High-Throughput FPGA Implementation of Matrix Inversion for Control Systems

Xiao-Wei Zhang, Lei Zuo, *Member, IEEE*, Ming Li, *Member, IEEE*, and Jian-Xin Guo

***Abstract*—In control engineering, numerical stability and real time are the two most important requirements for the matrix inversion. This article presents an efficient and robust method for the field-programmable gate array (FPGA) calculation of the matrix inversion. We initially consider the scenario that the matrix to be processed is a nonsingular Hermitian matrix. The proposed computation procedures are composed of the matrix decomposition, triangular matrix inversion, and matrices multiplication. The first procedure is completed by LDL factorization based on the outer form of Cholesky's method, whereas the recursive algorithm for block submatrices is adopted to achieve the triangular matrix inversion. The new method has the high level in the parallel pipelining mechanism and steals the characteristics of both the upper triangular matrix and its inversion to reduce the computation load and improve the numerical stability. Furthermore, the non-Hermitian matrix inversion can be easily solved if another procedure is added in the new method. Finally, we compare our method with the exiting FPGA-based techniques on one Xilinx Virtex-7 XC7VX690T FPGA. Meanwhile, it has solved one array antenna control problem of the adaptive digital beam forming for one phased array radar successfully.**

***Index Terms*—Field-programmable gate array (FPGA), Hermitian matrix, LDL factorization, matrix inversion.**

I. Introduction

THE matrix inversion has wide applications in the industrial and electronic engineering, such as the model predictive control (MPC) [1], [2], wireless communication [3], [4], power systems [2], [5], network [6], and phased array radar [7]–[9] to name a few. We have known that the matrix inversion is very expensive on the precious hardware resources and less stable in numerical calculation. Thus, researchers often modify the algorithms used in their applications or find new methods in order to avoid the matrix inversion. In some fields, engineers have no choice to solve the problem of achieving the matrix inversion in hardware. For example, we have to realize the hardware implementation of some algorithms containing the matrix inversion in the autocontrol systems and phased array radar. In fact, the matrix inversion is one mathematical tool to solve the different optimization problems in some application fields. Two reasons why we discuss this item in the two fields are that they are confronted with the high-dimensional matrix inversion and that real-time and numerical stability are the two significant technical requirements in control engineering.

Algorithmically, the calculation methods of matrix inversion can be divided into three categories: definition, iterative, and matrix decomposition. If the matrix size is small, such as 2×2, 3×3, the standard definition method is one good choice. Due to the propagation of roundoff errors [10], the definition method for the high-dimensional matrix shares the big computation load and is generally less stable in numerical calculation. The iterative methods first require an initial estimate of the solution and subsequent update based on the calculation of previous estimation errors, which are not particularly suitable for the real-time hardware implementation. Some preferred methods based on matrix decomposition (Cholesky [11], LDL [12], [13], and QR [14]) in engineering are stable in numerical calculation. After matrix decomposition, the triangular matrices often use either equation solving or back substitution to achieve the matrix inversion. In [11], the relationship of diagonal elements between the triangular matrix and its inversion is used to avoid the computation of intermediate results. However, there is no deduction process. Zhang *et al*. [15] proposed the method based on the Neumann series expansion, which is just suited for the banded matrix. As shown in [16], they designed a simple positive definite symmetric matrix inversion (SPMI) algorithm based on the definition of the block matrix inversion. The SPMI algorithm is still less stable for the high-dimensional matrices in numerical calculation and there is no discussion about this. The method presented in [17] is not suited for the common matrix. The reason is that some intermediate matrices are constant in their application. We refer to the work in [18] and [19] for details of the latest implementation of the large-scale matrix inversion based on the Gaussian–Jordan algorithm being one modified

Manuscript received December 4, 2019; revised February 18, 2020, March 22, 2020, and April 21, 2020; accepted May 3, 2020. Date of publication May 28, 2020; date of current version March 22, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61871307, in part by the Natural Science Foundation Research Project of Shaanxi Province of China under Grant 2018JM6098, and in part by the Shaanxi Key Laboratory of Integrated and Intelligent Navigation Open Fund under Grant SKLIN-20180211. *(Corresponding author: Lei Zuo.)*

Xiao-Wei Zhang is with the School of Information Engineering, Xijing University, Xi'an 710123, China, and also with the National Laboratory of Radar Signal Processing, Xidian University, Xi'an 710126, China (e-mail: xwzhang@stu.xidian.edu.cn).

Lei Zuo and Ming Li are with the National Laboratory of Radar Signal Processing, Xidian University, Xi'an 710126, China (e-mail: lzuo@mail.xidian.edu.cn; liming@xidian.edu.cn).

Jian-Xin Guo is with the School of Information Engineering, Xijing University, Xi'an 710123, China (e-mail: 1592124551@qq.com).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIE.2020.2994865

form of the LU decomposition, which is not advised to be adopted in engineering owing to its numerical instability and heavy computation load.

In this article, we present an efficient and robust method for the field-programmable gate array (FPGA) calculation of the matrix inversion. For a square matrix $\mathbf{B}$ of size $N \times N$, we can construct the Hermitian matrix $\mathbf{A}$ as $\mathbf{A} = \mathbf{B}\mathbf{B}^H$ where ^{H} denotes the complex conjugation. Once the inverse of $\mathbf{A}$ is solved, we can get $\mathbf{B}^{-1}$ by $\mathbf{B}^{-1} = \mathbf{B}^H\mathbf{A}^{-1}$. Thus, the Hermitian matrix plays an important role in the square matrix inversion and we discuss the calculation of the nonsingular Hermitian matrix inversion in detail. Now, the proposed method can be simply described as follows.

1) To get the upper triangular matrix, the Hermitian matrix is decomposed by LDL factorization based on the outer form of Cholesky's method.
2) The upper triangular matrix inversion is achieved by the recursive algorithm for block submatrices.
3) Finally, the Hermitian matrix inversion is the product of three matrices.

The main contributions of this article are as follows.

1) The characteristics of both the upper triangular matrix and its inversion are exploited to reduce the computation load and improve the numerical stability.
2) Our proposed method has the better programmability on the hardware platform and the higher level in parallel computation.
3) Numerical stability of the different methods is discussed in this article.

It is well known that the numerical stability is very important for the matrix inversion in engineering. The method based on QR factorization is suited for any square matrix but less stable in numerical calculation. In this article, we conduct some experiments to illustrate the performance of the different methods. The radar has the higher requirements of both the real-time and numerical stability than the general autocontrol systems, so we take one control problem for the adaptive digital beam former in one phased array radar as an example.

The reminder of this article is organized as follows. Section II presents the proposed matrix inversion method. We introduce the hardware architecture implementation of the new method on the Xilinx FPGA in Section III. The experimental results are described in Sections IV and V. Finally, Section VI concludes this article.

II. Proposed Method

This section presents the proposed method for the Hermitian matrix inversion whose calculation procedures contain the following three parts: the LDL factorization, triangular matrix inversion, and matrices multiplication.

A. LDL Factorization

For a nonsingular Hermitian matrix, $\mathbf{A} \in \mathbb{C}^{N\times N}$, it can be decomposed as $\mathbf{A} = \mathbf{R}^H\mathbf{D}\mathbf{R}$, where $\mathbf{R}$ and $\mathbf{D}$ are an upper triangular matrix and a diagonal matrix, respectively. Here, the nonsingular Hermitian matrix $\mathbf{A}$ is decomposed by LDL factorization based on the outer form of Cholesky's method. Hence, $\mathbf{A}$ can be expressed in the form of

$$\mathbf{A}_N = \begin{bmatrix} a_{11} & \mathbf{b}^H \\ \mathbf{b} & \mathbf{A}_{N-1} \end{bmatrix} \tag{1}$$

where $\mathbf{b} \in \mathbb{C}^{(N-1)\times 1}$ is a vector and $\mathbf{A}_{N-1} \in \mathbb{C}^{(N-1)\times(N-1)}$ is the submatrix. Now, we can derive the LDL factorization of (1) as

$$\begin{bmatrix} a_{11} & \mathbf{b}^H \\ \mathbf{b} & \mathbf{A}_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \mathbf{s} & \mathbf{R}_{N-1} \end{bmatrix} \begin{bmatrix} d_{11} & 0 \\ 0 & \mathbf{D}_{N-1} \end{bmatrix} \begin{bmatrix} 1 & \mathbf{s}^H \\ 0 & \mathbf{R}_{N-1} \end{bmatrix}. \tag{2}$$

Equating the blocks, we obtain

$$a_{11} = d_{11} \tag{3a}$$

$$\mathbf{b} = d_{11}\mathbf{s} \tag{3b}$$

$$\mathbf{A}_{N-1} = d_{11}\mathbf{s}\mathbf{s}^H + \mathbf{R}_{N-1}^H\mathbf{D}_{N-1}\mathbf{R}_{N-1}. \tag{3c}$$

Equations in (3) suggest the following procedures for calculating d_{11}, $\mathbf{s}$, $\mathbf{D}_{N-1,\text{new}}$, and $\mathbf{R}_{N-1,\text{new}}$:

$$d_{11} = a_{11} \tag{4a}$$

$$\mathbf{s} = \mathbf{b}/a_{11} \tag{4b}$$

$$\mathbf{A}_{N-1,\text{new}} = \mathbf{A}_{N-1} - a_{11}\mathbf{s}\mathbf{s}^H = \mathbf{A}_{N-1} - \mathbf{b}\mathbf{s}^H \tag{4c}$$

$$\text{Solve } A_{N-1,\text{new}} = \mathbf{R}_{N-1,\text{new}}^H\mathbf{D}_{N-1,\text{new}}\mathbf{R}_{N-1,\text{new}} \text{ for } \mathbf{D}_{N-1,\text{new}} \text{ and } \mathbf{R}_{N-1,\text{new}}. \tag{4d}$$

This procedure reduces the $N \times N$ problem to that of finding the Cholesky factor of the $(N-1) \times (N-1)$ matrix $\mathbf{A}_{N-1,\text{new}}$. Then, LDL factorization can be step by step achieved by the aforementioned algorithm. In the matrix decomposition, it can be observed that the outer form of Cholesky's method has an advantage in parallel computation. The diagonal elements of $\mathbf{R}$ are 1, so there is no need to compute its diagonal elements.

Cholesky's method can be solved by three algorithms: the inner-product form, outer-product form, and bordered form. The three algorithms actually have the same computation load and flop-counts, but the outer form has the best performance in parallel calculation. However, the LDL factorization based on Cholesky's method is not carefully discussed in those works [11]–[13] in which the inner-product form is often presented. Meanwhile, they did not illustrate how to realize the LDL factorization on the FPGA, which will be illustrated in the next section.

B. Triangular Matrix Inversion

After LDL factorization, we should consider how to realize the upper triangular matrix inversion. Traditionally, either equation solving or back substitution is often adopted to solve the triangular matrix inversion but does not have the good advantage in parallel calculation. We here give the proof for the conclusion described in [11] and then show that one more characteristic of both the upper triangular matrix and its inversion can also be used to reduce the computation load. Finally, the recursive algorithm for block submatrices is adopted to achieve the upper triangular matrix inversion.

Now, we show the derivation process of the conclusion described in [11]. For an upper triangular matrix $\mathbf{R}$, it is

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1N} \\ 0 & r_{22} & \dots & r_{2N} \\ \vdots & 0 & r_{ii} & \vdots \\ 0 & 0 & \cdots & r_{NN} \end{bmatrix} \tag{5}$$

where r_{ij} is the nonzero element of $\mathbf{R}$ and $1 \le i \le j \le N$. Based on the definition of matrix inversion, the inverse of $\mathbf{R}$ can be written as

$$\mathbf{R}^{-1} = \frac{1}{\det(\mathbf{R})}\mathrm{adj}(\mathbf{R}) = \frac{1}{|\mathbf{R}|}\begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{21} & \dots & \mathbf{R}_{N1} \\ \mathbf{R}_{12} & \mathbf{R}_{22} & \dots & \mathbf{R}_{N2} \\ \vdots & \vdots & \dots & \vdots \\ \mathbf{R}_{1N} & \mathbf{R}_{2N} & \dots & \mathbf{R}_{NN} \end{bmatrix} \tag{6}$$

where $\det(\cdot)$ and $\mathrm{adj}(\cdot)$ denote the matrix determinant and adjoint matrix, respectively; $\mathbf{R}_{ij}$is the matrix cofactor of $\mathbf{R}$. We have known that the elements r_{ij} of $\mathbf{R}$ are equal to 0 when $i > j$. Therefore, we can get the following equations by the definition of the matrix cofactor:

$$\mathbf{R}_{ij} = 0, \quad i < j \tag{7a}$$

$$\mathbf{R}_{ij} \neq 0, \quad i \ge j. \tag{7a}$$

Equations in (7) suggest that $\mathbf{R}^{-1}$ can be rewritten as

$$\mathbf{R}^{-1} = \frac{1}{\det(\mathbf{R})}\mathrm{adj}(\mathbf{R}) = \frac{1}{|\mathbf{R}|}\begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{21} & \dots & \mathbf{R}_{N1} \\ 0 & \mathbf{R}_{22} & \dots & \mathbf{R}_{N2} \\ \vdots & 0 & \dots & \vdots \\ 0 & \dots & 0 & \mathbf{R}_{NN} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\mathbf{R}_{11}}{|\mathbf{R}|} & \frac{\mathbf{R}_{21}}{|\mathbf{R}|} & \dots & \frac{\mathbf{R}_{1N}}{|\mathbf{R}|} \\ 0 & \frac{\mathbf{R}_{22}}{|\mathbf{R}|} & \dots & \frac{\mathbf{R}_{2N}}{|\mathbf{R}|} \\ \vdots & 0 & \dots & \vdots \\ 0 & \dots & 0 & \frac{\mathbf{R}_{NN}}{|\mathbf{R}|} \end{bmatrix} = \begin{bmatrix} \hat{r}_{11} & \hat{r}_{12} & \dots & \hat{r}_{1N} \\ 0 & \hat{r}_{22} & \dots & \hat{r}_{2N} \\ \vdots & 0 & \dots & \vdots \\ 0 & \dots & 0 & \hat{r}_{NN} \end{bmatrix} \tag{8}$$

where $\hat{r}_{ij}$ is the nonzero element of $\mathbf{R}^{-1}$. Next, we analyze the diagonal elements of $\mathbf{R}^{-1}$. Based on the definition of the matrix cofactor, $\hat{r}_{ii}$ is computed by

$$\hat{r}_{ii} = \mathbf{R}_{ii}/\det(\mathbf{R}). \tag{9}$$

Because $\det(\mathbf{R}) = \prod_{i=1}^{N} r_{ii}$ and $\mathbf{R}_{ii} = \prod_{n=1,n\neq i}^{N} r_{nn}$where $\prod(\cdot)$ means the quadrature operation, we can immediately obtain the following results:

$$\hat{r}_{ii} = \left(\prod_{n=1,n\neq i}^{N} r_{nn}\right) / \left(\prod_{i=1}^{N} r_{ii}\right) = 1/r_{ii}. \tag{10}$$

Using the relationships described in (10), the back substitution is often adopted to achieve the triangular matrix inversion. For the upper triangular matrix obtained by LDL factorization, we can get the following equation: $\hat{r}_{ii} = r_{ii} = 1$. In [11], the authors use the relationship of diagonal elements between the triangular matrix and its inversion to compute the matrix inversion.

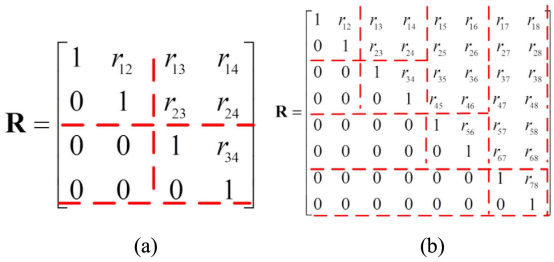

Fig. 1. Division of upper triangular matrices. (a) Matrix size is 4 × 4. (b) Matrix size is 8 × 8.

For an upper triangular matrix obtained by LDL factorization, the relationship between $\hat{r}_{i(i+1)}$ in $\mathbf{R}^{-1}$ and $r_{i(i+1)}$ in $\mathbf{R}$ can still be used for reducing the computation load in fact. Based on the definition of the matrix inversion, $\hat{r}_{i(i+1)}$ is computed by

$$\begin{aligned} \hat{r}_{i(i+1)} &= \mathbf{R}_{(i+1)i}/\det(\mathbf{R}) \\ &= \left((-1)^{2i+1} r_{i(i+1)} \prod_{n=1,n\neq i}^{N} r_{nn}\right) / \left(\prod_{n=1}^{N} r_{nn}\right) \end{aligned} \tag{11}$$

where $r_{nn} = 1$ and both $\prod_{n=1}^{N} r_{nn}$and $\prod_{n=1,n\neq i}^{N} r_{nn}$ are equal to 1. Thus, (11) is simply expressed as

$$\hat{r}_{i(i+1)} = (-1)^{2i+1} r_{i(i+1)} = -r_{i(i+1)}. \tag{12}$$

Now, the inverse of an upper triangular matrix obtained by LDL factorization $\mathbf{R}^{-1}$ can be rewritten as

$$\mathbf{R}^{-1} = \begin{bmatrix} 1 & -r_{12} & \hat{r}_{13} & \dots & \hat{r}_{1N} \\ 0 & 1 & -r_{23} & \dots & \vdots \\ 0 & 0 & 1 & \dots & \hat{r}_{3N} \\ 0 & 0 & \dots & \vdots & -r_{(N-1)N} \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix}. \tag{13}$$

We here can say that the number of nonzero elements to be computed in $\mathbf{R}^{-1}$ is 0.5 × (N^2–3N+2), because its diagonal elements are equal to 1 and $\hat{r}_{i(i+1)}$ is the reverse of $r_{i(i+1)}$.

Now, we consider the solution of the upper triangular matrix inversion. In general, the standard approach for the triangular matrix inversion is either back substitution or equation solving that does not have the big advantage in parallel calculation. In other words, they are not meet with the scenarios requiring real time. Thus, we propose the recursive algorithm for block submatrices to calculate the upper triangular matrix inversion. Similar to the SPMI proposed in [16], the new algorithm is also based on the block submatrices inversion. However, our algorithm has the lower computation load but is more stable in numerical calculation. Two reasons can be described as follows: the two characteristics described in (10) and (12) are stolen to reduce the computation load and our algorithm is based on the matrix decomposition. In Fig. 1, we first take the two matrices as an example.

Before analyzing Fig. 1, we consider that an upper triangular matrix size is 2 × 2. Immediately we write the upper triangular

matrix and its inversion as follows:

$$\mathbf{R} = \begin{bmatrix} 1 & r_{12} \\ 0 & 1 \end{bmatrix} \tag{14a}$$

$$\mathbf{R}^{-1} = \begin{bmatrix} 1 & -r_{12} \\ 0 & 1 \end{bmatrix}. \tag{14b}$$

From (14), it can be seen that there is just one step of the sign bit inversion for r_{12} in the calculation procedure of $\mathbf{R}^{-1}$. Of course, the aforementioned example is one special matrix. In Fig. 1, the upper triangular matrix is first divided into many small size submatrices. Then, the block submatrices inversion is used to solve the whole upper triangular matrix inversion. Xu *et al.*, [16] have shown that the SPMI algorithm has the better performance than Cholesky's method in the running time. However, SPMI is less stable in numerical computation when the matrix inversion is directly computed without the matrix decomposition [10]. Numerical stability is still important in engineering, whereas they did not say anything about this.

From Fig. 1, it can be observed that the recursive algorithm is step by step computing the small submatrices that are divided by 2×2 along the diagonal elements. Our calculation procedures are similar to the SPMI, whereas our algorithm has its own features. Here, we will discuss its one intermediate step in detail. For an upper triangular matrix, its block version can be expressed as

$$\mathbf{R}_B = \begin{bmatrix} \mathbf{C} & \mathbf{E} \\ \mathbf{O} & \mathbf{G} \end{bmatrix} \tag{15}$$

where $\mathbf{R}_B \in \mathbb{C}^{(m+p)\times(m+p)}$, $\mathbf{C} \in \mathbb{C}^{m\times m}$, $\mathbf{E} \in \mathbb{C}^{m\times p}$, $\mathbf{O} \in \mathbb{C}^{p\times m}$, and $\mathbf{G} \in \mathbb{C}^{p\times p}$. Based on the definition of block matrix inversion, we can get the following equations:

$$\mathbf{R}_B^{-1} = \begin{bmatrix} \mathbf{C}^{-1} & -\mathbf{C}^{-1}\mathbf{E}\mathbf{G}^{-1} \\ \mathbf{O} & \mathbf{G}^{-1} \end{bmatrix} \tag{16}$$

where $\mathbf{C}^{-1}$ is the previous calculation result; $\mathbf{G}$ is expressed in the form of $\left[\begin{smallmatrix} 1 & x \\ 0 & 1 \end{smallmatrix}\right]$ and x is the nonzero element. It should be pointed out that $\mathbf{G}^{-1}$ is easily solved by (14b).

Next, we analyze the expression of $-\mathbf{C}^{-1}\mathbf{E}\mathbf{G}^{-1}$ in (16) for Fig. 1(b). The following equations describe the second step of the block submatrices inversion:

$$-\mathbf{C}^{-1} = \begin{bmatrix} -1 & -\hat{r}_{12} & -\hat{r}_{13} & -\hat{r}_{14} \\ 0 & -1 & -\hat{r}_{23} & -\hat{r}_{24} \\ 0 & 0 & -1 & -\hat{r}_{34} \\ 0 & 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -1 & r_{12} & -\hat{r}_{13} & -\hat{r}_{14} \\ 0 & -1 & r_{23} & -\hat{r}_{24} \\ 0 & 0 & -1 & r_{34} \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{17a}$$

$$\mathbf{E} = \begin{bmatrix} r_{16} & r_{26} & r_{36} & r_{46} \\ r_{15} & r_{25} & r_{35} & r_{46} \end{bmatrix}^T \tag{17b}$$

$$\mathbf{G}^{-1} = \begin{bmatrix} 1 & -r_{(i+2)(i+3)} \\ 0 & 1 \end{bmatrix}. \tag{17c}$$

Based on (17), we can immediately obtain the following results:

$$-\mathbf{C}^{-1}\mathbf{E} = \begin{bmatrix} -1 & r_{12} & -\widehat{r}_{13} & -\widehat{r}_{14} \\ 0 & -1 & r_{23} & -\widehat{r}_{24} \\ 0 & 0 & -1 & r_{34} \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} r_{15} & r_{16} \\ r_{25} & r_{26} \\ r_{35} & r_{36} \\ r_{45} & r_{46} \end{bmatrix} = \begin{bmatrix} r'_{15} & r'_{16} \\ r'_{25} & r'_{26} \\ r'_{35} & r'_{36} \\ r'_{45} & r'_{46} \end{bmatrix} \tag{18a}$$

$$-\mathbf{C}^{-1}\mathbf{E}\mathbf{G}^{-1} = \begin{bmatrix} r'_{15} & r'_{16} \\ r'_{25} & r'_{26} \\ r'_{35} & r'_{36} \\ r'_{45} & r'_{46} \end{bmatrix} \begin{bmatrix} 1 & -r_{(i+2)(i+3)} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r'_{14} & -r'_{15}r_{(i+2)(i+3)} + r'_{16} \\ r'_{25} & -r'_{25}r_{(i+2)(i+3)} + r'_{26} \\ r'_{35} & -r'_{35}r_{(i+2)(i+3)} + r'_{36} \\ r'_{45} & -r'_{45}r_{(i+2)(i+3)} + r'_{46} \end{bmatrix}. \tag{18b}$$

For (18a), there are many "0" and "–1" in $-\mathbf{C}^{-1}$, but the authors suggest that the implementation of (18a) be followed by the definition of matrices multiplication. The reason is that we can easily write programs in parallel calculation. However, there is no need to compute the first column vector of (18b).

Finally, we simply conclude the recursive algorithm for the upper triangular matrix inversion as follows: First, we subdivide the upper triangular matrix into many small submatrices, such as in Fig. 1; Second, the recursive algorithm for the block submatrices is then used to solve the whole matrix inversion. In this procedure, the three characteristics in (10), (12), and (14) are exploited to calculate$-\mathbf{C}^{-1}$ and $\mathbf{G}^{-1}$. Meanwhile, there is no need to compute the first column vector of (18b) deleting the redundant operations. Overall, the more characteristics used in this calculation procedure are conducive to improving the numerical stability.

C. Matrix Inversion and Complexity

In this part, the ultimate procedure is computing the product of three matrices to obtain the whole matrix inversion. For the matrix $\mathbf{A}$ as$\mathbf{A} = \mathbf{R}^H\mathbf{D}\mathbf{R}$, when the inverse of $\mathbf{R}$ is solved, we can compute the inverse of $\mathbf{A}$ by

$$\mathbf{A}^{-1} = (\mathbf{R}^H\mathbf{D}\mathbf{R})^{-1} = \mathbf{R}^{-1}\mathbf{D}^{-1}\mathbf{R}^{-H}. \tag{19}$$

To the best our knowledge, there is no shortcut to compute the product of three matrices in (19). Finally, the total number of different operations for Cholesky's method, the SPMI algorithm, and our method is listed in Table I. From Table I, we can see that our method has the least operations number.

D. Hardware Arithmetic and Numerical Stability

In general, the algorithms implemented on FPGAs are preferred in fixed-point arithmetic. Fixed-point two's complement representation are used for algorithms with smaller requirements

TABLE I
TOTAL NUMBER OF DIFFERENT OPERATIONS FOR THE THREE METHODS

Method	Add/Sub	Multiply	Division	Square Root
Cholesky	$0.5(N^3-N^2)$	$0.5(N^3+N^2-2N)$	N	N
SPMI	$0.5(N^3-N^2-N)$	$0.5(N^3+N^2-N)$	$0.5N$	0
Our method	$0.5(N^3-3N)$	$0.25(2N^3+N^2-2N)$	$2N$	0

of dynamic range, whereas floating-point representation are used where the dynamic requirements are larger. Some researchers [20], [21] have discussed the matrix inversion implemented in fixed- and floating-point arithmetic. We advise that the matrix inversion be performed in floating-point arithmetic, though it requires the more precious hardware resources. Consider the following matrix:

$$\mathbf{A} = \begin{bmatrix} 1000 & 999 \\ 999 & 998.01 \end{bmatrix}. \tag{20}$$

If the inversion of (20) is calculated on FPGAs in single floating-point arithmetic, we can get the result as follows:

$$\mathbf{A}^{-1} = \begin{bmatrix} 111.6672 & -111.7780 \\ -111.7780 & 111.8899 \end{bmatrix}. \tag{21a}$$

For 24-b fixed-point arithmetic, its inversion calculated by FPGAs is

$$\mathbf{A}^{-1} = \begin{bmatrix} 998 & -999 \\ -999 & 1000 \end{bmatrix}. \tag{21b}$$

From (21), we can see that for the fixed-point representation, the calculation result is not correct due to the finite word length, though it can save the hardware resource. In practice, we may not be confronted with the close singular matrix. For the sake of safety, the single floating-point arithmetic (IEEE 754-1985 standard) is strongly suggested to be adopted in engineering.

Meanwhile, numerical stability of the matrix inversion algorithms is also important for engineering. First, we consider the famous ill-conditioned Hilbert matrices defined by $h_{ij} = 1/(i+j-1)$. For example

$$\mathbf{H} = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 \\ 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 \end{bmatrix}. \tag{22}$$

For (22), we will discuss its inversion calculated in MATLAB. If we type A = single(hilb(7)) in MATLAB to get (22) and compute its inversion by Inv_A = inv(A), the sentence "Matrix is close to singular or badly scaled." will be printed in the command window. If A is decomposed by LDL factorization and the inverse of the triangular matrix is computed by the inv function, we will not see this warning. The inv function in MATLAB is achieved by LU factorization, whereas we strongly advise that the matrix inversion should be performed by Cholesky's method or LDL factorization in engineering.

Algorithm 1: Recursive Block Submatrices Algorithm Based on LDL Factorization.

Input: One nonsingular Hermitian matrix $\mathbf{A} \in \mathbb{C}^{N\times N}$
Output: The inverse of matrix $\mathbf{A}^{-1}$

For i = 1→ N, do $\mathbf{A} = \mathbf{R}^H\mathbf{D}\mathbf{R} = \begin{bmatrix} a_{ii} & \mathbf{b}^H \\ \mathbf{b} & \mathbf{A}_{N-1} \end{bmatrix}$
  $d_{ii} = a_{ii}$
  If $a_{ii} \simeq 0$ set error flag, exit
  else $\mathbf{s} = \frac{\mathbf{b}}{a_{ii}} = \frac{1}{a_{ii}} \times \mathbf{b}$
  $\mathbf{A}_{N-1,\text{new}} = \mathbf{A}_{N-1} - a_{ii}\mathbf{s}\mathbf{s}^H = \mathbf{A}_{N-1} - \mathbf{b}\mathbf{s}^H$
End For
For k = 1→ N/2, do $\mathbf{R}^{-1} = \begin{bmatrix} \mathbf{C}^{-1} & -\mathbf{C}^{-1}\mathbf{E}\mathbf{G}^{-1} \\ \mathbf{O} & \mathbf{G}^{-1} \end{bmatrix}$
  $\mathbf{R}_{T1} = -\mathbf{C}^{-1}$
  $\mathbf{R}_{T2} = -\mathbf{C}^{-1}\mathbf{E} = \mathbf{R}_{T1}\mathbf{E}$
  $\mathbf{R}_{T3} = -\mathbf{C}^{-1}\mathbf{E}\mathbf{G}^{-1} = \mathbf{R}_{T2}\mathbf{G}^{-1}$
End For
$\mathbf{T} = \mathbf{R}^{-1}\mathbf{D}^{-1}$
$\mathbf{A}^{-1} = \mathbf{T}\mathbf{R}^{-H}$

Actually, Watkins [10] has proved that the stability of Cholesky's method for a positive definite matrix is stable in practice. Meanwhile, the most comprehensive work on this subject is Higham's book, *Accuracy and Stability of Numerical Algorithms* [22].

III. HARDWARE ARCHITECTURE IMPLEMENTATION

This section presents the implementation of the hardware architecture for the matrix inversion. We first provide a top-level description of the proposed method in Section III-A. Then, we show the hardware architecture for the LDL factorization, triangular matrix inversion, and matrices multiplication in the following sections, respectively.

A. Top-Level Description

In this part, the top-level description for calculating the inverse of the nonsingular Hermitian matrix $\mathbf{A}$ is shown in Algorithm 1. To sum up, the pseudocode for the proposed method contains two for-loops and the matrices multiplication.

There are two considerations in the proposed method that must be taken into the hardware implementation, which are as follows.

1) If $a_{ii} = 0$ or a_{ii} is equivalent to 0 (due to the computation accuracy of the hardware platform), the proposed method will stop the calculation and set the error flag, which is very important in engineering. The reason is that we may encounter the ill-conditioned matrices in practice. However, the related literatures did not consider this scenario.
2) We begin to calculate the inverse of the upper triangular matrix once obtaining the first several row vectors of the LDL factorization results in the hardware implementation.

Remark: The second for-loop of our method is similar to the SPMI algorithm, but there are also some differences between

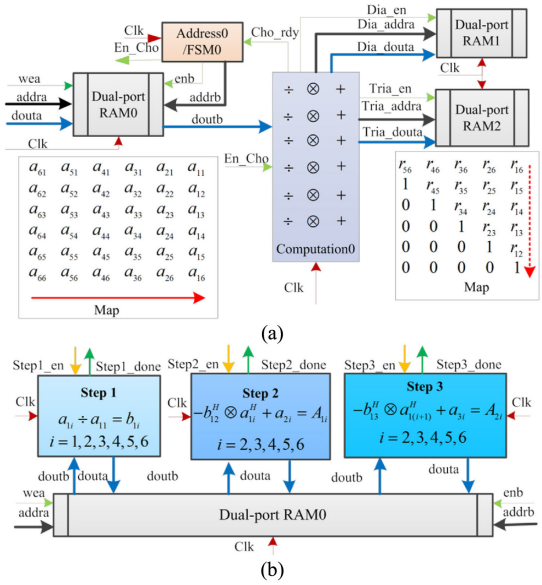

Fig. 2. Hardware architecture for LDL factorization. (a) Modules of LDL factorization. (b) First three steps of LDL factorization and the data stream.

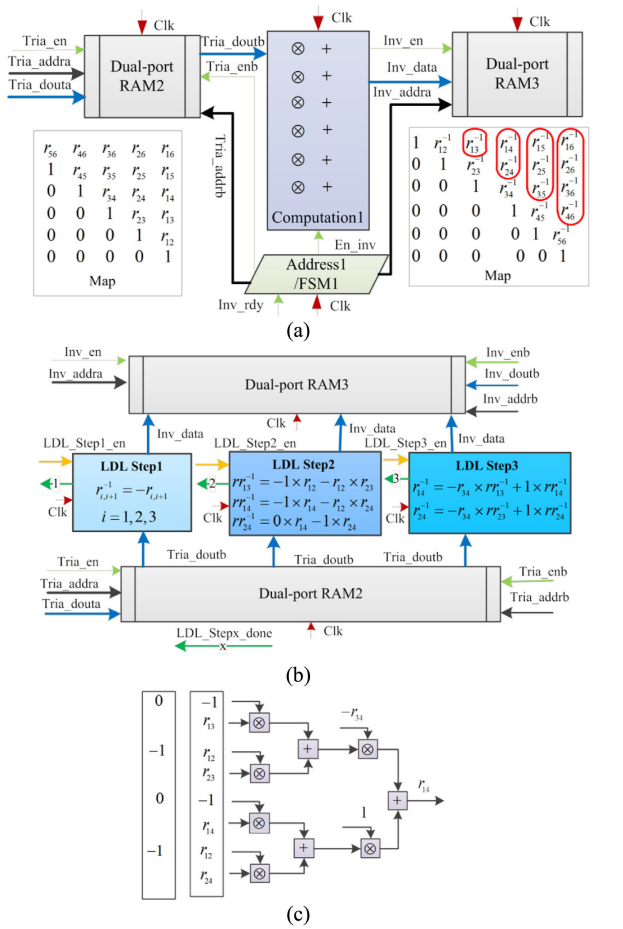

Fig. 3. Hardware architecture for upper triangular matrix inversion. (a) Modules of the upper triangular matrix inversion. (b) First three steps of the triangular matrix inversion. (c) PE.

the two methods. In our method, the characteristics of both the upper triangular matrix and its inversion are stolen to delete the redundant operations to improve the numerical stability and reduce the computation load, whereas the SPMI is just a simple fixed form of block matrix inversion.

To illustrate how to achieve the matrix inversion on the FPGA, we will take the Hermitian matrix of size 6×6 as an example. Note that both the data-map cached in dual-port RAMs and the calculation process for the different subsystems are the two important points in the descriptions of hardware implementation. In addition, some key signals passed between modules are presented in Fig. 2, 3 and 4.

B. Implementation of LDL Factorization

This part shows the hardware architecture for LDL factorization based on the outer form of Cholesky's method in Fig. 2.

From Fig. 2(a), we can see that there are five modules in the hardware implementation of LDL factorization. The dual-port RAM0 is used to receive and cache the Hermitian matrix, whereas the dual-port RAM1 and RAM2 cache the intermediate results of the diagonal elements and triangular matrix, respectively. An important aspect is the internal memory RAM data access, which achieves read/write operations over words up to 4608-b width. Thus, we use parallel loads to transfer the data to the Computation0 module in order to operate the row vectors in a single clock cycle. When the matrix size is very big, we can use more than one RAMs to cache the data. The Address0 module writes/reads the intermediate results of the matrix saved in the RAM0 and finite state machine0 (FSM0) achieves the control of the first calculation procedure. To illustrate the calculation process, Fig. 2(b) shows the first three steps of LDL factorization controlled by the FSM0 submodule, where the Stepx_en signals are from the FSM0 to start the computation and the Stepx_done signals from the Computation0 indicate the state of LDL factorization.

The following three points must be considered in the hardware implementation of LDL factorization.

1) The dual-port RAM1 just caches half the Herimitian matrix because of its symmetrical characteristic.
2) For the dual-port RAM2, we only save the nonzero elements of the upper triangular matrix not containing its diagonal elements.
3) The LUTs are selected to realize the sign bit inversion so as to economize on the precious DSP resources.

C. Implementation of Upper Triangular Matrix Inversion

In this part, we show the hardware implementation of the upper triangular matrix inversion in Fig. 3.

In Fig. 3(a), there are four modules in the hardware implementation of the upper triangular matrix inversion. The dual-port RAM3 is used to cache the matrix inversion result, whereas the Address1 module reads the data from the dual-port RAM2 and saves the results into the dual-port RAM3. The second procedure is controlled by the FSM1 module. We here also use the LUTs to achieve the sign bit inversion in the Computation1

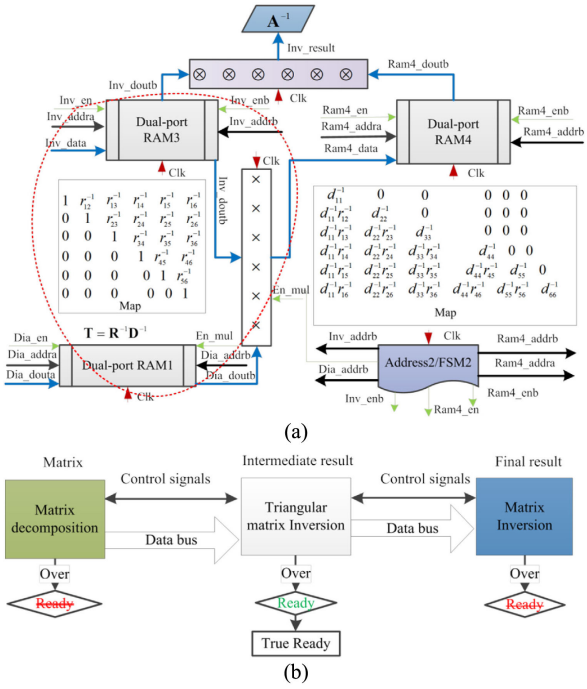

Fig. 4. Hardware architecture for the three matrices multiplication. (a) Modules of matrices multiplication. (b) Whole hardware architecture.

module. Fig. 3(b) shows the first three steps of the calculation process for the upper triangular matrix inversion, where the LDL_Stepx_en signals are from the FSM1 to start the computation and the LDL_Stepx_ done signals indicate the state of the calculation process in order that the Address1 can generate the new address to send the following data from the dual-port RAM2 to the Computation1. Here, we just compute the $0.5(N^2-3N+2)$ nonzero elements of the upper triangular matrix inversion marked by the ellipses with the red lines in Fig. 3(a). The basic circuit in Fig. 3(c) is often termed as processing element (PE), which is also used in LDL factorization. For Fig. 3(c), the current and next data are $\{-1, r_{13}, r_{12}, r_{23}, -1, r_{14}, r_{12}, r_{24}\}$ and $\{0, r_{13}, -1, r_{23}, 0, r_{14}, -1, r_{24}\}$, respectively.

D. Implementation of Matrices Multiplication

Finally, the hardware architecture of the three matrices multiplication is shown in Fig. 4(a).

From Fig. 4(a), it can be observed that there are the six modules in the last procedure. We first compute $\mathbf{T} = \mathbf{R}^{-1}\mathbf{D}^{-1}$, which is marked by the red dashed line. Here, the dual-port RAM4 is used to cache the intermediate result $\mathbf{T}$. Then, $\mathbf{A} = \mathbf{T}\mathbf{R}^{-H}$ is accomplished through the data from the dual-port RAM3 and RAM4. Then, we discuss the whole hardware architecture for our method. For Fig. 4(b), there is the tristage hardware architecture in our method, which means that if the first two parts accomplish the matrix decomposition and upper triangular matrix inversion, respectively, we can receive the new matrix at once. However, the algorithms not based on the matrix decomposition cannot receive the new matrix until their circuit

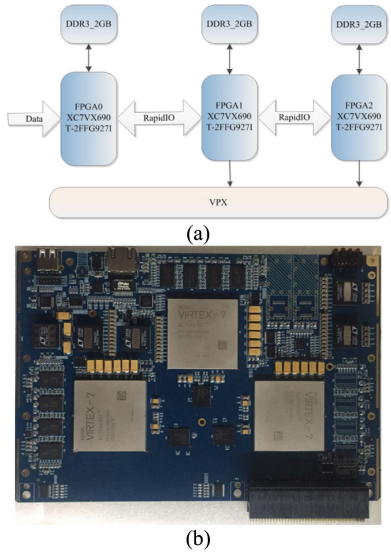

Fig. 5. Hardware platform. (a) Structure scheme. (b) Physical FPGA board.

completes the whole matrix inversion. Thus, our method has the characteristic of high throughput.

Remark: There are the six dual-port RAMs to cache the data for our method in the hardware implementation. To save the memory resources, both the symmetrical characteristic and zero padding must be considered in your program.

IV. Experimental Result

In this section, we will conduct the different experiments to illustrate the performances of the four different methods, where the numerical calculation is performed in single floating-point arithmetic. Initially, we simply introduce the hardware platform shown in Fig. 5. Fig. 5(a) describes the structure of the FPGA board, whereas the physical map is shown in Fig. 5(b).

From Fig. 5, it can be seen that the FPGA board is based on the open VPX architecture. The board is mainly composed of three FPGAs (Xilinx XC7VX690T) equipped with three groups of 2-GB DDR3, respectively. Meanwhile, FPGAs are linked with the different high-speed serial rapid input/output (SRIOs) in order to exchange the data efficiently.

A. Calculation in MATLAB

In the first experiment, we run the four matrix inversion methods (Cholesky's method [11], the SPMI [16], our method, and QR factorization) in MATLAB on Intel Core i7 4600U processor by randomly generating 100 000 nonsingular Hermitian matrices. All the computing is obtained by coding with the loop control statements in MATLAB according their algorithm structures (without multicore optimizations) and the results of their average time are shown in Table II.

For Table II, it can be seen that our method has the least computing time. In fact, the elapsed time of CPUs just indicates the operations number of different methods and the results are not simply suit for the FPGAs. It should be pointed out that the

TABLE II
COMPUTING TIME OF THREE METHOD IN MATLAB (UNIT: MS)

Matrix Dimension	Cholesky	SPMI	QR	Our Method
6×6	0.551	0.219	0.529	0.217
12×12	2.049	0.541	0.805	0.539
24×24	7.832	1.347	1.583	1.342
36×36	17.493	2.557	3.267	2.551
72×72	34.991	5.102	7.369	5.093

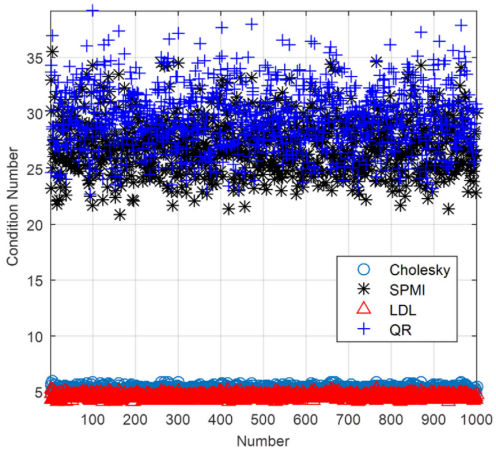

Fig. 6. Condition number of the random data processed by the four methods. The circles, stars, triangles, and plus signs represent the methods based on Cholesky's method, SPMI, LDL, and QR factorization, respectively.

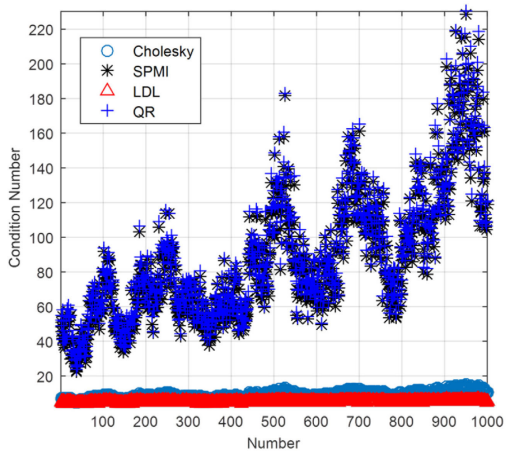

Fig. 7. Condition number of the sea clutter processed by the four methods. The circles, stars, triangles, and plus signs represent the methods based on Cholesky's method, SPMI, LDL, and QR factorization, respectively.

TABLE III
UTILIZATION OF THE FOUR METHODS

Resource	Cholesky	SPMI	QR	Our Method
LUT	261,432	263,072	230,161	250,681
LUTRAM	6,804	6,276	6,132	5,924
BRAM	891	636	910	834
DSP48	2421	2,283	2,174	1,839
Device Latency(*us*)	4703.95	3074.82	3947.05	2359.39
Throughput (*Inv/s*)	418.7	315.2	253.4	589.1
f_{max} (MHz)	200	200	200	250

FPGA uses one NOR gate not multiplier to achieve the reverse of the sign bit in one single clock cycle. If we use one multiplier to realize the reverse operation, it needs more than one clock cycles. Hence, our method and the SPMI have the approximate results on the computer platform, whereas it has the better performance than the SPMI on the FPGA platform.

B. Condition Number

It is known that the condition number [23] of a matrix measures the sensitivity of the solution of a system of linear equations to errors in the data. It gives an indication of the accuracy of the results from matrix inversion. Values of the condition number near 1 indicate a well-conditioned matrix. If the matrix has the smaller value of the condition number, it will be of advantage to calculate its inversion. Thus, we test the condition number of 1000 random square matrices of size 32 × 32 generated by the RANDN function in MATLAB, which are processed by the four methods listed in Table II. The experimental results are shown in Fig. 6.

From Fig. 6, we can see that the inversion method based on LDL factorization has the smallest condition number among the four methods. In other words, our method shares the biggest advantage in the matrix inversion.

In fact, we are not always confronted with the matrix of normally distributed random numbers. To further illustrate the robustness of the four methods, we use the IPIX sea clutter [24] as the dataset to compute the condition number. It should be noted that we just select the sea clutter as the test data, which has no meaning in any method. The experimental results are shown in Fig. 7.

From Fig. 7, we can get the basic same conclusion with that from Fig. 6. Both the simulated data and raw data show that the matrix inversion method based on LDL factorization has the smallest condition number. All in all, our proposed method indeed is a robust algorithm to realize the matrix inversion. From the experimental results, we can see that there is a big difference between the random numbers and raw data, so we suggest that the matrix inversion should be implemented by the method based on LDL factorization in engineering.

C. Hardware Implementation

In this part, we still select the four methods listed in Table II and program them in the Verilog language on the FPGA board shown in Fig. 5(b). In the following experiments, the basic mathematical calculations in the FPGA are completed by the implementation of the Xilinx floating-point operator v7.1 [25], which is in full compliance with IEEE-754 Standard. We use the Xilinx Vivado toolset (2016.02) to accomplish the inverse of complex matrices whose size is 72 × 72. The resource utilizations of the four methods are listed in Table III.

For Table III, we analyze the results from the following four aspects: resource utilization, maximum work frequency, device latency, and throughput.

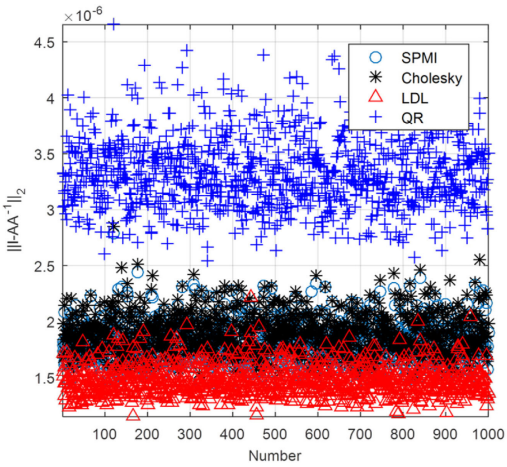

Fig. 8. Matrix two-norm errors of the simulated data. The circles, stars, triangles, and plus signs represent the methods based on Cholesky's method, SPMI, LDL, and QR factorization, respectively.

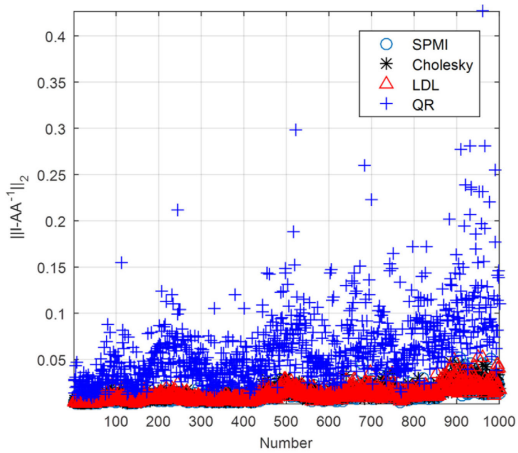

Fig. 9 Matrix two-norm errors of the sea clutter. The circles, stars, triangles, and plus signs represent the methods based on Cholesky's method, SPMI, LDL, and QR factorization, respectively.

First, we discuss the resource utilization of the four methods, the matrix inversion based on Cholesky's method needs the most RAMs (LUTRAM and block RAM), but the SPMI needs the least RAMs. The reason is that there is no matrix decomposition in the SPMI algorithm. For DSP48, the elapsed number of multipliers can be simply assumed to be corresponding to that of the multiplication numbers needed by the different methods.

From Table III, we can see that our method has the maximum work frequency. The reason is that our method has the lowest computation complexity, which can be seen from the device latency of the four methods.

Finally, we discuss the device latency and throughput, respectively. The device latency is the computation time for one set data, whereas the throughput denotes the number of matrices calculated by the programmed circuit per second. From Table III, it can be observed that our method has the smallest device latency and the highest throughput rate. Thus, our method has the best performance.

In the following, we use the matrix two-norm error to illustrate the performance of the four methods. Here, the matrix two-norm error is defined by

$$c = \left\| \mathbf{I} - \mathbf{A}\mathbf{A}^{-1} \right\|_2 \tag{23}$$

where $\| \|_2$ denotes the matrix two-norm and $\mathbf{I}$ is one unit matrix. It can be easily seen that when c is smaller, $\mathbf{A}^{-1}$ is more closely to the true solution. Now, 1000 random Hermitian matrices of size 72×72 generated by MATLAB are processed by FPGA and the calculation results are exported via the Ethernet interface. Then, the matrix two-norm errors of the test data are analyzed by MATLAB and shown in Fig. 8.

From Fig. 8, it can be observed that the four methods have the small matrix two-norm errors overall. The method based on the QR factorization has the highest amplitude values of the matrix two-norm errors, whereas our method has the smallest matrix two-norm errors. Next, the IPIX sea clutter is also used to test the four methods on the FPGA board and the experimental results are shown in Fig. 9.

Form Fig. 9, we can get the basic same conclusion with that from Fig. 8. In this experiment, we use the different dataset to test the four methods. From Figs. 8 and 9, it can be seen that our method has the smallest matrix two-norm errors. Note that Cholesky's method has the similar performance to our method. The reason is that the LDL factorization is actually one variant form of Cholesky's method. However, we can still say our method is better than Cholesky's method owing to the least resource utilization and lowest condition number. On the other hand, the values of the raw data are higher than that of the simulated data, which suggests that the method based on LDL factorization should be widely adopted in engineering.

V. Application

To evaluate our method, we apply the proposed method to the adaptive digital beam forming (ADBF) algorithm [26] for the array antenna control problem in one phased array radar. At first, we simply revisit the matrix inversion used in the different projects. For the MPC, it is commonly formulated as a quadratic programming problem and the matrix inversion is often one solver. For the power electronic converter, its real-time simulator is usually solved by the linear and classical nodal equation. Today, the array antennas have been widely used in many diverse fields of science and engineering. Some examples are astronomy, medical diagnosis, radar and seismology, etc. Thus, ADBF is one important topic for the array antenna and solved by a constrained optimization problem mathematically. In engineering, the linear constrained minimum variance (LCMV) algorithm is usually selected as a solver for the array antenna.

In this experiment, the radar is equipped with one 1-D phased-scanned array antenna in the azimuth and the parameters of the radar and experiment are listed in Table IV. The experiment is conducted in one microwave chamber shown in Fig. 11(a). Meanwhile, the flowchart of the LCMV algorithm is simply described in Fig. 10(b).

TABLE IV
PARAMETERS OF THE RADAR AND EXPERIMENT

Name	Value	Name	Value
Channel number	72	Azimuth angle	[-45°, 45°]
Snapshot	512	Scanning angle	10°
Number of jamming	2	Matrix size	72×72
Jamming angle	[-20°, -15°]	Data latency	≤ 2ms
JNR	[40dB,65dB]	Device latency	≤ 3.5ms

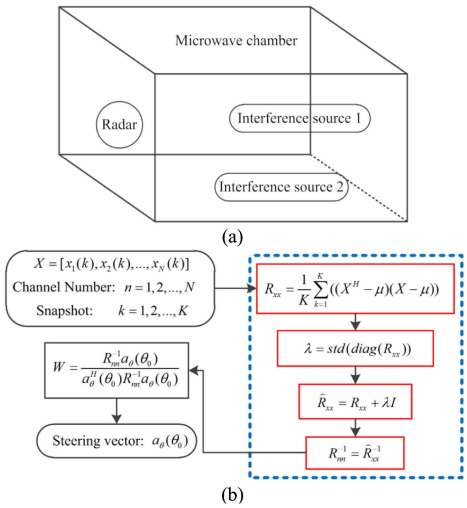

Fig. 10. Experimental scenario and flowchart of the ADBF algorithm. (a) Experimental scenario. (b) Flowchart of the LCMV algorithm.

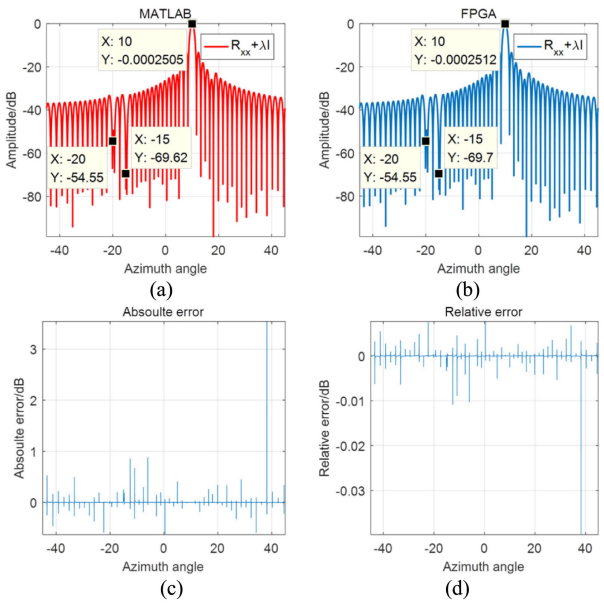

Fig. 11. Antenna pattern and the calculation errors. (a) MATLAB results. (b) FPGA results. (c) Absolute errors. (d) Relative errors.

From Fig. 10(a), we can see that two interference sources are placed at the predetermined location. The FPGA-based board acquires the returned signals through the optical fibers of the digital receiver. Here, the digital receiver board equipped in the antenna performs the analog-to-digital (AD) sampling, digital down conversion, and data packaging and transmission. Meanwhile, the LCMV algorithm is programmed and configured on one Xilinx XC7VX690T FPGA accomplishing the four parts (the calculation of the covariance matrix, standard deviation, diagonal loading, and matrix inversion) of the algorithm marked by the blue-dashed line shown in Fig. 10(b). We use the Verilog hardware language to program the LCMV algorithm (in single floating-point arithmetic) on the FPGA and verify the Verilog program with the Modelsim SE-64 10.2c. After the synthesis, the logical resource utilization listed in Table V can be obtained by the report from the Xilinx Vivado toolset.

TABLE V
UTILIZATION-POST IMPLEMENTATION

Resource	Utilization (%)	Name	Value
LUT	301,183(69.53)	Device latency(us)	3099.91
LUTRAM	15,124(8.68)	Data latency (us)	1668.98
BRAM	1,034(70.34)	f_{max} (MHz)	200
DSP48	2,439(78.86)	Word length	32-bit

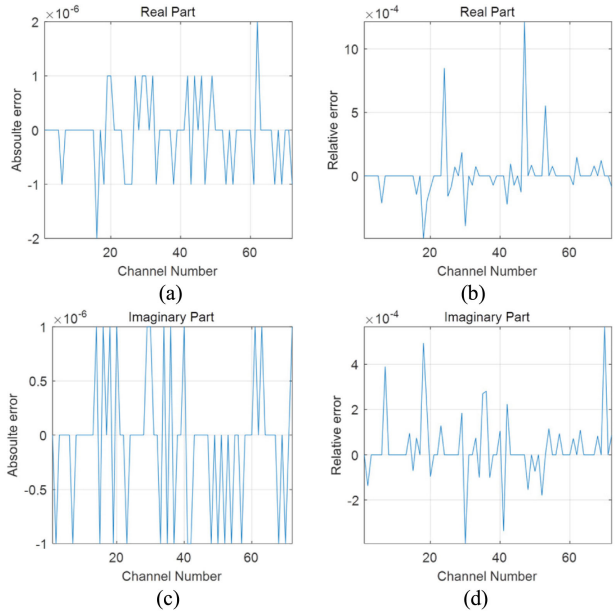

Fig. 12. Errors of the real and imaginary parts. (a) Relative errors of the real part. (b) Absolute errors of the real part. (c) Relative errors of the imaginary part. (d) Absolute errors of the imaginary part.

Before analyzing Table V, it is well known that real time is one important characteristic for the radar signal processing (RSP). From Table IV, it can be seen that the required latencies of the data and device are not bigger than 2 and 3.5 ms, respectively, for the RSP subsystem. Now, we can analyze the experimental results listed in Table V. First, the implementation of the LCMV algorithm is achieved on one FPGA in single floating-point arithmetic. The latencies of the data and device are about 1.669 and 3.100 ms, respectively, which meets the RSP system requirements.

Next, the antenna pattern is used to illustrate the performance of the LCMV algorithm implemented by our proposed method. For the FPGA, both the input data and output results are cached

in the DDR3 and exported to the computer via the Ethernet interface.

From Fig. 11(a) and (b), we can see that the antenna pattern is computed by MATLAB and FPGA, respectively. The jamming angle is set at [–20° –15°] and it is easily seen that the interference suppression results are consistent with the theory. Meanwhile, both the absolute and relative errors of the antenna pattern are shown in Fig. 11(c) and (d), respectively. For the calculation errors, it is tolerable in engineering.

Finally, we analyze the calculation errors in its real and imaginary part, respectively. Both the absolute and relative errors of the real and imaginary parts are shown in Fig. 12.

From Fig. 12, it can be observed that the amplitude values of the relative errors are lower than that of absolute errors. Specially, the absolute errors denote the difference between the FPGA and MATLAB, whose amplitude values are at 10^{-4}. Finally, we can say that our proposed method has solved the control problem for the array antenna of one phased array radar successfully.

VI. Conclusion

In this article, the characteristics of both the triangular matrix and its inversion were stolen to accomplish the matrix inversion. Real time and numerical stability were the two focuses in this article. In fact, we designed the high-performance signal processing board based three FPGAs to verify our method, which solved the antenna control problem for one phased array radar successfully. For scalability, the idea was similar to other matrix inversion methods. Actually, we have achieved the square matrix inversion with the different sizes, such as 54×54, 64×64, and 128×128 on one FPGA chip. In theory, our method can be divided in the hardware implementation on different FPGAs, which means that the board can achieve the bigger size of square or common matrix inversion (include the Moore–Penrose matrix inversion). Due to the space, we do not discuss the matrix inversion on the different FPGAs in detail.

References

[1] T. J. Besselmann, S. Van de Moortel, S. Almér, P. Jörg, and H. J. Ferreau, "Model predictive control in the multi-megawatt range," *IEEE Trans. Ind. Electron.*, vol. 63, no. 7, pp. 4641–4648, Jul. 2016.

[2] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Eng. Pract.*, vol. 11, no. 7, pp. 733–764, 2003.

[3] C. X. Wang *et al.*, "Cellular architecture and key technologies for 5G wireless communication networks," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 122–130, Feb. 2014.

[4] F. Rusek *et al.*, "Scaling up MIMO: Opportunities and challenges with very large arrays," *IEEE Signal Process. Mag.*, vol. 30, no. 1, pp. 40–60, Jan. 2013.

[5] M. O. Faruque *et al.*, "Real-time simulation technologies for power systems design, testing, and analysis," *IEEE Power Energy Technol. Syst. J.*, vol. 2, no. 2, pp. 63–73, Jun. 2015.

[6] T. Ould-Bachir, H. Blanchette, and K. Al-Haddad, "A network tearing technique for FPGA-based real-time simulation of power converters," *IEEE Trans. Ind. Electron.*, vol. 62, no. 6, pp. 3409–3418, Jun. 2015.

[7] J. Ward, "Space-time adaptive processing for airborne radar," Lincoln Lab., Massachusetts Inst. Technol., Lexington, MA, USA, Tech. Rep. 1015, 1994.

[8] S. D. Somasundaram, "Linearly constrained robust Capon beam forming," *IEEE Trans. Signal Process.*, vol. 60, no. 11, pp. 5845–5856, Nov. 2012.

[9] A. De Maio and D. Orlando, "Adaptive radar detection of a subspace signal embedded in subspace structured plus Gaussian interference via invariance," *IEEE Trans. Signal Process.*, vol. 64, no. 8, pp. 2156–2167, Apr. 2016.

[10] D. S. Watkins, *Fundamentals of Matrix Computations*, 2nd ed. Hoboken, NJ, USA: Wiley, 2002.

[11] A. Krishnamoorthy and D. Menon, "Matrix inversion using Cholesky decomposition," in *Proc. Signal Process., Algorithms, Architectures, Arrangements, Appl.*, 2013, pp. 70–72.

[12] A. Burian, P. Salmela, and J. Takala, "Complex fixed-point matrix inversion using transport triggered architecture," in *Proc. IEEE Int. Conf. Appl.-Specific Syst. Architecture Processors*, 2005, pp. 107–112.

[13] M. He, C. Chen, and X. Zhang, "FPGA implementation of a recursive rank one updating matrix inversion algorithm for constrained MPC," in *Proc. 6th World Congr. Intell. Control Automat.*, 2006, vol. 1, pp. 733–737.

[14] S.D. Munoz and J. Hormigo, "High-throughput FPGA implementation of QR decomposition," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 9, pp. 861–865, Sep. 2015.

[15] C. Zhang *et al.*, "On the low-complexity, hardware-friendly tridiagonal matrix inversion for correlated massive MIMO systems," *IEEE Trans. Veh. Technol.*, vol. 68, no. 7, pp. 6272–6285, Jul. 2019.

[16] Y. W. Xu, Y. Xi, J. Lan, and T. F. Jiang, "An improved predictive controller on the FPGA by hardware matrix inversion," *IEEE Trans. Ind. Electron.*, vol. 65, no. 9, pp. 7395–7405, Sep. 2018.

[17] A. Hadizadeh, M. Hashemi, M. Labbaf, and M. Parniani, "A matrix-inversion technique for FPGA-based real-time EMT simulation of power converters," *IEEE Trans. Ind. Electron.*, vol. 66, no. 2, pp. 1224–1234, Feb. 2019.

[18] J. Arias-García, C. H. Llanos, M. Ayala-Rincón, and R. P. Jacob, "FPGA implementation of large-scale matrix inversion using single, double and custom," in *Proc. VIII Southern Conf. Programmable Log.*, 2012, pp. 1–6.

[19] J. P. David, "Low latency and division free Gauss-Jordan solver in floating point arithmetic," *J. Parallel Distrib. Comput.*, vol. 106, pp. 185–193, 2017.

[20] A. Burian, J. Takala, and M. Ylinen, "A fixed-point implementation of matrix inversion using Cholesky decomposition," in *Proc. 46th Midwest Symp. Circuits Syst.*, Dec. 2003, vol. 3, pp. 1431–1434.

[21] C. Ingemarsson and O. Gustafsson, "Finite wordlength properties of matrix inversion algorithms in fixed-point and logarithmic number systems," in *Proc. 20th Eur. Conf. Circuit Theory Des.*, 2011, pp. 673–676.

[22] N.J. Higham, *Accuracy and Stability of Numerical Algorithms*. Philadelphia, PA, USA: SIAM, 1996.

[23] L. N. Trefethen and I. David Bau, *Numerical Linear Algebra*. Philadelphia, PA, USA: SIAM, 1997.

[24] The McMaster IPIX Radar Sea Clutter Database, 1998. [Online]. Available: http://soma.ece.mcmaster.ca/ipix/

[25] Xilinx, Floating-point operator v7.1, 2017. [Online]. Available: https://www.xilinx.com

[26] B. Windrow and S. D. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1985.

Xiao-Wei Zhang received the B.S. degree in electronic and information engineering from the First Aeronautical College, People's Liberation Army, Xinyang, China, in 2008, and the Ph.D. degree in signal and information processing from Xidian University, Xi'an, China, in 2014.

In 2014, he joined the Shaanxi Huang-He Group Company, Ltd., Xi'an, China. Since 2016, he has been an Associate Professor with the School of Information Engineering, Xijing University, Xi'an, and also with the National Laboratory of Radar Signal Processing, Xidian University. His current research interests include wideband radar signal processing, compressive sensing, real-time control system, and target tracking.

Lei Zuo (Member, IEEE) received the B.S. in navigation, guidance, and control technology and Ph.D. degrees in signal and information processing from Xidian University, Xi'an, China, in 2008 and 2014, respectively.

In 2014, he joined the Department of Electronic Engineering, Xidian University, where he is currently an Associate Professor with the National Laboratory of Radar Signal Processing. His current research interests include the time–frequency analysis and synthesis, radar sea clutter suppression, and target detection.

Ming Li (Member, IEEE) received the B.S., M.S. in electronic and information engineering and Ph.D. degrees in signal and information processing from Xidian University, Xi'an, China, in 1987, 1990, and 2007, respectively.

In 1987, he joined the Department of Electronic Engineering, Xidian University, where he is currently a Professor with the National Laboratory of Radar Signal Processing. His research interests include adaptive signal processing, detection theory, high-performance hardware integration, and ultra-wideband radar system.

Jian-Xin Guo received the B.S. degree in electronic and information engineering from the Telecommunications Engineering Institute of the Air Force, People's Liberation Army, Xi'an, China, in 1997, the M.S. degree in information and communications engineering from the Air Force Engineering University, People's Liberation Army, Xi'an, China, in 2000, and the Ph.D. degree in information and communications engineering from the PLA Information Engineering University, Zhengzhou, China, in 2004.

Since 2013, he has been a Professor with the School of Information Engineering, Xijing University, Xi'an. His current research interests include cognitive radio technologies, MIMO-OFDM wireless communications, and their implementation in GNU radio.