matrix $\mathbf{H}$, and the decoding methods such as ZF and MMSE performs matrix inversion involving $\mathbf{H}$ and the noise n before detecting s from r. For MIMO-OFDM, the same matrix inversion needs to be performed on a tone-by-tone basis. According to different OFDM standards, the number of tones ranges from 48 (WLAN) to 6817 (DVB-T) [1]. Thus matrix inversion is a very computationally intensive operation in MIMO-OFDM.

For practical MIMO signal processing, latency is a critical issue. How often matrix inversion is needed depends on the variation of the wireless channel. Taking a 4×4 (four sending antennas and four receiving antennas) MIMO-OFDM based DVB-H system as an example, 3409 subcarriers are used (the medium case for DVB-H) and the working frequency f is 1675 MHz. If the mobile handset is moving at speed v = 250km/h, then based on the following formula

$$f_m = \frac{vf}{c}$$

the maximum Doppler shift f_m is 385 Hz. The following formula given in [2],

$$T_c \approx \frac{9}{16\pi f_m}$$

estimates the channel coherence time T_c to be $465\ \mu s$. In this case, even if Zero-Forcing is used as the decoding scheme and interpolation is applied every 8 subcarriers, the number of matrices to be inverted is 427 during this time period. Thus the inversion of a 4×4 matrix as well as all other channel estimation computation needs to be finished within $1\mu s$. Even for an ASIC running at 500 MHz, the inversion of this 4×4 matrix needs to be completed within 150 cycles (30% of the total time budget). The goal of this paper is to find a practical solution that meets this performance requirement.

III. Classical Matrix Inversion

A. Algorithms

Matrix inversion is a traditional topic in the area of numerical analysis, and it is well elaborated in many publications, for example in the book by Golub and Van Loan [3].

For large matrices, matrix inversion is traditionally implemented by applying QR factorization to the original matrix to generate an upper triangular matrix R, then the result can be computed using back substitution. As mentioned in [3], there are several ways to compute QR decomposition, such as

Gram-Schmidt transform, Householder matrices, and Givens rotations which is also known as Jacobi rotations.

Recently, Squared Givens rotations (SGR) [4] has received attention for hardware implementation for QR decomposition. As mentioned by [6] and [7], it avoids the square-root operation and reduces the number of multiplications by half. At the same time, parallelism exists in SGR so that it can be mapped to parallel processing hardware to achieve higher performance.

B. HW Implementation

Systolic array is a classical architecture to implement QR decomposition for high performance solutions. There have been numerous contributions in this area such as the systolic array for SGR in [5]. However, traditional quadratic systolic arrays usually consume large silicon area and does not scale very well as the size of the matrix increases. Edman proposed a linear array that is more scalable than the traditional array [6]. In [7], Karkooti also presented a solution by combining similar nodes, adding memory blocks and a scheduler that controls the movement of data between nodes to avoid using a quadratic array. However, both of these solutions introduce heavy latency.

IV. The Selected Algorithms

A. Direct Analytic Matrix Inversion

One straightforward way to compute matrix inversion is the analytic approach, which is to generate the inverted matrix $\mathbf{H^{-1}}$ by calculating the adjugate matrix $(\mathbf{C_{ij}})^{\mathbf{T}}$ containing cofactors $\mathbf{C_{ji}}$ and the determinant $|\mathbf{H}|$ of the original matrix.

$$\mathbf{H^{-1}} = \frac{\mathbf{1}}{|\mathbf{H}|}(\mathbf{C_{ij}})^{\mathbf{T}} = \frac{\mathbf{1}}{|\mathbf{H}|}\begin{pmatrix} C_{11} & C_{21} & \dots & C_{j1} \\ C_{12} & \ddots & & C_{j2} \\ \vdots & & \ddots & \vdots \\ C_{1i} & \dots & \dots & C_{ji} \end{pmatrix}$$

For example, the inversion of a 2×2 matrix is computed as follows:

$$\mathbf{H^{-1}} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

The analytic approach lacks scalability because the computational complexity grows very quickly as the size of the matrix increases. However, the focus of this paper is matrix inversion in MIMO baseband processing for mobile handsets. As explained in Sec. I, the size of matrices to be processed is decided by the number of antennas. Thus only matrices not larger than 4×4 are considered here. In this case, the scalability of the algorithm is a less important issue. Furthermore, the number of arithmetic operations of analytic approach for small matrices is significantly smaller than QR decomposition which is an iterative procedure. This makes the analytic approach very efficient for computing the inverse of small matrices. Meanwhile, as explained in Sec. VI-A, it is easy to be mapped to programmable HW units.

However, direct analytic matrix inversion is sensitive to finite-length errors. During simulation, we found that for 4×4 matrices, the direct analytic approach is not very stable due to the large number of subtractions involved in the computation which might introduce cancellation. In other words, the finite-length error generated by the analytical approach will significantly affect the receiver performance in case the numerical representation does not have enough precision (or number of bits). In order to avoid this drawback of direct analytic matrix inversion, a slightly different method is considered in Sec. IV-B.

B. Blockwise Analytic Matrix Inversion

An alternative way to compute matrix inversion is to partition the matrix into four smaller matrices, and then compute the inverse based on computations on these smaller parts. For example, to compute the inverse of a4×4 matrix M, it is first divided into four submatrices

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

The inverse of M can be computed as:

$$\begin{bmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{bmatrix}$$

By computing the inverse of two 2×2 matrices and some additional 2×2 matrix computations, the inversion of M can be computed.

Compared to the direct analytic approach, this blockwise analytic matrix inversion is more stable due to the fewer number of subtractions. Therefore it requires fewer number of bits to keep the precision (e.g. in the above case, only 2×2 matrices are inverted instead of 4×4 matrices, where the risk of cancellation is much smaller than the direct analytic approach). As elaborated above, blockwise analytic matrix inversion is proposed by us as an alternation to classical QR decomposition to invert small matrices such as 4×4. For 2×2 and 3×3 matrices, the direct analytic approach is used.

V. Design Considerations

A. Numerical Representation

Whether to use floating-point or fixed-point as the numerical representation for baseband processing is a question difficult to answer. Traditionally, for applications such as RADAR, because of the large scale of the matrix and dynamic range of signals, floating-point is preferred. For mobile handsets, fixed-point is usually chosen for the sake of area efficiency. However, since matrix inversion algorithms are sensitive to finite-length effects, the numerical representation needs to be carefully decided. As can be seen, [7] uses floating-point while fixed-point is adopted by [6].

B. Simulation

Simulation was performed to find the best trade-off among different algorithms and numerical representations. The simulated system has four sending antennas and two receiving antennas using STBC with two time slots, the resulting matrix is 4×4. The sender modulates random binary data with QPSK

modulation. The channel is modeled as complex gain between all antennas and Gaussian noise. The decoder uses the MMSE algorithm to recover the symbols. As shown in the BER/SNR curves in Fig. 1, the blockwise analytic inversion is more robust to finite-length errors, and the 20 bit (14 bit mantissa, 6 bit exponent) floating-point representation brings the best receiver performance in this simulation. The curve labeled "double" in the diagram shows the result achieved with 64-bit IEEE "double precision" floating point, which in this case can be considered as "infinite" precision without finite-length errors.

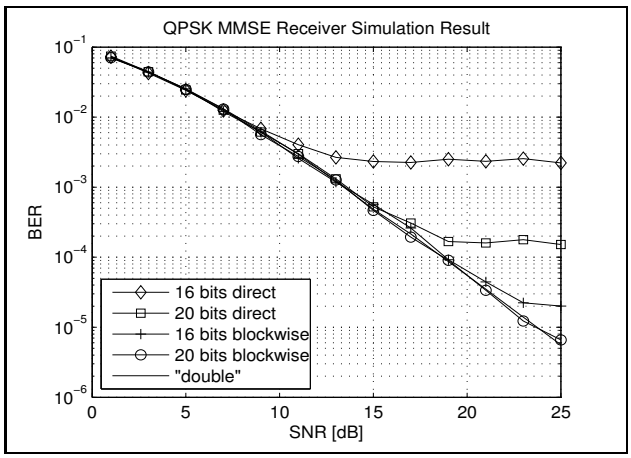

Fig. 1. Simulation with Finite-length Datatypes

VI. Architecture Design

A. HW Architecture

In order to meet the demand of baseband processing of various emerging wireless technology that utilize MIMO, an SDR baseband processor is being developed in the division of computer engineering, Linköping University. Similar to its predecessor [8], it can accommodate heterogeneous MIMO applications just by loading different programs. The architecture of the processor is illustrated by Fig. 2.

As depicted in Fig. 2, the datapath of this processor will include several Floating-Point Complex Multiply-ACcumulate (FPCMAC) units. In this paper, we propose an FPCMAC-based solution to compute matrix inversion for small matrices. Fig. 3 shows the combination of only one FPCMAC and a real divider, with 32 general and 8 accumulation registers which is enough to compute the inverse of matrices not larger than 4×4 with high efficiency.

B. Programming and Scheduling

Instructions such as CMUL, CMAC, ABS, RMUL and DIV are used to compute the matrix inversion. It is very important to schedule the algorithm so that the pipeline stalls caused by data dependency can be kept the minimum. Still it is rather straightforward to implement both the direct and blockwise analytic matrix inversion by programming the ASIP. For example, the assembly code in Table I shows how to compute the inverse of a 2×2 matrix.

```
cmul   Aa,Ad,AC0
cmac   -Ab,Ac,AC0,t0
(wait 2 cycles, unrelated
operations are here)
abs    t0,div
(wait 1-2 cycles, put
unrelated operations here)
rmul   ~t0,div,AC0
(wait 1 cycle, put an
unrelated operation here)
cmul   AC0,Ad,Xa
cmul   AC0,-Ac,Xb
cmul   AC0,-Ab,Xc
cmul   AC0,Aa,Xd
```

TABLE I
Code Example for 2×2 Matrix Inversion

VII. Implementation & Comparison

In order to compare with other solutions, the proposed datapath including the FPCMAC, the real divider, and the register file (32 general registers and 8 accumulation registers), has been implemented and synthesized using both FPGA and ASIC technologies. Based on the simulation result, two different wordlengths were chosen for Implementation A (16 bit) and B (20 bit).

A. FPGA

Xilinx ISE and Core Generator were used to synthesize the design based on the Virtex4-xc4vlx200 FPGA. The input is a 4×4 matrix of complex floating-point values and output is the inverse matrix. All the basic units such as the floating-point real adder, subtracter, multiplier and divider are generated using Xilinx Core Generator which enable the optimized design based on soft IP Core V.3 from Xilinx. All these operators are compatible with IEEE754 standard. Both the implementation

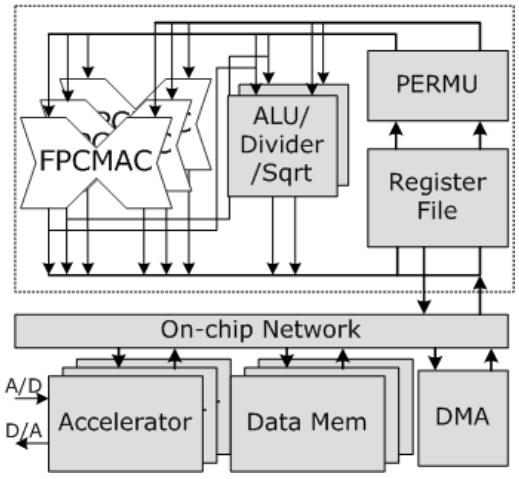

Fig. 2. Floating-Point Baseband SDR Processor

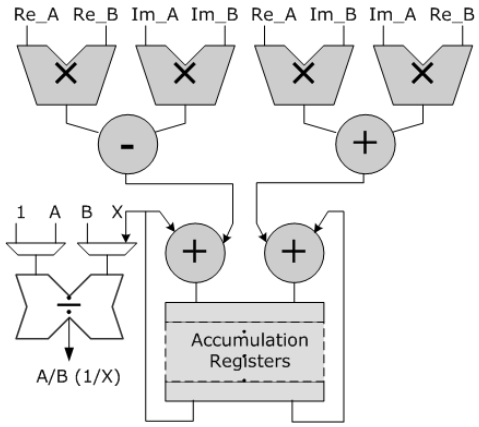

Fig. 3. Floating-Point Complex MAC Unit with A Real Divider

A and B have 9 pipeline stages in the FPCMAC and take 8 cycles to finish the real value division (1/X). Compared to the synthesis results of other solutions shown in Table II, our implementation B is almost three times faster than the linear array proposed by [6] and around seven times faster than the solution proposed by [7].

	Impl A	Impl B		Ref [6]	Ref [7]
FPGA Type	Virtex4	Virtex4		Virtex2	Virtex4
Datatype	floating	floating		fixed	floating
Wordlength (bits)	16	20		12	20
Num of Slices	1561	1716	2094	4400	9117
Num of DSP48	0	8	0	N/A	22
Latency (cycles)	120	120		350	933
Frequency (MHz)	125	100		100	115

TABLE II
FPGA IMPLEMENTATION RESULT

	Impl A	Impl B
Wordlength (bits)	16	20
Area (kgate)	23	43
Num of Pipeline Stages	3	3
Cycles for Division	2	3
Latency (cycles)	90	92
Working Frequency (MHz)	500	500

TABLE III
ASIC IMPLEMENTATION RESULT

B. ASIC

The ASIC synthesis is done using the Synopsys design flow and 90 nm process from ST Microelectronics. Table III depicts the different synthesized gate count, and working frequencies for various floating-point wordlength.

As depicted in Table III, our implementation A can easily run at 500 MHz and takes 90 cycles to compute the inverse of a 4×4 matrix. The implementation B can run at 500 MHz by having 3 stages in the divider. In this case, it requires 92 cycles to finish the same task. In other words, it can finished the inversion within 200 ns which is well enough for the DVB-H baseband processing example defined in Sec. II.

VIII. CONCLUSION

In this paper, we proposed a simple and efficient method to compute matrix inversion for small matrices, e.g. 4×4. This method is based on a programmable ASIP targeting SDR systems. According to the comparison with other solutions made in Sec. VII, our solution not only achieves higher performance but also consumes less silicon area. In addition, the ASIP is a purely programmable device driven by instructions, which has much higher flexibility for multi-standard baseband processing. It can be used for other normal complex vector computing such as complex filters, correlations, and FFT. For example, with a minimum of extra hardware (two adders), it can execute a 64-point FFT in around 200 clock cycles. Finally, for deep-submicron semiconductor process, it is important to shrink the chip area so that the leakage power can be minimized. By time-division hardware multiplexing (reuse), the ASIP achieves not only silicon efficiency but also power efficiency.

IX. ACKNOWLEDGMENT

The authors would like to thank D. Wang (UT Dallas) for the discussion in MIMO theory and algorithms. This work is supported by the Swedish National Foundation of Strategic Research SSF.

REFERENCES

[1] M. Borgmann and H. Bölcskei, *"Interpolation-based Efficient Matrix Inversion for MIMO-OFDM receivers"*, Proc. 38th Asilomar conf. Signals, Systems, and Computers, 2004.
[2] T. Rappaport, *"Wireless Communications: Principles and Practice, 2nd Edition"*, Prentice Hall, 2001
[3] G. H. Golub, C. F. Van Loan, *"Matrix Computations, Third Edition"*, The Johns Hopkins University Press, 1996.
[4] R. Döhler, *"Squared Givens Rotation"*, IMA Journal of Numerical Analysis, no. 11, pp. 15, 1991.
[5] J. Gotze and U. Schwiegelshohn, *"A Square Root and Division Free Givens Rotation for Solving Least Square Problems on Systolic Arrays"*, J. SCI. STAT. COMPUT., vol. 12, pp. 800-807, July 1991.
[6] F. Edman, V. Öwall, *"A Scalable Pipelined Complex Valued Matrix Inversion Architecture"*, Proc. ISCAS'05, 2005.
[7] M. Karkooti, J. Cavallaro, C. Dick, *"FPGA Implementation of Matrix Inversion Using QRD-RLS Algorithm"*, Proc. 39th Asilomar Conference on Signals, Systems, and Computers, 2005.
[8] A. Nilsson, E. Tell, D. Liu, *"Simultaneous Multi-standard Support in Programmable Baseband Processors"*, Proc. IEEE PRIME, 2006.