*remote sensing*

*Article*

# On-Board Georeferencing Using FPGA-Based Optimized Second-Order Polynomial Equation

**Dequan Liu [1,3], Guoqing Zhou [1,2,3,4,*], Jingjin Huang [2,3], Rongting Zhang [2,3], Lei Shu [2,3], Xiang Zhou [1,4] and Chun Sheng Xin [5]**

1 School of Microelectronics, Tianjin University, Tianjin 300072, China; 1017232001@tju.edu.cn (D.L.); zqx0711@tju.edu.cn (X.Z.)
2 School of Precision Instrument & Opto-Electronics Engineering, Tianjin University, Tianjin 300072, China; jingjin_huang@tju.edu.cn (J.H.); zrt65@tju.edu.cn (R.Z.); shulei@tju.edu.cn (L.S.)
3 The Center for Remote Sensing, Tianjin University, Tianjin 300072, China
4 GuangXi Key Laboratory for Spatial Information and Geomatics, Guilin University of Technology, Guilin 541004, China
5 Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, VA 23529, USA; cxin@odu.edu
* Correspondence: gzhou@glut.edu.cn; Tel.: +86-773-589-6073

Received: 27 November 2018; Accepted: 3 January 2019; Published: 10 January 2019

**Abstract:** For real-time monitoring of natural disasters, such as fire, volcano, flood, landslide, and coastal inundation, highly-accurate georeferenced remotely sensed imagery is needed. Georeferenced imagery can be fused with geographic spatial data sets to provide geographic coordinates and positing for regions of interest. This paper proposes an on-board georeferencing method for remotely sensed imagery, which contains five modules: input data, coordinate transformation, bilinear interpolation, and output data. The experimental results demonstrate multiple benefits of the proposed method: (1) the computation speed using the proposed algorithm is 8 times faster than that using PC computer; (2) the resources of the field programmable gate array (FPGA) can meet the requirements of design. In the coordinate transformation scheme, 250,656 LUTs, 499,268 registers, and 388 DSP48s are used. Furthermore, 27,218 LUTs, 45,823 registers, 456 RAM/FIFO, and 267 DSP48s are used in the bilinear interpolation module; (3) the values of root mean square errors (*RMSE*s) are less than one pixel, and the other statistics, such as maximum error, minimum error, and mean error are less than one pixel; (4) the gray values of the georeferenced image when implemented using FPGA have the same accuracy as those implemented using MATLAB and Visual studio (C++), and have a very close accuracy implemented using ENVI software; and (5) the on-chip power consumption is 0.659 W. Therefore, it can be concluded that the proposed georeferencing method implemented using FPGA with second-order polynomial model and bilinear interpolation algorithm can achieve real-time geographic referencing for remotely sensed imagery.

**Keywords:** georeferencing; second-order polynomial equation; bilinear interpolation; the field programmable gate array (FPGA)

## 1. Introduction

With the advancement of technological, remote sensing (RS) images are becoming more widely used in natural disasters monitoring and positioning [1–3]. They are required to rapidly produce highly accurate georeferenced image. However, the processing speed of the traditional methods for indirect georeferencing of RS images cannot to meet the timeliness requirements [4–9]. Therefore, it would be highly desirable to develop a rapid, inexpensive technique to implement an on-board georeferencing.

This paper proposes a georeferencing method based onfield programmable gate arrays (FPGAs) with the optimized second-order polynomial equation and bilinear interpolation scheme.

Georeferencing is a key step in geometric correction which aims to establish the relationship between image coordinates and ground coordinates through various function, such as collinearity equation models (CEM), polynomial function models (PFM), rational function models (RFM), and direct linear transformation models (DLT) [10–15]. However, these traditional algorithms were developed for serial instruction systems based on personal computers (PC). As such, these systems hardly meet response time demands of time-critical disasters [5].

To solve the limitation of the speed of RS images processing, many researchers proposed effective alternative processing methods. High-performance computing (HPC) was widely used in RS data processing system. Cluster computing has already offered access to greatly increased computational power at a low cost in a few hyperspectral imaging applications [16–18]. Although the algorithms of RS data processing generally map quite nicely to multi-processor systems composed of clusters or networks of CPUs, these systems are usually expensive and difficult to adapt to the on-board RS data processing scenarios. For these reasons, the specialized integrated hardware devices of low weight and low power consumption are essential to reduce mission payload and obtain analysis results in real-time. FPGAs and graphics processing units (GPUs) exhibit good potential to allow for on-board real-time analysis of RS data. Fang et al. [19] presented near real-time approach for CPU/GPU based preprocessing of ZY-3 satellite images. Van et al. [20] proposed a new method to generate undistorted images by implementing the required distortion correction algorithm on a commercial GPU. Thomas et al. [21] described the georeferencing of an airborne hyperspectral imaging system based on pushbroom scanning. Reguera et al. [22] presented a method for real-time geo-correction of images from airborne pushbroom sensors using the hardware acceleration and parallel computing characteristics of modern GPU. López-Fandiño et al. [23] proposed a parallel function with GPU to accelerate the extreme learning machine (ELM) algorithm which is performed on RS data for land cover applications and achieved competitive accuracy results. Lu et al. [24] mapped the fusion method of RS images to GPU. The result shown that the image fusion speed based on GPU was much quicker than that based on CPU when the image size was getting bigger. Now, the use of GPUs to accelerate RS processing has resulted in further related research achievements. However, most parallel processing methods based on GPU multitasking are not alone capable of surmounting the shortcomings of serial instrument methods [25,26]. Additionally, FPGAs exhibit lower power dissipation figures than GPUs [27]. FPGAs have been consolidated as the standard choice for on-board RS data processing due to their smaller size, weight, and power consumption when compared to other HPC systems [27–32]. Zhou et al. [28] presented the concept of the "on-board processing system". Huang et al. [29] proposed an FPGA architecture that consisted of corner detection, corner matching, outlier rejection, and sub-pixel precision localization. Pakartipangi et al. [30] analyzed the camera array on-board data handling using an FPGA for nano-satellite application. Huang et al. [31] proposed a new FPGA architecture that considered the reuse of sub-image data. Yu et al. [32] presented a new on-board image compression system architecture for future disaster monitoring by low-earth-orbit (LEO) satellites. Zhou et al. [5] proposed an on-board ortho-rectification for remote sensing images based on an FPGA. Qi et al. [7] presented an FPGA and digital singnal processor (DSP) co-processing system for an optical RS images preprocessing algorithm. Long et al. [33] focused on an automatic matching technique for the specific task of georeferencing RS images and presented a technical frame to match large RS images efficiently using the prior geometric information of the images. Williams et al. [34] discussed the design and implementation of a real-time cloud detection system for on-board RS platform. González et al. [35] presented an N-finder (N-FINDR) algorithm implementation using FPGA for hyperspectral image analysis.

This paper presents an on-board georeferencing scheme using FPGA for remote sensing images. By decomposing the georeferencing algorithm, the proposed georeferencing platform integrates three modules: a data memory, coordinate transformation (including the transformation from geodetic coordinates to the raw image coordinates and the raw image coordinates to the scanning coordinates), and bilinear interpolation. The contributions are summarized as follows.

(1) An optimized second-order polynomial equation and bilinear interpolation are proposed for on-board georeferencing for remotely sensing imagery. (2) An architecture for floating-point block Lower-Upper (LU) decomposition is designed to solve the inverse for large-sized matrices. (3) To reduce resource consumption, some strategies are adopted in FPGA implementation, i.e., 32-bit integer and floating-point mixed operation and serial-parallel data communication.

The remainder of the paper is organized as follows. Section 2 reviews the traditional second-order polynomial georeferencing algorithm, bilinear interpolation method and optimizes the algorithms based on the FPGA; Section 3 gives the details of the implementation using FPGA for the georeferencing scheme; Section 4 describes and validates the experimental results. The conclusion is presented in Section 5.

## 2. Optimization for Georeferencing Scheme

The georeferencing scheme includes the selection of a suitable mathematical distortion model, coordinate transformation, and resampling (interpolation) [36–38]. The georeferencing scheme can be classified into direct and indirect methods. The direct method applies the raw image coordinates to compute the georeferenced coordinates, and the indirect method applies the georeferenced image coordinates to compute the raw image coordinates. In this paper, the indirect method is used to establish the relationship of coordinate by the second-order polynomial equations.

### *2.1. A Brief Review of Georeferencing*

#### 2.1.1. Traditional Second-Order Polynomial Equations

The second-order polynomial equations are expressed as follows [39–42]:

$$x = a_0 + a_1 X + a_2 Y + a_3 X^2 + a_4 XY + a_5 Y^2, \tag{1}$$

$$y = b_0 + b_1 X + b_2 Y + b_3 X^2 + b_4 XY + b_5 Y^2, \tag{2}$$

where $(x, y)$ are the coordinates of the raw image, $(X, Y)$ are the corresponding ground coordinates (longitude, latitude), and $a_i$ and $b_i$ ($i = 0, 1, \ldots, 5$) are the unknown coefficients of the second-order polynomial equation. After choosing a suitable mathematical distortion model, the unknown coefficients $a_i$ and $b_i$ can be obtained from Equations (1) and (2) with the ground control points (GCPs). GCPs are an important parameter in the geometric calibration, which affect the accuracy of subsequent correction [43]. To ensure accuracy, ten GCPs are selected in the study area. Let the coordinate pairs of ten GCPs can be represented as: $(x_1, y_1, X_1, Y_1)$, $(x_2, y_2, X_2, Y_2)$, ... , $(x_{10}, y_{10}, X_{10}, Y_{10})$. The matrix form of Equation (1) can be expressed as:

$$\begin{bmatrix} 1 & X_1 & Y_1 & X_1^2 & X_1Y_1 & Y_1^2 \\ 1 & X_2 & Y_2 & X_2^2 & X_2Y_2 & Y_2^2 \\ 1 & X_3 & Y_3 & X_3^2 & X_3Y_3 & Y_3^2 \\ 1 & X_4 & Y_4 & X_4^2 & X_4Y_4 & Y_4^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & X_{10} & Y_{10} & X_{10}^2 & X_{10}Y_{10} & Y_{10}^2 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{10} \end{bmatrix}. \tag{3}$$

According to the least square method [5], Equation (3) can be simplified as Equation (4).

$$(A^T PA)\Delta_a = A^T PL_x, \tag{4}$$

where $A$ is the matrix of the ground coordinates of GCPs, $\Delta_a$ is the coefficients matrix of the second-order polynomial Equation (1), $L_x$ is the coordinates matrix of GCPs in the raw image, and $P$ is the weight matrix. $A$ , $\Delta_a$, $L_x$ and $P$ are expressed by Equation (5) through Equation (8).

$$A = \begin{bmatrix} 1 & X_1 & Y_1 & X_1^2 & X_1Y_1 & Y_1^2 \\ 1 & X_2 & Y_2 & X_2^2 & X_2Y_2 & Y_2^2 \\ 1 & X_3 & Y_3 & X_3^2 & X_3Y_3 & Y_3^2 \\ 1 & X_4 & Y_4 & X_4^2 & X_4Y_4 & Y_4^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & X_{10} & Y_{10} & X_{10}^2 & X_{10}Y_{10} & Y_{10}^2 \end{bmatrix}, \tag{5}$$

$$\Delta_a = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \end{bmatrix}^T, \tag{6}$$

$$L_x = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix}^T, \tag{7}$$

$$P = \begin{bmatrix} P_1 & & & & & \\ & P_2 & & & & \\ & & P_3 & & & \\ & & & P_4 & & \\ & & & & \vdots & \\ & & & & & P_{10} \end{bmatrix} \tag{8}$$

Typically, the weight of each GCPs is the same. So, $P = I$. Equation (4) can be descripted as:

$$\Delta_a = (A^TA)^{-1}(A^TL_x). \tag{9}$$

In the same way, the coefficients of $b$ can be solved by the formula $\Delta_b = (A^TA)^{-1}(A^TL_y)$.

### 2.1.2. Coordinate Transformation

After establishing the second-order polynomial equation and solving the coefficients of Equations (1) and (2), the raw image can be georeferenced pixel-to-pixel. The steps are as follows [44]:

1. The Size of the Georeferenced Image

To properly obtain the georeferenced image, the storage area of georeferenced image must be computed in advance, as shown in Figure 1 [7]. *abcd* is the raw image, with corners $a$ , $b$, $c$, and $d$ in the $o - xy$ image coordinate system in the Figure 1a. Figure 1b shows the correct range of output image. $a'b'c'd'$ is the georeferenced image with corners $a'$, $b'$, $c'$ and $d'$ in the $O - XY$ ground coordinate system, and $ABCD$ is the storage area for georeferenced image. Obviously, if the storage range is not correctly defined, the georeferenced image is improper with a large blank area, as shown in Figure 1c. Therefore, the right boundary should include the entire georeferenced image with minimum exterior blank rectangle as possible.

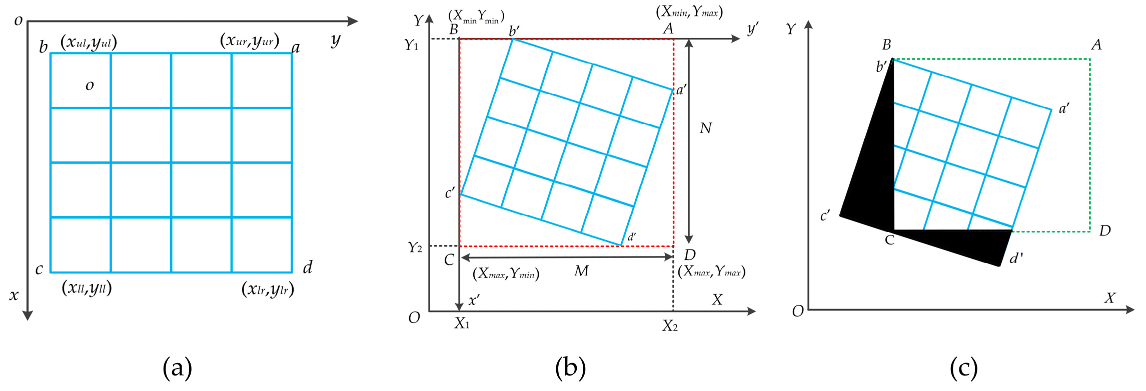


**Figure 1.** The georeferenced image size. (**a**) Input image. (**b**) Output image. (**c**) Incorrect output image. *ur*:, upper right; *ul*:, upper left; *ll*:, lower left; *lr*:, lower right. *abcd* is the input image. *ABCD* is the storage area for the georeferenced image. $a'b'c'd'$ is the georeferenced image.

2. Coordinates of the Four Corners

$(x_{ul}, y_{ul})$, $(x_{ur}, y_{ur})$, $(x_{lr}, y_{lr})$, and $(x_{ll}, y_{ll})$ are the coordinates of the four corner in the raw image (input image), $(X_{ul}, Y_{ul})$, $(X_{ur}, Y_{ur})$, $(X_{lr}, Y_{lr})$, and $(X_{ll}, Y_{ll})$ are the corresponding ground coordinates of the georeferenced image. The maximum and minimum ground coordinates of the storage area for the georeferenced image ($X_{\min}$, $X_{\max}$, $Y_{\min}$, and $Y_{\max}$) are calculated from the $(X_{ul}, Y_{ul})$, $(X_{ur}, Y_{ur})$, $(X_{lr}, Y_{lr})$, and $(X_{ll}, Y_{ll})$.

$$\begin{aligned} X_{\min} &= \min(X_{ur}, X_{ul}, X_{lr}, X_{ll}) \\ X_{\max} &= \max(X_{ur}, X_{ul}, X_{lr}, X_{ll}) \\ Y_{\min} &= \min(Y_{ur}, Y_{ul}, Y_{lr}, Y_{ll}) \\ Y_{\max} &= \max(Y_{ur}, Y_{ul}, Y_{lr}, Y_{ll}). \end{aligned} \tag{10}$$

3. Row ($M$) and Column ($N$) of the Georeferenced Image.

$M$ and $M$ can be solved by Equation (11).

$$\begin{aligned} M &= (X_{\max} - X_{\min})/X_{GSD} + 1 \\ N &= (Y_{\max} - Y_{\min})/Y_{GSD} + 1, \end{aligned} \tag{11}$$

where $X_{GSD}$ and $Y_{GSD}$ are the ground-sampling distances ($GSD$) of the output image. Hence, the location of pixel can be described by the row and column in the $B - x'y'$ coordinate system ($x' = 1, 2, 3, \ldots, M; y' = 1, 2, 3, \ldots, N$).

4. Coordinate Transformation.

The georeferenced model only expresses the relationship between the ground coordinates $(X, Y)$ and the $(x, y)$ coordinates of the raw image. To further express the relationship between the raw image coordinates and scanning coordinates of the georeferenced image, it is necessary to convert the ground coordinate into the row and column of georeferenced image, as shown in Equation (12).

$$\begin{aligned} X_g &= X_{\min} + X_{GSD}(x' - 1) \\ Y_g &= Y_{\max} - Y_{GSD}(y' - 1), \end{aligned} \tag{12}$$

where $x'$ and $y'$ are the row and column of the georeferenced image, respectively, and $(X_g, Y_g)$ are the corresponding ground coordinate.

### 2.1.3. Resample

As to be expected, the defined grid center from the georeferenced image will not usually project to the center location of pixel in the raw image after coordinate transformation. To solve this problem, nearest-neighbor interpolation, bilinear interpolation, and cubic interpolation [45] are commonly used. To balance the accuracy and complexity of the preprocessing function, bilinear interpolation is chosen in this paper. The algorithm obtains the pixel value by taking a weighted sum of the pixel values of the four nearest neighbors surrounding the calculated location [46], as shown in Figure 2 and Equation (13).

$$I(x,y) = I(i,j)(1-u)(1-v) + I(i+1,j)u(1-v) + I(i,j+1)(1-u)v + I(i+1,j+1)uv, \tag{13}$$

where, $(x,y)$ are the coordinates of the georeferenced image, $I(x,y)$ is the gray value of a pixel with $(x,y)$ coordinates in the georeferenced image; $i$ and $j$ are the integer part of the corresponding coordinate in the raw image, respectively; $u$ and $v$ are the fractional parts of the corresponding coordinate in the raw image, respectively. $I(i,j)$ is the gray value of $(i,j)$ coordinates in the raw image.

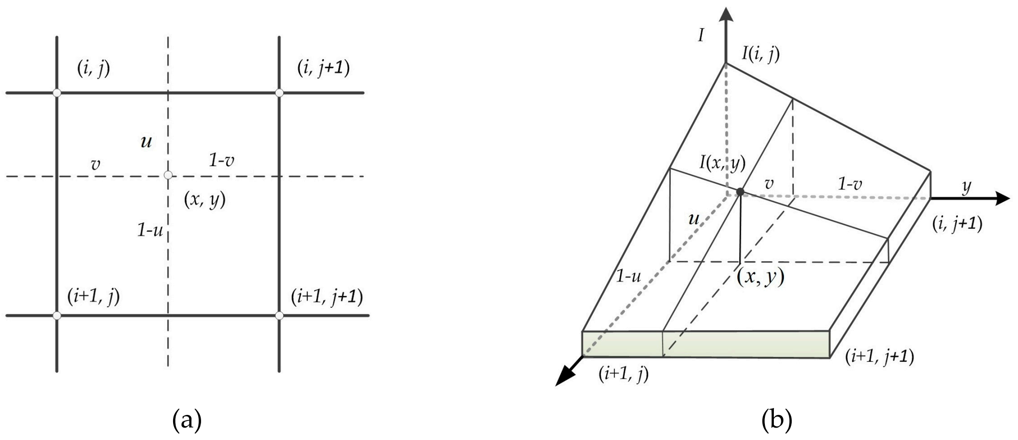


**Figure 2.** The architecture for the bilinear interpolation. (**a**) Distributions of the four nearest neighbor pixels and (**b**) 3D spatial distribution of the bilinear interpolation.

## *2.2. Optimized Georeferencing Scheme*

### 2.2.1. Optimized Second-Order Polynomial Model

To parallel implement the Equations (1) and (2), the second-order polynomial equations are optimized to Equations (14) and (15).

$$\begin{aligned} x &= a_0 + a_1X + a_2Y + a_3X^2 + a_4XY + a_5Y^2 = a_0 + (a_1 + a_3X)X + (a_2 + a_4X + a_5Y)Y \\ &= a_0 + p_1(X) + p_2(X,Y) \end{aligned} \tag{14}$$

$$\begin{aligned} y &= b_0 + b_1X + b_2Y + b_3X^2 + b_4XY + b_5Y^2 = b_0 + (b_1 + b_3X)X + (b_2 + b_4X + b_5Y)Y \\ &= b_0 + q_1(X) + q_2(X,Y) \end{aligned} \tag{15}$$

where, $p_1(X) = (a_1 + a_3X)X$, $p_2(X,Y) = (a_2 + a_4X + a_5Y)Y$, $q_1(X) = (b_1 + b_3X)X$, and $q_2(X,Y) = (b_2 + b_4X + b_5Y)Y$.

In Equation (1), eight multipliers and five adders are required to complete a transformation. However, in Equation (14), five multipliers and five adders are required to complete a transformation. Comparatively, six multipliers are reduced in all. The multipliers and adders required to complete the coordinate transformation are in Table 1.

**Table 1.** The number of multiplier and adder comparison with Equations (1), (2), (14), and (15).

| Coordinate | Traditional 2nd Polynomial | | Optimized 2nd Polynomial | | Reduce | |
|---|---|---|---|---|---|---|
| | Multiplier | Adder | Multiplier | Adder | Multiplier | Adder |
| x | 8 | 5 | 5 | 5 | 6 | 0 |
| y | 8 | 5 | 5 | 5 | | |

### 2.2.2. Optimized Bilinear Interpolation

Equation (13) appears to require eight multiplications, but careful factorization to exploit separability can reduce the multiplication by three [47].

$$\begin{cases} I_1 &= I(i,j) + u(I(i+1,j) - I(i,j)) \\ I_2 &= I(i,j+1) + v(I(i+1,j+1) - I(i,j+1)) \\ I(x,y) &= I_1 - v(I_2 - I_1) \end{cases} . \tag{16}$$

To compute the Equation (16), the gray values of four nearest-neighbors must be read from the memory. The principle of the mapping storage method is to optimize and balance the line and block data access rate [7]. In order to improve memory access efficiency, a multi-array memory storage method is designed in Figure 3. The RS image size is $m \times n$. As such, $m \times n$ memory units are designed in double data rate (DDR). The relationship between the address of DDR and $(i, j)$ coordinates can be described by Equation (17).

$$addr_{ij} = (i-1) \times m + (j-1) \times n, \tag{17}$$

where, $i$ and $j$ are the row and column of the raw image, respectively. $addr_{ij}$ is the corresponding address of memory.

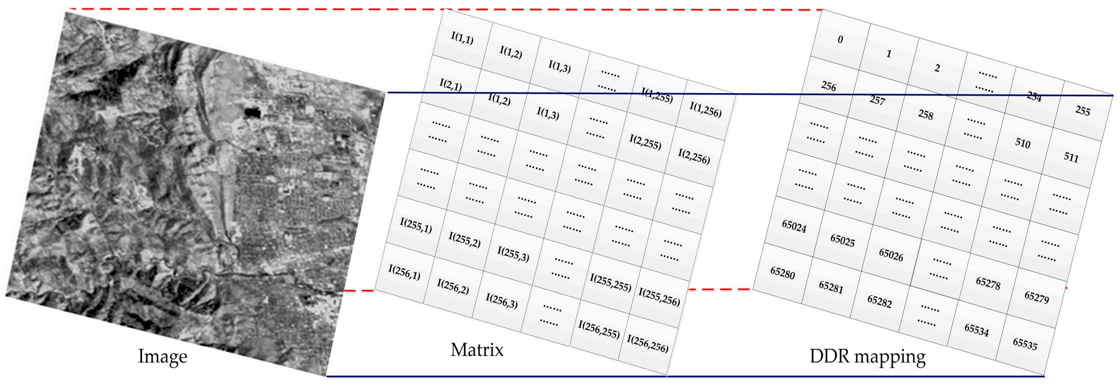


**Figure 3.** Raw image mapping with double data rate (DDR)

## 3. FPGA-Based Implementation of the Georeferencing Scheme

An FPGA hardware architecture is designed for georeferenced of the remotely sensed imagery as shown in Figure 4, which contains five modules: (I) GCP data module; (II) input image module (IIM); (III) coefficient calculation module (CCM); (IV) coordinate transformation module (CTM); and (V) bilinear interpolation module (BIM). The details of five modules are described as follows.

- The GCPs data is stored in the RAM of the GCP data module. These parameters are sent to the CCM, when the enable signal is being received.
- The gray values of the input image are saved to the ROM through the IIM, when the enable signal is being received.
- The coefficients of $a_0$, $a_1$, $a_2$, $a_3$, $a_4$, $a_5$, $b_0$, $b_1$, $b_2$, $b_4$, and $b_5$ are calculated in the CCM when the GCP data are arrived. The matrices $A^T A$ , $A^T L_x$, $A^T L_y$, and $(A^T A)^{-1}$ are parallel computed.

- The coordinate transformation is carried out in the CTM when the coefficients of second-order polynomial equation are arrived. The values of $i$, $j$, $u$, $1-u$, $v$ and $1-v$ are simultaneously sent to the BIM.
- According to Equations (13) and (16), the interpolation values are calculated. At last, the gray values, latitude, and longitude of the georeferenced image are outputted at the same clock cycle.

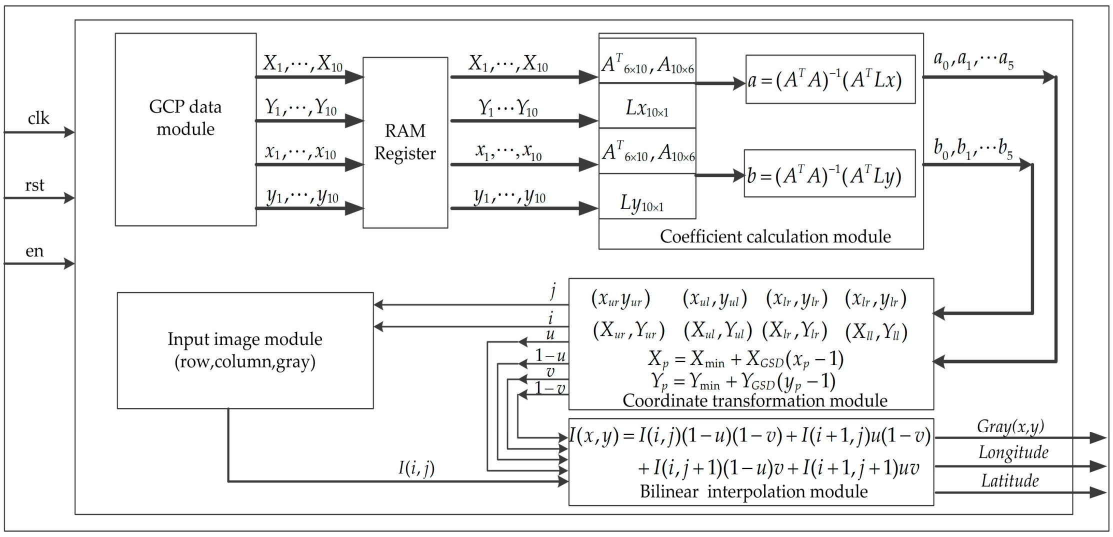

clk
rst
en
GCP data module
RAM Register
Coefficient calculation module
Input image module (row,column,gray)
Coordinate transformation module
Bilinear interpolation module
Gray(x,y)
Longitude
Latitude

**Figure 4.** A field programmable gate array (FPGA) architecture for the georeferenced remote sensing images with second-order polynomial equation and bilinear interpolation scheme. RAM: random access memory; clk: clock; rst: reset; en: enable.

## 3.1. *FPGA-Based Solution of the Second-Order Polynomial Equation*

To implement the second-order polynomial equation based on an FPGA, the processing of solving the coefficient is decomposed into three modules (as shown in Figure 5): (I) Form the matrices $A$ , $L_x$, and $L_y$; (II) Calculate $A^TA$ , $(A^TA)^{-1}$, $A^TL_x$, and $A^TL_y$, and (III) perform $(A^TA)^{-1}(A^TL_x)$ and $(A^TA)^{-1}(A^TL_y)$.

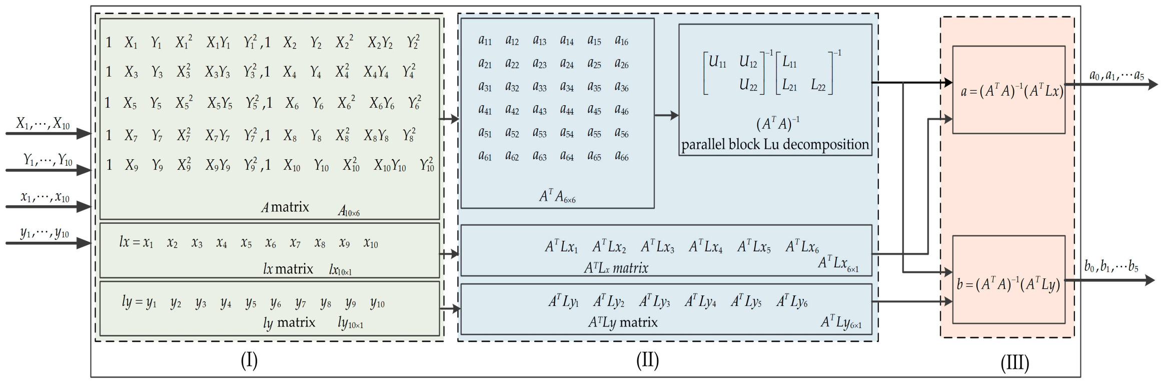


**Figure 5.** The proposed parallel implementation of the solving coefficient.

### 3.1.1. Calculation of Matrix $A^TA$

According to Equation (9) and the principle of matrix multiplication, the coefficients of $a_{1j}$ ($j = 1, 2, \ldots, 6$) are calculated, as shown in Equations (18) and (19).

$$A^TA = \begin{bmatrix} A_{11} & A_{21} & A_{31} & A_{41} & A_{51} & A_{61} & A_{71} & A_{81} & A_{91} & A_{101} \\ A_{12} & A_{22} & A_{32} & A_{42} & A_{52} & A_{62} & A_{72} & A_{82} & A_{92} & A_{102} \\ A_{13} & A_{23} & A_{33} & A_{43} & A_{53} & A_{63} & A_{73} & A_{83} & A_{93} & A_{103} \\ A_{14} & A_{24} & A_{34} & A_{44} & A_{54} & A_{64} & A_{74} & A_{84} & A_{94} & A_{104} \\ A_{15} & A_{25} & A_{35} & A_{45} & A_{55} & A_{65} & A_{75} & A_{85} & A_{95} & A_{105} \\ A_{16} & A_{26} & A_{36} & A_{46} & A_{56} & A_{66} & A_{76} & A_{86} & A_{96} & A_{106} \end{bmatrix} \times \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} & A_{16} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} & A_{26} \\ A_{31} & A_{32} & A_{33} & A_{34} & A_{35} & A_{36} \\ A_{41} & A_{42} & A_{43} & A_{44} & A_{45} & A_{46} \\ A_{51} & A_{52} & A_{53} & A_{54} & A_{55} & A_{56} \\ A_{61} & A_{62} & A_{63} & A_{64} & A_{65} & A_{66} \\ A_{71} & A_{72} & A_{73} & A_{74} & A_{75} & A_{76} \\ A_{81} & A_{82} & A_{83} & A_{84} & A_{85} & A_{86} \\ A_{91} & A_{92} & A_{93} & A_{94} & A_{95} & A_{96} \\ A_{101} & A_{102} & A_{103} & A_{104} & A_{105} & A_{106} \end{bmatrix}, \quad (18)$$

$$\begin{bmatrix} A_{11} \\ A_{21} \\ A_{31} \\ A_{41} \\ A_{51} \\ A_{61} \\ A_{71} \\ A_{81} \\ A_{91} \\ A_{101} \end{bmatrix}^T \times \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} & A_{16} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} & A_{26} \\ A_{31} & A_{32} & A_{33} & A_{34} & A_{35} & A_{36} \\ A_{41} & A_{42} & A_{43} & A_{44} & A_{45} & A_{46} \\ A_{51} & A_{52} & A_{53} & A_{54} & A_{55} & A_{56} \\ A_{61} & A_{62} & A_{63} & A_{64} & A_{65} & A_{66} \\ A_{71} & A_{72} & A_{73} & A_{74} & A_{75} & A_{76} \\ A_{81} & A_{82} & A_{83} & A_{84} & A_{85} & A_{86} \\ A_{91} & A_{92} & A_{93} & A_{94} & A_{95} & A_{96} \\ A_{101} & A_{102} & A_{103} & A_{104} & A_{105} & A_{106} \end{bmatrix} = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{15} \\ a_{16} \end{bmatrix}^T. \quad (19)$$

Generally, ten multipliers and nine adders are needed to compute $a_{11}$. Therefore, it takes about 324 additions and 360 multiplications to calculate all coefficients. However, the resources are limited on an FPGA chip. Therefore, a combination of serial and parallel strategy is adopted [48,49]. To improve the processing speed, multiplier-adder ($MD$) modules are used. At the same way, the other coefficients of $a_{1j}$, $a_{2j}$, $a_{3j}$, $a_{4j}$, $a_{5j}$, and $a_{6j}$ ($j = 1, 2, 3, \ldots, 6$) are parallel computed at the same clock cycle. Figure 6 shows the architecture of $A^TA$ based on an FPGA.

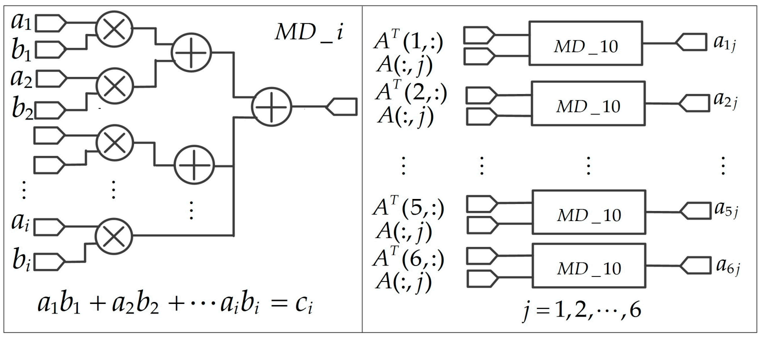


**Figure 6.** The architectures of $A^TA$ .

### 3.1.2. LU Decomposition of the Matrix $A^TA$

The coefficient accuracy of the second-order polynomial equation is crucial to the entire system. This paper proposes a parallel structure that uses the floating-point block LU decomposition to solve the inverse of $A^TA$ . The matrix $A$ can be represented as four matrices $A_{11}$, $A_{12}$, $A_{21}$, and $A_{22}$ [48–51], as shown in Equation (20).

$$A^TA = \left[\begin{array}{ccc|ccc} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ \hline a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} \end{array}\right] = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & \\ L_{21} & L_{22} \end{bmatrix} \times \begin{bmatrix} U_{11} & U_{12} \\ & U_{22} \end{bmatrix}, \quad (20)$$

where $A_{11}$, $A_{12}$, $A_{21}$, $A_{22}$, $L_{21}$ and $U_{12}$ are $3 \times 3$ matrices; $L_{11}$ and $L_{22}$ are $3 \times 3$ lower triangular matrices, $U_{11}$ and $U_{22}$ are $3 \times 3$ upper triangular matrices. From Equation (20), some additional equations can be achieved:

$$A_{11} = L_{11}U_{11}, \tag{21}$$

$$A_{12} = L_{11}U_{12}, \tag{22}$$

$$A_{21} = L_{21}U_{11}, \tag{23}$$

$$A_{22} = L_{21}U_{12} + L_{22}U_{22}. \tag{24}$$

The steps of LU decomposition are as follows.

Step 1: $A_{11}$ is performed by block LU decomposition; $L_{11}$ and $U_{11}$ are obtained.

Step 2: From Equation (22), $U_{12}$ can be solved by Equation (25):

$$U_{12} = L_{11}^{-1}A_{12}. \tag{25}$$

Step 3: From Equation (23), $L_{21}$ can be calculated by the product of $A_{21}$ and $(U_{11})^{-1}$ (see Equation (26)):

$$L_{21} = A_{21}U_{11}^{-1}. \tag{26}$$

Step 4: $A_{22} - A_{21}U_{12}$ matrix is performed by LU decomposition; $L_{22}$ and $U_{22}$ matrices are obtained.

1. FPGA-Based Parallel Computation for the LU Decomposition of Matrix $A_{11}$

To implement the LU decomposition of matrix $A_{11}$ on the FPGA, Equation (21) can be modified as Equation (27). The steps of the LU decomposition of matrix $A_{11}$ are as follows.

$$A_{11} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & & \\ l_{21} & 1 & \\ l_{31} & l_{32} & 1 \end{bmatrix} \times \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ & u_{22} & u_{23} \\ & & u_{33} \end{bmatrix} = L_{11}U_{11}, \tag{27}$$

Step 1. $u_{11} = a_{11}$, $u_{12} = a_{12}$, $u_{13} = a_{13}$, $l_{11} = l_{22} = l_{33} = 1$, $l_{21} = a_{21}/u_{11} = a_{21}/a_{11}$, $l_{31} = a_{31}/a_{11}$.

Step 2. $u_{22} = a_{22} - l_{21}u_{12}$, $u_{23} = a_{23} - l_{21}u_{13}$.

Step 3. $l_{32} = (a_{32} - l_{31}u_{12})/u_{22}$.

Step 4. $u_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23} = a_{33} - (l_{31}u_{13} + l_{32}u_{23})$.

According to Equation (27), a parallel computation architecture for matrices $L_{11}$ and $U_{11}$ is presented in Figure 7. Five multipliers, three divisors, one adder, and four subtractors are used. Additionally, some control signals are used to ensure that the results are outputted at the same clock cycle.

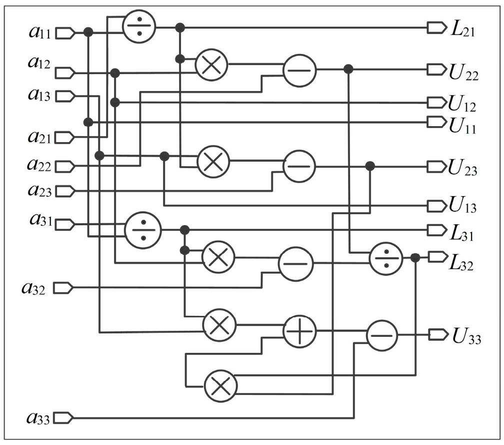


**Figure 7.** Parallel computation block lower-upper (LU) decomposition of $A_{11}$.

2. FPGA-Based Parallel Computation for Matrices $(L_{11})^{-1}$ and $(U_{11})^{-1}$

It is well known that the inverse of a lower triangular matrix is another lower triangular matrix, and the inverse of an upper triangular matrix is also another upper triangular matrix. The inversion of matrices $L_{11}$ and $U_{11}$ can be rewritten as Equations (28) and (29), respectively.

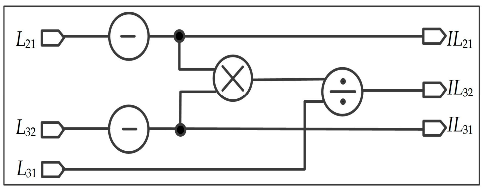
$$\begin{bmatrix} 1 & & \\ IL_{21} & 1 & \\ IL_{31} & IL_{32} & 1 \end{bmatrix} \times \begin{bmatrix} 1 & & \\ L_{21} & 1 & \\ L_{31} & L_{32} & 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}, \tag{28}$$

where $IL_{11} = -L_{21}$, $IL_{32} = -L_{32}$, and $IL_{31} = -IL_{32}L_{21} - L_{31}$.

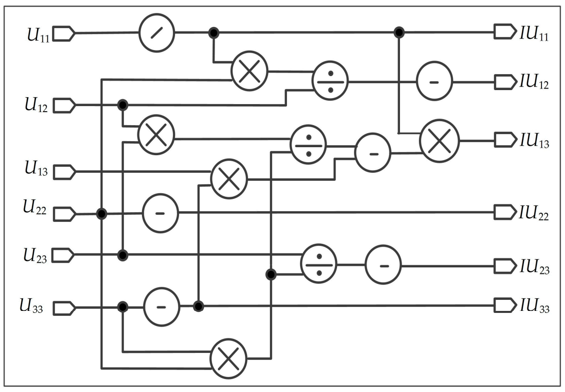
$$\begin{bmatrix} IU_{11} & IU_{12} & IU_{13} \\ & IU_{22} & IU_{23} \\ & & IU_{33} \end{bmatrix} \times \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & U_{33} \end{bmatrix} = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}, \tag{29}$$

where $IU_{11} = 1/U_{11}$, $IU_{22} = 1/U_{22}$, $IU_{33} = 1/U_{33}$, $IU_{12} = -(IU_{11}U_{12})/U_{22} = -U_{12}/U_{11}U_{22}$, $IU_{23} = -(IU_{22}U_{23})/U_{33} = -U_{23}/U_{22}U_{33}$, and $IU_{13} = -(IU_{11}U_{13} + IU_{12}U_{23})/U_{33}$.

The parallel computation structures for calculating $(L_{11})^{-1}$ and $(U_{11})^{-1}$ are shown in Figure 8. Six multipliers, four divisors, one subtractor ("—" in the circle), and three reciprocals ("/" in the circle) are used. The reverse operation ("-" in the circle) for floating-point is simply to reverse the symbol bit and require very few resources. Additionally, some control signals are used to ensure that the results are outputted at the same clock cycle.



(a)



(b)

**Figure 8.** The architecture FPFA-based for $(L_{11})^{-1}$ and $(U_{11})^{-1}$ matrices. (**a**) The architecture for solving $(L_{11})^{-1}$. (**b**) The architecture for solving $(U_{11})^{-1}$.

3. FPGA-Based Implementation of Matrices $U_{12}$ and $L_{21}$

To implement $U_{12}$ and $L_{21}$ based on the FPGA, Equations (25) and (26) can be rewritten into Equations (30) and (31). In Equation (30), the elements of the matrix contain three formats, i.e., $a_{14}$, $IL_{21}a_{14} + a_{24}$, and $IL_{31}a_{14} + IL_{32}a_{24} + a_{34}$. In Equation (31), the elements of the matrix contain three formats, i.e., $a_{41}IU_{11}$, $a_{41}IU_{12} + a_{42}IU_{22}$, and $a_{41}IU_{13} + a_{42}IU_{23} + a_{43}IU_{33}$. The proposed parallel architecture for calculation of $L_{21}$ and $U_{12}$ is depicted in Figure 9. Twelve multipliers, fifteen $MD$ modules, and three adders are used.

$$\begin{aligned} U_{12} = \quad & (L_{11})^{-1}\mathrm{A}_{12} = \begin{bmatrix} 1 & & \\ IL_{21} & 1 & \\ IL_{31} & IL_{32} & 1 \end{bmatrix} \times \begin{bmatrix} a_{14} & a_{15} & a_{16} \\ a_{24} & a_{25} & a_{26} \\ a_{34} & a_{35} & a_{36} \\ & & \end{bmatrix} \\ &= \begin{bmatrix} a_{14} & a_{15} & a_{16} \\ IL_{21}a_{14} + a_{24} & IL_{21}a_{15} + a_{25} & IL_{21}a_{16} + a_{26} \\ IL_{31}a_{14} + IL_{32}a_{24} + a_{34} & IL_{31}a_{15} + IL_{32}a_{25} + a_{35} & IL_{31}a_{16} + IL_{32}a_{26} + a_{36} \\ & & \end{bmatrix} \\ &= \begin{bmatrix} U_{12\_11} & U_{12\_12} & U_{12\_13} \\ U_{12\_21} & U_{12\_22} & U_{12\_23} \\ U_{12\_31} & U_{12\_32} & U_{12\_33} \end{bmatrix} \end{aligned} \tag{30}$$

$$\begin{aligned} L_{21} \quad &= A_{21}{U_{11}}^{-1} = \begin{bmatrix} a_{41} & a_{42} & a_{43} \\ a_{51} & a_{52} & a_{53} \\ a_{61} & a_{62} & a_{63} \end{bmatrix} \times \begin{bmatrix} IU_{11} & IU_{12} & IU_{13} \\ & IU_{22} & IU_{23} \\ & & IU_{33} \\ & & \end{bmatrix} \\ &= \begin{bmatrix} a_{41}IU_{11} & a_{41}IU_{12} + a_{42}IU_{22} & a_{41}IU_{13} + a_{42}IU_{23} + a_{43}IU_{33} \\ a_{51}IU_{11} & a_{51}IU_{12} + a_{52}IU_{22} & a_{51}IU_{13} + a_{52}IU_{23} + a_{53}IU_{33} \\ a_{61}IU_{11} & a_{61}IU_{12} + a_{62}IU_{22} & a_{61}IU_{13} + a_{62}IU_{23} + a_{63}IU_{33} \\ & & \end{bmatrix} \\ &= \begin{bmatrix} L_{21\_11} & L_{21\_12} & L_{21\_13} \\ L_{21\_21} & L_{21\_22} & L_{21\_23} \\ L_{21\_31} & L_{21\_32} & L_{21\_33} \end{bmatrix} \end{aligned} \tag{31}$$

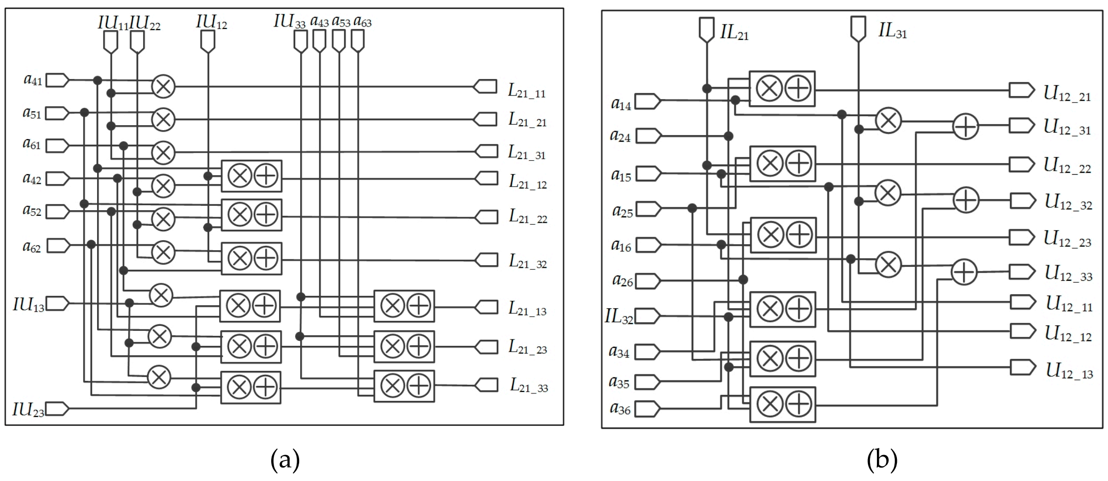

$IU_{11}$ $IU_{22}$ $IU_{12}$ $IU_{33}$ $a_{43}$ $a_{53}$ $a_{63}$
$a_{41}$ $a_{51}$ $a_{61}$ $a_{42}$ $a_{52}$ $a_{62}$ $IU_{13}$ $IU_{23}$
$L_{21\_11}$ $L_{21\_21}$ $L_{21\_31}$ $L_{21\_12}$ $L_{21\_22}$ $L_{21\_32}$ $L_{21\_13}$ $L_{21\_23}$ $L_{21\_33}$
(a)
$IL_{21}$ $IL_{31}$
$a_{14}$ $a_{24}$ $a_{15}$ $a_{25}$ $a_{16}$ $a_{26}$ $IL_{32}$ $a_{34}$ $a_{35}$ $a_{36}$
$U_{12\_21}$ $U_{12\_31}$ $U_{12\_22}$ $U_{12\_32}$ $U_{12\_23}$ $U_{12\_33}$ $U_{12\_11}$ $U_{12\_12}$ $U_{12\_13}$
(b)

**Figure 9.** Parallel computation method for $L_{21}$ and $U_{12}$. (**a**) The architecture of $L_{21}$. (**b**) The architecture of $U_{12}$.

## 4. FPGA-Based Implementation of $L_{22}$ and $U_{22}$

To implement $L_{22}$ and $U_{22}$, Equation (24) can be described by

$$\begin{aligned} NewA_{22} &= A_{22} - L_{21}U_{12} \\ &= \begin{bmatrix} a_{44} & a_{45} & a_{46} \\ a_{54} & a_{55} & a_{56} \\ a_{63} & a_{65} & a_{66} \end{bmatrix} - \begin{bmatrix} L_{21\_11} & L_{21\_12} & L_{21\_13} \\ L_{21\_21} & L_{21\_22} & L_{21\_23} \\ L_{21\_31} & L_{21\_32} & L_{21\_33} \end{bmatrix} \times \begin{bmatrix} U_{12\_11} & U_{12\_12} & U_{12\_13} \\ U_{12\_21} & U_{12\_22} & U_{12\_23} \\ U_{12\_31} & U_{12\_32} & U_{12\_33} \end{bmatrix} \\ &= \begin{bmatrix} 1 & & \\ L_{22\_21} & 1 & \\ L_{22\_31} & L_{22\_32} & 1 \end{bmatrix} \times \begin{bmatrix} U_{22\_11} & U_{22\_12} & U_{22\_13} \\ & U_{22\_22} & U_{22\_23} \\ & & U_{22\_33} \end{bmatrix} \end{aligned} \tag{32}$$

where $NewA_{22}$ is a $3 \times 3$ matrix, with a similar format of $A_{11}$. Therefore, the LU decomposition of $NewA_{22}$ is not deduced in details. The parallel implementation is shown in Figure 10.

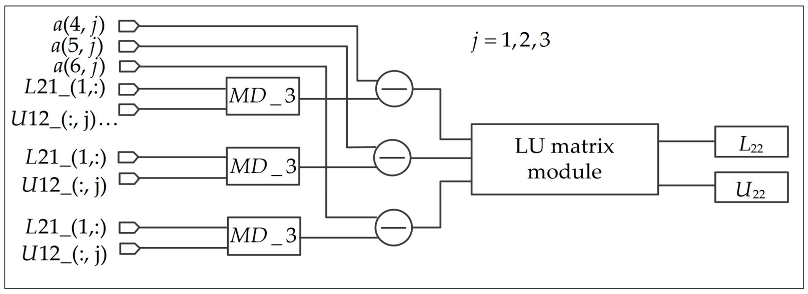


**Figure 10.** The architecture of $L_{22}$ and $U_{22}$.

For the Block LU decomposition, the number of multiplications is about $n^3/3 + (n\%d - 0.5)n^2$, and division is $(d+1)n/2$ (where $d$ is the dimension of each block, $n$ is the size of matrix). In the standard LU decomposition, the number of multiplication operations is about $n(2n-1)(n-1)/6\,n$, and division is $n(n-1)/2$ [52]. The multipliers and dividers based block LU decomposition are approximately reduced 1.02 and 1.25 times than that the traditional LU decomposition.

### 3.1.3. FPGA-Based Implantation of the Matrix $(A^TA)^{-1}$

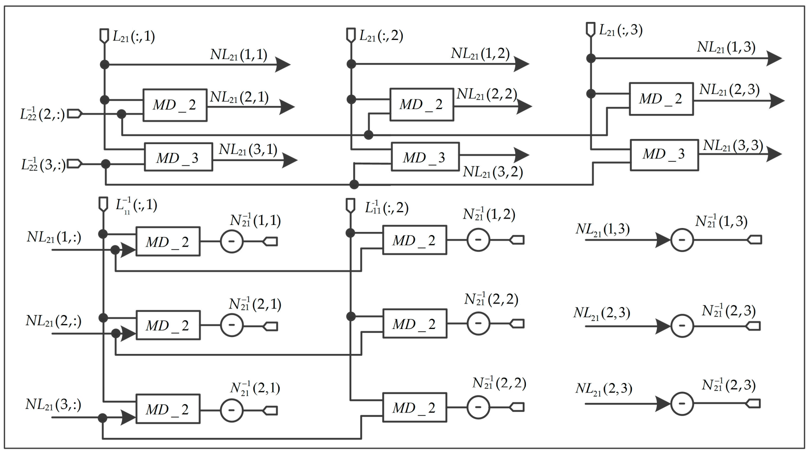
When $L_{11}$, $L_{21}$, $L_{22}$, $U_{11}$, $U_{12}$, and $U_{22}$ matrices are solved, the block LU decomposition of matrix $A^TA$ has been completed,. The processing of inversion matrix $A^TA$ is based on the Equation (33).

$$\begin{aligned}(A^TA)^{-1} &= B = \left(\begin{bmatrix} L_{11} & \\ L_{21} & L_{22}\end{bmatrix}\begin{bmatrix} U_{11} & U_{12} \\ & U_{22}\end{bmatrix}\right)^{-1} = \begin{bmatrix} U_{11} & U_{12} \\ & U_{22}\end{bmatrix}^{-1}\begin{bmatrix} L_{11} & \\ L_{21} & L_{22}\end{bmatrix}^{-1}, \\ &= \begin{bmatrix} U_{11}^{-1}L_{11}^{-1} + M_{12}^{-1}N_{21}^{-1} & M_{12}^{-1}L_{22}^{-1} \\ U_{22}^{-1}N_{21}^{-1} & U_{22}^{-1}L_{22}^{-1}\end{bmatrix}\end{aligned} \tag{33}$$

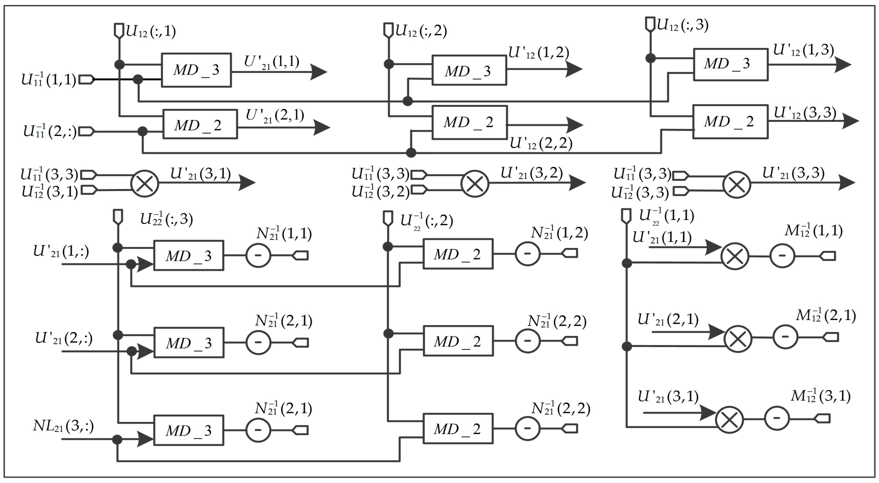
where $N_{21}^{-1} = -L_{22}^{-1}L_{21}L_{11}^{-1}$ and $M_{12}^{-1} = -U_{11}^{-1}U_{12}U_{22}^{-1}$.

1. FPGA-Based Implementation of $N_{21}^{-1}$ and $M_{12}^{-1}$

Figure 11 shows the implementation of $N_{21}^{-1}$ and $M_{12}^{-1}$, It contains *MD*_3 module, *MD*_2 module and negative operation.



(a)

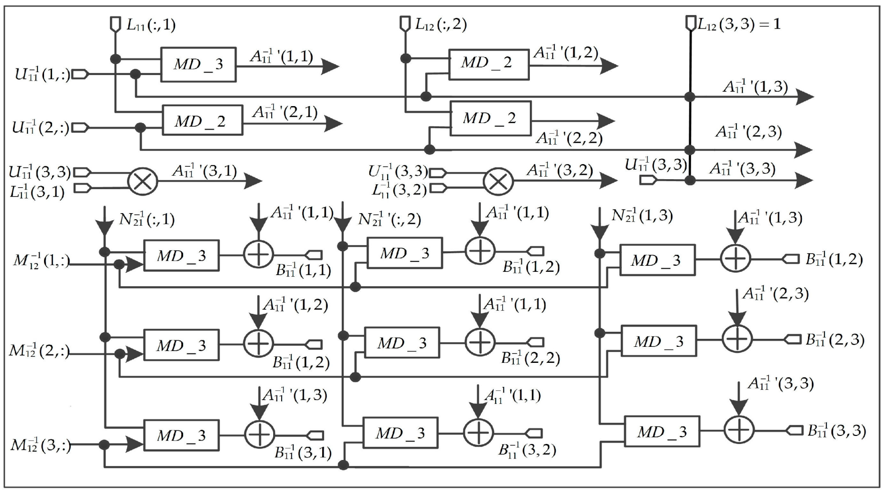


(b)

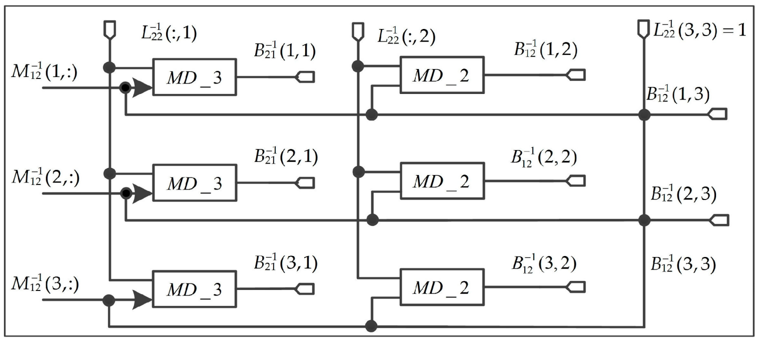
**Figure 11.** Implementation of $N_{21}^{-1}$ and $M_{12}^{-1}$ (**a**) The architecture of $N_{21}^{-1}$. (**b**) The architecture of $M_{12}^{-1}$.

2. FPGA-Based Implementation of $(A^TA)^{-1}$

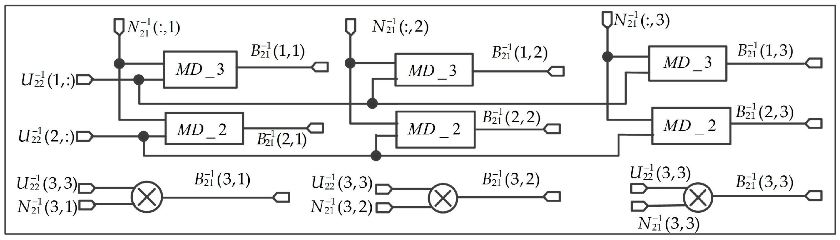
The scheme for $(A^TA)^{-1}$ is shown in Figure 12. Figure 12a is an FPGA implementation of solving the $U_{11}^{-1}L_{11}^{-1} + M_{21}^{-1}N_{21}^{-1}$ matrix. Ten *MD*_3 modules, three *MD*_2 modules, two multipliers, and nine adders are used to solve the $U_{11}^{-1}L_{11}^{-1} + M_{21}^{-1}N_{21}^{-1}$ matrix. Figure 12b shows the parallel computation for $M_{21}^{-1}L_{22}^{-1}$ matrix, three *MD*_3 modules, and three *MD*_2 modules are used. The structure for solving the $U_{22}^{-1}N_{21}^{-1}$ matrix is shown in Figure 12c, three *MD*_3 modules, three *MD*_2 modules, and three multipliers are used. Figure 12d shows the implementation of the $U_{22}^{-1}L_{22}^{-1}$ matrix, one *MD*_3 module, two *MD*_2 modules, and two multipliers are used. The Figure 12a through Figure 12d are parallel calculated. Finally, $(A^TA)^{-1}$ matrix is obtained and the elements are outputted at the same clock cycle.



(a)

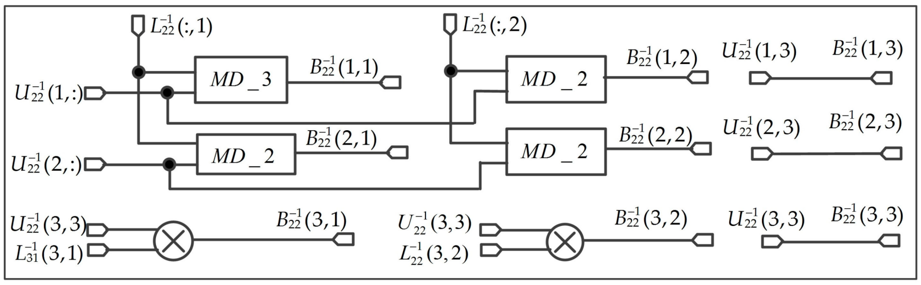


(b)



(c)

**Figure 12.** *Cont.*



(d)

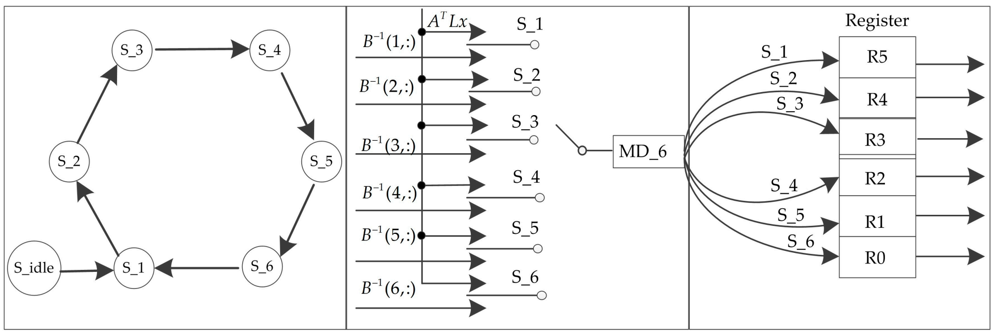
**Figure 12.** The implementation of $(A^TA)^{-1}$. (**a**) Calculate $U_{11}^{-1}L_{11}^{-1} + M_{21}^{-1}N_{21}^{-1}$ matrix. (**b**) Solve $M_{21}^{-1}L_{22}^{-1}$ matrix. (**c**) Calculate $U_{22}^{-1}N_{21}^{-1}$ matrix. (**d**) Compute $M_{21}^{-1}L_{22}^{-1}$ matrix.

### 3.1.4. FPGA-Based Implementation of *a* and *b Matrices*

From the ten GCPs data, $L_x$ and $L_y$ matrix can be obtained, since $a$ and $b$ matrices have the same structure, such as $A^TL_x$, $(A^TA)^{-1}(A^TL_x)$, $A^TL_y$, and $(A^TA)^{-1}(A^TL_y)$, the implementation of $a$ is given to illustrate the processes. From Equation (34), six *MD*_10 modules are adopted. Six *MD*_6 modules are needed for solving Equation (35). Considering the limited resources of an FPGA, a serial framework is used to implement $a$ , the strategy is depicted in Figure 13 (such as $(A^TA)^{-1}(A^TL_x)$).

The details of state transition graph (STG) are as follows:

- S_idle, idle state. When the rst signal is low, the system is in the reset state, all registers (R0, R1, ..., R5) and other signals are reset.
- S_1 to S_6 are the six different state machines (SMs). Under the different SMs, different row values of the matrix $(A^TA)^{-1}$ with matrix $A^TL_x$ are calculated in the *MD*_6 modules. The result of *MD*_6 is serial saved in the registers R5 to R0. When the six SMs are finished, the values of registers R5 to R0 are parallel outputted to the matrix $a$ .
- S_1, the first SM. When the rst signal goes high, the current state enters the first state when the enable signal is being received. The values of matrix $B^{-1}(1,:) = (A^TA)^{-1}(1,:)$ and $A^TL_x$ are put into the *MD*_6 module for multiplication and addition. The result is saved in the register R5.
- S_2, the second SM. When the S_1 state is complete, the current state enters the second state. The values of matrix $B^{-1}(2,:) = (A^TA)^{-1}(2,:)$ and $A^TL_x$ begin to calculate multiplication and addition. Then, the result is saved in the register R4.



(a) (b) (c)

**Figure 13.** The processing of state transition graph (STG). (**a**) The state machine. (**b**) Serial framework. (**c**) Serial input and parallel output.

In this order, when the S_6 SM (final State) is completed, the matrix $a$ is derived. Under the same clock cycle, the values of the registers R5 to R0 ($a_0$–$a_5$) are parallel outputted. The current state returns to the first state. Additionally, each state should contain a certain delay time. The delay time should include *MD_6* module operation time and serial storage time.

$$
\begin{aligned}
A^T Lx &= \begin{bmatrix} A_{11} & A_{21} & A_{31} & A_{41} & A_{51} & A_{61} & A_{71} & A_{81} & A_{91} & A_{101} \\ A_{12} & A_{22} & A_{32} & A_{42} & A_{52} & A_{62} & A_{72} & A_{82} & A_{92} & A_{102} \\ A_{13} & A_{23} & A_{33} & A_{43} & A_{53} & A_{63} & A_{73} & A_{83} & A_{93} & A_{103} \\ A_{14} & A_{24} & A_{34} & A_{44} & A_{54} & A_{64} & A_{74} & A_{84} & A_{94} & A_{104} \\ A_{15} & A_{25} & A_{35} & A_{45} & A_{56} & A_{65} & A_{75} & A_{85} & A_{95} & A_{104} \\ A_{16} & A_{26} & A_{36} & A_{46} & A_{56} & A_{66} & A_{76} & A_{86} & A_{96} & A_{106} \end{bmatrix} \begin{bmatrix} lx_1 \\ lx_2 \\ lx_3 \\ lx_4 \\ \vdots \\ lx_{10} \end{bmatrix} = \begin{bmatrix} Lx_1 \\ Lx_2 \\ Lx_3 \\ Lx_4 \\ Lx_5 \\ Lx_6 \end{bmatrix} \\
&= \begin{bmatrix} A_{11}lx_1 + A_{21}lx_2 + A_{31}lx_3 + A_{41}lx_4 + A_{51}lx_5 + A_{61}lx_6 + A_{71}lx_7 + A_{81}lx_8 + A_{91}lx_9 + A_{101}lx_{10} \\ A_{12}lx_1 + A_{22}lx_2 + A_{32}lx_3 + A_{42}lx_4 + A_{52}lx_5 + A_{62}lx_6 + A_{72}lx_7 + A_{82}lx_8 + A_{92}lx_9 + A_{102}lx_{10} \\ A_{13}lx_1 + A_{23}lx_2 + A_{33}lx_3 + A_{43}lx_4 + A_{53}lx_5 + A_{63}lx_6 + A_{73}lx_7 + A_{83}lx_8 + A_{93}lx_9 + A_{103}lx_{10} \\ A_{14}lx_1 + A_{24}lx_2 + A_{34}lx_3 + A_{44}lx_4 + A_{54}lx_5 + A_{64}lx_6 + A_{74}lx_7 + A_{84}lx_8 + A_{94}lx_9 + A_{104}lx_{10} \\ A_{15}lx_1 + A_{25}lx_2 + A_{35}lx_3 + A_{45}lx_4 + A_{55}lx_5 + A_{65}lx_6 + A_{75}lx_7 + A_{85}lx_8 + A_{95}lx_9 + A_{105}lx_{10} \\ A_{16}lx_1 + A_{26}lx_2 + A_{36}lx_3 + A_{46}lx_4 + A_{56}lx_5 + A_{66}lx_6 + A_{76}lx_7 + A_{86}lx_8 + A_{96}lx_9 + A_{106}lx_{10} \end{bmatrix},
\end{aligned} \tag{34}
$$

$$
a = (A^T A)^{-1}(A^T L_x) = \begin{bmatrix} B_{11}^{-1} & B_{12}^{-1} \\ B_{21}^{-1} & B_{22}^{-1} \end{bmatrix} \begin{bmatrix} Lx_1 \\ Lx_2 \\ Lx_3 \\ Lx_4 \\ Lx_5 \\ Lx_6 \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}. \tag{35}
$$

## *3.2. FPGA-Based Implementation of Coordinate Transformation and Bilinear Interpolation Algorithm*

Figure 14 shows the flowchart for the coordinate transformation and bilinear interpolation method, which consists of the transformation from the output image coordinates to the ground coordinates, conversion the ground coordinate to the raw image coordinates, and bilinear interpolation.

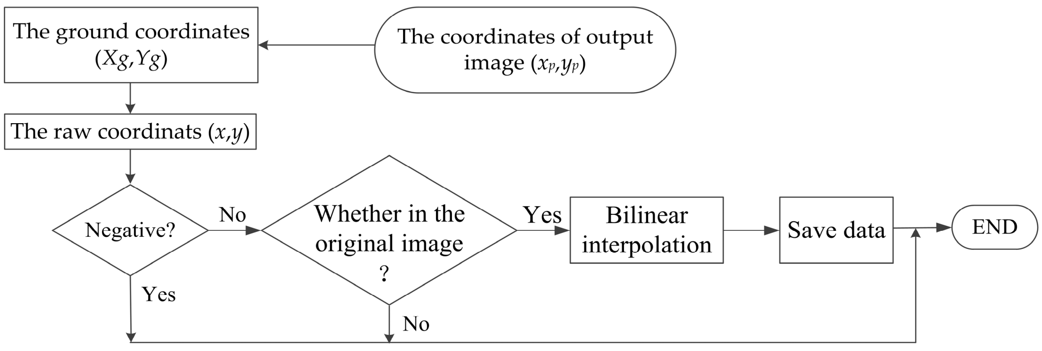

The ground coordinates (Xg,Yg)
The coordinates of output image ($x_p$,$y_p$)
The raw coordinats ($x$,$y$)
Negative?
No
Yes
Whether in the original image ?
Yes
No
Bilinear interpolation
Save data
END

**Figure 14.** The flowchart of the coordinate transformation and bilinear interpolation.

1. FPGA-Based Implementation of $X_g$ and $Y_g$

The purpose of this step is to obtain $X_g$ and $Y_g$ in Equation (12) of Section 2.1.2, as shown in Figure 15. In this part, 32-bit integer and shift register are used to reduce the resource utilization of an FPGA. To implement the shift operation, $X_{GSD}$ and $Y_{GSD}$ are approximated as $2^m$ or $2^m - 2^n$ ($m$ and $n$ are integers). For instance, $X_{GSD} = Y_{GSD} = 30$, which can be expressed as the form of $2^5 - 2$. In Figure 15, the symbol "<<" in the circle represents the left shift operation.

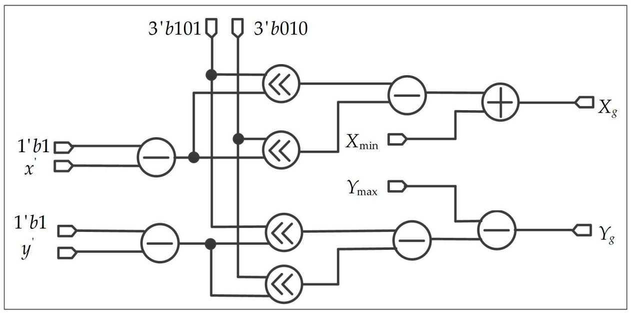


**Figure 15.** Parallel computation method for $X_g$ and $Y_g$.

2. FPGA-Based Implementation of Bilinear Interpolation

When $X_g$ and $Y_g$ are calculated, the coordinates of *x_row* and *y_column* are obtained from Equations (14) and (15). The bilinear interpolation algorithm is performed based on the *x_row* and *y_column* as shown in Figure 16. As observed from Figure 16, the processing of bilinear interpolation is divided into three steps.

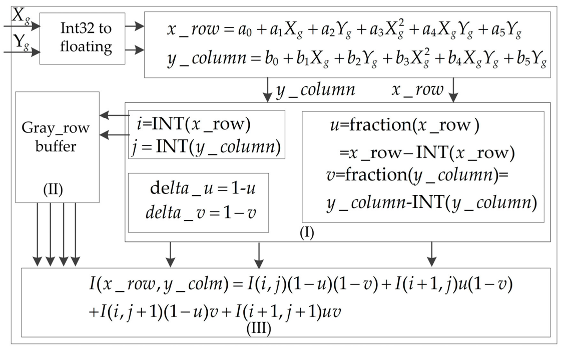

X_g
Y_g
Int32 to floating
x_row = a_0 + a_1X_g + a_2Y_g + a_3X_g^2 + a_4X_gY_g + a_5Y_g
y_column = b_0 + b_1X_g + b_2Y_g + b_3X_g^2 + b_4X_gY_g + b_5Y_g
y_column
x_row
Gray_row buffer
(II)
i=INT(x_row)
j = INT(y_column)
delta_u = 1-u
delta_v = 1−v
u=fraction(x_row) =x_row−INT(x_row)
v=fraction(y_column)= y_column-INT(y_column)
(I)
I(x_row, y_colm) = I(i, j)(1−u)(1−v) + I(i+1, j)u(1−v) +I(i, j+1)(1−u)v + I(i+1, j+1)uv
(III)

**Figure 16.** The diagram of bilinear interpolation.

In step (I), the integer part $i$, $j$, the fractional part $u$, $v$ and the weight part $1-u$ and $1-v$ of the floating-point $(x\_row, y\_column)$ coordinates are computed, respectively. Figure 17 shows the parallel implementation method for $i$, $j$, $u$, $v$, $1-u$ and $1-v$. Four subtractors, two INT (integer) modules and two absolute modules ("| |" in the circle) are used.

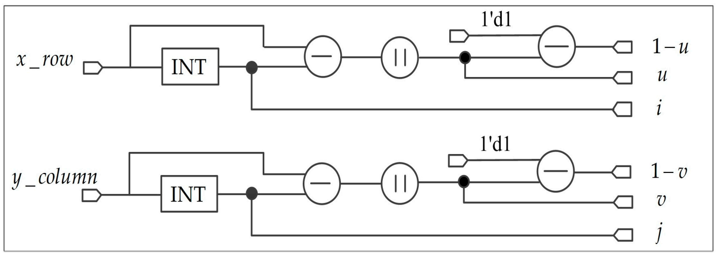


**Figure 17.** The parallel implementation method for $i$, $j$, $u$, $v$, $1-u$ and $1-v$.

In step (II), it is mainly to calculate the address of four nearest neighbor pixels and read the corresponding gray value. The address of memory can be calculated from Equation (17) of Section 2.2.2. The processing of reading gray value is shown in Figure 18. To reduce the resources of an FPGA, the multiplication is converted into left shift operation.

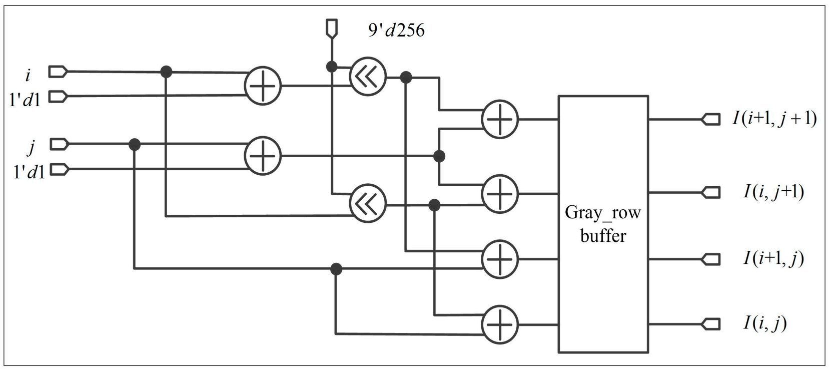


**Figure 18.** The architecture for parallel reading gray values.

After the values of $i$, $j$, $u$, $v$, $1-u$, $1-v$, $I(i,j)$, $I(i+1,j)$, $I(i,j+1)$, and $I(i+1,j+1)$ are computed, the gray value of the georeferenced image can be calculated in step (III). According to Equations (13) and (16). The implementation of bilinear interpolation algorithm is shown in Figure 19. Eight multipliers and three adders are used to compute the value of $I(x-row, y\_column)$.

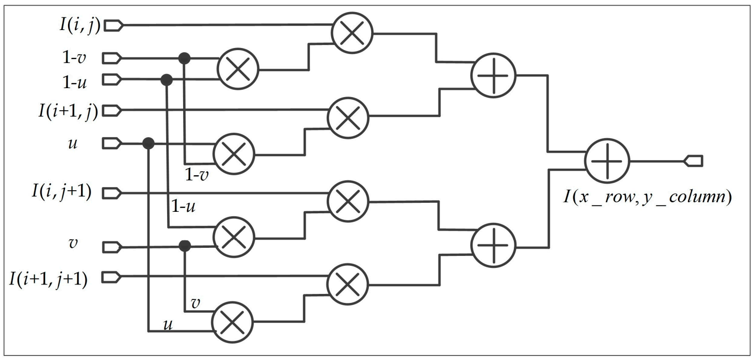


**Figure 19.** The parallel computation for bilinear interpolation.

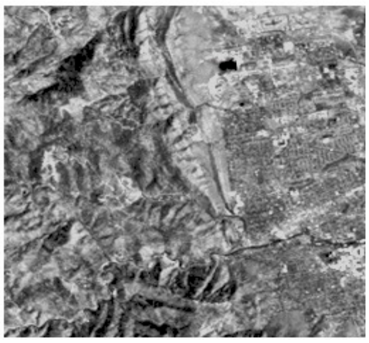
## 4. Experiment and Performance Analysis

### *4.1. The Software and Hardware Environment*

The proposed scheme is implemented on a custom-designed board which contains a Xilinx Virtex-7-XC7VX980t-ffg1930-1 FPGA that has 612,000 logic cells, 1,500 kB Block RAM, 1,224,000 Flip-Flops, and 3,600 DSP slices. Additionally, the design tool is Vivado 2014.2, the simulation tool is ModelSim SE-64 10.4, and the hardware design language is Verilog HDL. To validate the proposed method, the georeferenced algorithm is also implemented by MATLAB R2014a, Visual Studio 2015 (C++) and ENVI 5.3 on a PC equipped with an Intel (R) Core i7-4790 CPU @ 3.6 GHz and 8 GB RAM, running Windows 7 (64 bit).

### *4.2. Data*

To validate the proposed FPGA-based algorithm, two data sets are used to perform the georeferencing. The first data set is acquired from the ENVI example dataset, i.e., bldt_tm.img and bldt_tm.pts. The second data set is obtained from the ERDAS example dataset, i.e., tmAtlanta.img and panAtanta.img. Figure 20a,b display the raw data sets of bldt_tm.img and tmAtlanta.img, respectively. The image size is $256 \times 256$ pixels$^2$. The information of two datasets is shown in Table 2.


(a)

(b)

**Figure 20.** The raw image. (**a**) The first raw image. (**b**) The second raw image.

**Table 2.** The information of two data sets.

| Image | Projection | Zone | Resolution (m) | Band | Wave Length (μm) |
|---|---|---|---|---|---|
| bldt_tm.img | UTM | 13 | 30 × 30 | 3 | 0.63~0.69 |
| tmAtlanta.img | State plan (NAD 27) | 3676 | 30 × 30 | 4 | 0.76~0.90 |

## 4.3. Processing Performance

### 4.3.1. Error Analysis

To quantitatively evaluate the accuracy of the georeferencing implemented using FPGA, the root mean squared error (*RMSE*) is used in Equation (36) through Equation (38) [45].

$$RMSE_x = \sqrt{\frac{\sum_{k=1}^{n}(x'_k - x_k)^2}{n}}, \tag{36}$$

$$RMSE_y = \sqrt{\frac{\sum_{k=1}^{n}(y'_k - y_k)^2}{n}}, \tag{37}$$

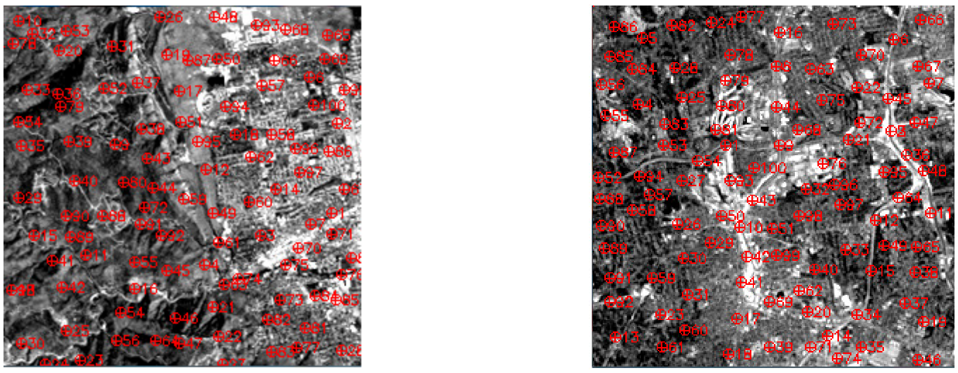
$$RMSE = \sqrt{RMSE_x^2 + RMSE_y^2}, \tag{38}$$

where $x'_k$ and $y'_k$ are coordinates of the georeferenced image which are computed by the proposed method; $x_k$ and $y_k$ are reference geodetic coordinates; and $n$ is the number of check points (CPs).

To compute the root mean squared errors (*RMSE*s), one hundred CPs are selected as shown in Figure 21.

With the experimental results, a few conclusions can be drawn as follows.

(1) From Equation (36) through (38), the *RMSE*s are computed based on FPGA, MATLAB, Visual Studio (C++), and ENVI, as shown in Table 3. It can be concluded that the $RMSE_x$, $RMSE_y$, and *RMSE* of the georeferenced image of the first data set implemented using FPGA are 0.1441, 0.1672, and 0.2207 pixels, respectively. The accuracy of the georeferenced using FPGA has the same values when implemented by MATLAB and Visual studio (C++), and is close to the values implemented using ENVI software. Additionally, other statistics, such as maximum, minimum and mean error are computed and listed in Table 4. It can be found that the proposed FPGA-based algorithm has a mean error of x and y coordinates at 0.0775 pixels and 0.0945 pixels, respectively; the minimum and the maximum errors of the $x$ and $y$ coordinates are 0.0008 and 0.0026 pixels, and 0.5113 and 0.4038 pixels, respectively. The accuracy of the various errors are calculated by the proposed algorithm has the same values than those based on MATLAB and Visual Studio (C++), and has a close accuracy than that based on ENVI software.



(a) (b)

**Figure 21.** Check points distribution in: (**a**) the first image and (**b**) the second image.

**Table 3.** The values of root mean square errors (*RMSE*s) of the georeferenced image of the first image.

| Platform | $RMSE_x$ | $RMSE_y$ | $RMSE$ |
|---|---|---|---|
| FPGA | 0.1441 | 0.1672 | 0.2207 |
| MATLAB | 0.1441 | 0.1672 | 0.2207 |
| Visual Studio (C++) | 0.1441 | 0.1672 | 0.2207 |
| ENVI | 0.1442 | 0.1699 | 0.2229 |

**Table 4.** Statistics values of the georeferenced image of the first image.

| Platform | Coordinate | Max Error (pixel) | Min Error (pixel) | Mean Error (pixel) |
|---|---|---|---|---|
| FPGA | $x$ | 0.5113 | 0.0008 | 0.0775 |
| | $y$ | 0.4038 | 0.0026 | 0.0945 |
| MATLAB | $x$ | 0.5113 | 0.0008 | 0.0775 |
| | $y$ | 0.4038 | 0.0026 | 0.0945 |
| Visual Studio (C++) | $x$ | 0.5113 | 0.0008 | 0.0775 |
| | $y$ | 0.4038 | 0.0026 | 0.0945 |
| ENVI | $x$ | 0.5252 | 0.0008 | 0.0829 |
| | $y$ | 0.5252 | 0.0008 | 0.1031 |

(2) Table 5 lists the *RMSE*s of the georeferenced image of the second data set. The values of the $RMSE_x$, $RMSE_y$, and $RMSE$ implemented by FPGA are 0.0965, 0.1268, and 0.1593 pixels, respectively. The *RMSE*s implemented using FPGA have the same accuracy than those based on MATLAB and Visual Studio (C++), and have a close accuracy implemented using ENVI software. However, it can be considered acceptable for the absolute error is less than one pixel [53]. In addition, maximum, minimum and mean errors are computed and listed in Table 6. It can be found that the mean errors implemented using FPGA of the $x$ and $y$ coordinates are 0.0789 and 0.1144 pixels, respectively, and the minimum and maximum errors for the $x$ and $y$ coordinates are 0.0008 and 0.0046 pixels, and 0.2613 and 0.2081 pixels, respectively.

**Table 5.** The values of *RMSE*s of the georeferenced image of the second image.

| Platform | $RMSE_x$ | $RMSE_y$ | $RMSE$ |
|---|---|---|---|
| FPGA | 0.0965 | 0.1268 | 0.1593 |
| MATLAB | 0.0965 | 0.1268 | 0.1593 |
| Visual Studio (C++) | 0.0965 | 0.1268 | 0.1593 |
| ENVI | 0.0897 | 0.1009 | 0.1350 |

**Table 6.** Statistics values of the georeferenced image of the second image.

| Platform | Coordinate | Max Error (pixel) | Min Error (pixel) | Mean Error (pixel) |
|---|---|---|---|---|
| FPGA | $x$ | 0.2613 | 0.0008 | 0.0789 |
| | $y$ | 0.2081 | 0.0046 | 0.1144 |
| MATLAB | $x$ | 0.2613 | 0.0008 | 0.0789 |
| | $y$ | 0.2081 | 0.0046 | 0.1144 |
| Visual Studio (C++) | $x$ | 0.2613 | 0.0008 | 0.0789 |
| | $y$ | 0.2081 | 0.0046 | 0.1144 |
| ENVI | $x$ | 0.4039 | 0.0003 | 0.0588 |
| | $y$ | 0.4039 | 0.0003 | 0.0673 |

As observed form Table 3 through 6, the accuracy of the georeferenced image when implemented using FPGA can reach the requirements because its *RMSE*s are less than one pixel [54].

Figure 22a–d show the georeferenced images of the first data set implemented by FPGA, MATLAB, Visual Studio (C++) and ENVI, respectively. Figure 23a–d show georeferenced images of the second image implemented by FPGA, MATLAB, Visual Studio (C++), and ENVI, respectively.

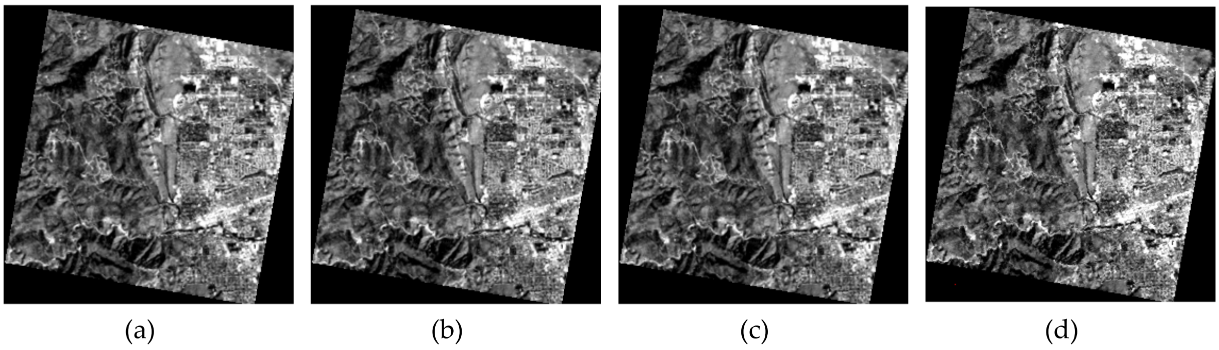


**Figure 22.** The georeferenced image of the first data set. (**a**) By FPGA. (**b**) By MATLAB. (**c**) By Visual Studio (C++). (**d**) By ENVI 5.3.

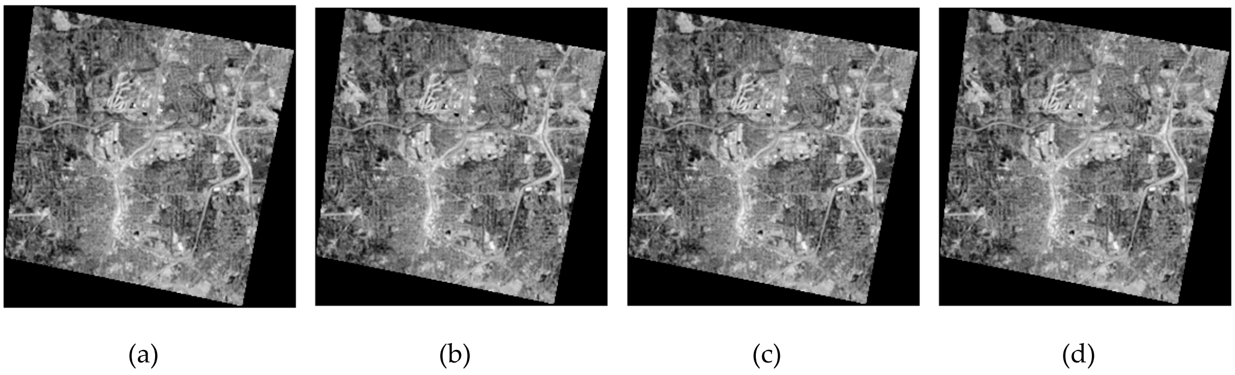


**Figure 23.** The georeferenced image of the second data set. (**a**) By FPGA. (**b**) By MATLAB. (**c**) By Visual Studio (C++). (**d**) By ENVI 5.3.

### 4.3.2. Gray Value Comparison

To verify the accuracy of gray value, the gray levels of georeferenced image are compared to those implemented using FPGA, MATLAB, Visual Studio (C++), and ENVI software. The georeferenced image implemented using FPGA as a referencing image, called "Ref-Img". The georeferenced image implemented using MATLAB, Visual Studio (C++), and ENVI are called "Img-MATLAB", "Img-C++", and "Img-ENVI", respectively. The gray differences between the Ref-Img and those georeferenced images implemented by MATLAB, Visual Studio (C++), and ENVI are obtained and shown in Figures 24 and 25.

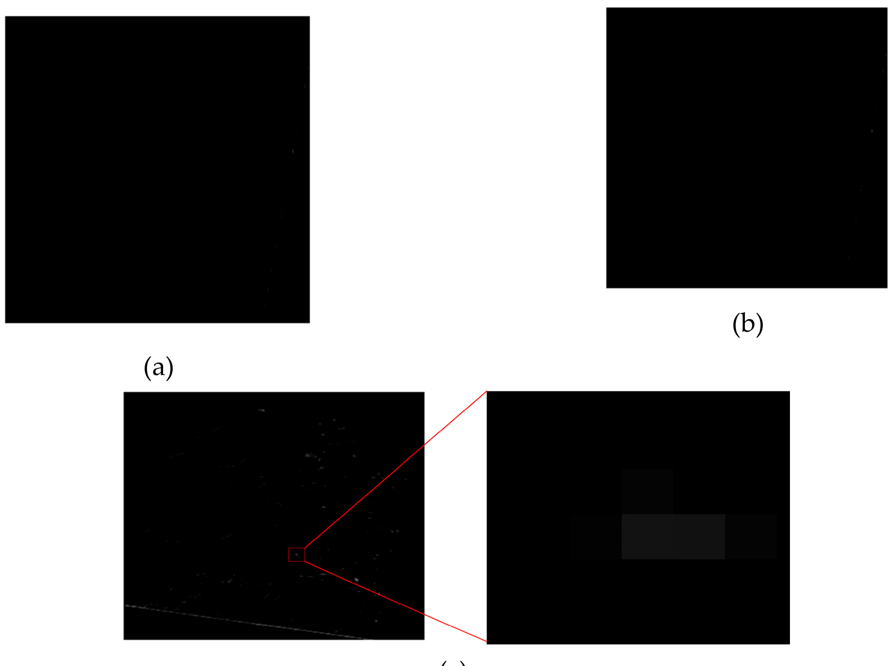

(a) (b) (c)

**Figure 24.** The differences between referencing image and other georeferenced images of the first image. (**a**) The differences between Ref-Img and Img-C++. (**b**) The differences between Ref-Img and Img-MATLAB. (**c**) The differences between Ref-Img and Img-ENVI.

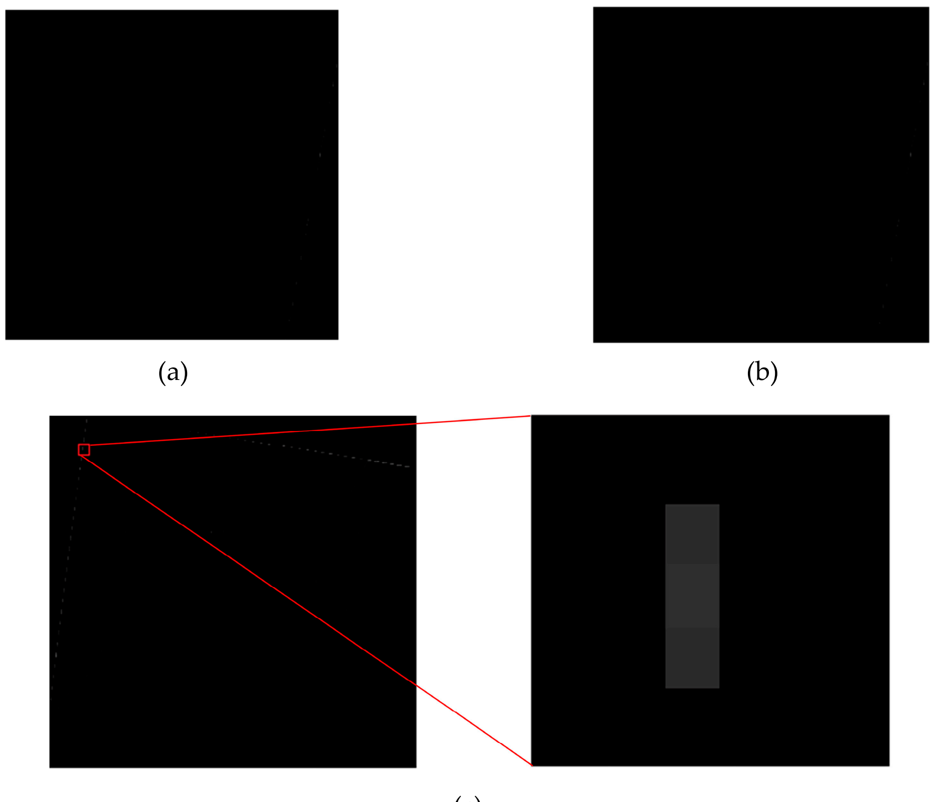

(a) (b) (c)

**Figure 25.** The differences between referenced image and other georeferenced images of the second image. (**a**) The differences between Ref-Img and Img-C++. (**b**) The differences between Ref-Img and Img-MATLAB. (**c**) The differences between Ref-Img and Img-ENVI.

As observed from Figure 24, the gray values of Figure 24a,b are 0, which means the proposed method implemented using FPGA has the same accuracy with the implemented by MATLAB, and Visual Studio (C++) [55,56]. Figure 24c indicates that the difference image between Ref-Img and Img-ENVI has many pixels are not same. The mean values of Figure 24a–c are 0, 0, and 0.713 respectively, which are less than 1 pixel.

As observe from Figure 25, the proposed method has the same accuracy as those implemented using MATLAB, and Visual Studio (C++), and has a small portion of the gray values are different from the Img-ENVI. The mean values of the Figure 25a–c are 0, 0, and 0.1265, respectively, which are all less than 1 pixel.

### 4.3.3. Resource Occupation Analysis

The resource utilization ration, including the flip-flop (FF), look-up-table (LUT) and DSP48s of the FPGA for the coordinate transformation method and bilinear interpolation function are assessed, respectively.

In the coordinate transformation method, 250,656 LUTs, 499,268 registers, and 388 DSP48s are utilized at rates of 40.96% (250656/612000 = 40.96%), 40.79%, and 37.96%, respectively (see Table 7).

In the bilinear interpolation function, floating-point and 32-bit fixed-point mixed operations are adopted, which can reduce the resource consumption of an FPGA. Table 8 lists the resources occupied for the bilinear interpolation scheme. 27,218 LUTs, 45,823 registers, 456 RAM/FIFO, and 267 DSP48s are utilized at rates of 4.45%, 3.74%, 30.40%, and 7.42%, respectively.

**Table 7.** The logic unit utilization ratio of the coordinate transformation method.

| Parameter | Used | Available | Utilization Ratio (%) |
|---|---|---|---|
| Number of slice LUTs | 250,656 | 612,000 | 40.96 |
| Number of slice Registers | 499,268 | 1,224,000 | 40.79 |
| Number of DSP48s | 388 | 3600 | 37.96 |

**Table 8.** The logic unit utilization ratio of the bilinear interpolation method.

| Parameter | Used | Available | Utilization Ratio (%) |
|---|---|---|---|
| Number of slice LUTs | 27,218 | 612,000 | 4.45 |
| Number of slice Registers | 45,823 | 1,224,000 | 3.74 |
| Number of block RAM/FIFO | 456 | 1500 | 30.40 |
| Number of DSP48s | 267 | 3600 | 7.42 |

### 4.3.4. Processing Speed Comparison

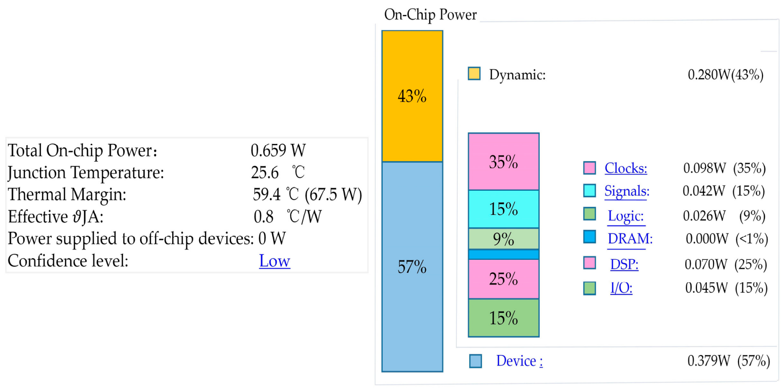
The processing speed is considered as one of the most important factors for implementation by an FPGA. Table 9 lists the processing speed implemented by FPGA, Visual Studio (C++) and MATLAB. The size of the first raw image is 256 $\times$ 256 pixels$^2$. After georeferencing, the size of georeferenced image is 281 $\times$ 281 pixels$^2$. The running time of the georeferencing method for the first image using FPGA, Visual Studio (C++) and MATLAB is 0.13s, 1.06s, and 1.12s, respectively. The size of the second raw image is 256 $\times$ 256 pixels$^2$; after georeferencing, the image size is 285 $\times$ 277 pixels$^2$. The running time of the georeferencing method for the second raw image using FPGA, Visual Studio (C++) and MATLAB is 0.15s, 1.21s, 1.26s, respectively. To put it simply, the processing speed using FPGA is 8 times faster than that based on PC computer.

**Table 9.** The consumption speed.

| Raw image | FPGA (second) f = 100 MHz | Visual Studio (C++) (second) | MATLAB (second) | Size (pixels$^2$) |
|---|---|---|---|---|
| bldt_tm.img | 0.13 | 1.06 | 1.12 | 281 × 281 |
| tmAtlanta.img | 0.15 | 1.21 | 1.26 | 285 × 277 |

### 4.3.5. Power Consumption

With the development of technology and the improvement of system performance, low power consumption has become one of the measurement objectives of on-board system. Resources, speed, and power consumption are three key factors in FPGA design. To obtain the power consumption, Vivado software provides a comprehensive methodologies and strategies for power consumption As observed from Figure 26, the powers of the dynamic and the device are 0.280 W (43%) and 0.379 W (57%), respectively. The total on-chip power is 0.659 W, which is acceptable in on-board processing platform.



| | |
|---|---|
| Total On-chip Power: | 0.659 W |
| Junction Temperature: | 25.6 ℃ |
| Thermal Margin: | 59.4 ℃ (67.5 W) |
| Effective ϑJA: | 0.8 ℃/W |
| Power supplied to off-chip devices: | 0 W |
| Confidence level: | Low |

On-Chip Power

| | | |
|---|---|---|
| 🟨 | Dynamic: | 0.280W(43%) |
| 🟪 | Clocks: | 0.098W (35%) |
| 🟦 | Signals: | 0.042W (15%) |
| 🟩 | Logic: | 0.026W (9%) |
| 🟦 | DRAM: | 0.000W (<1%) |
| 🟪 | DSP: | 0.070W (25%) |
| 🟩 | I/O: | 0.045W (15%) |
| 🟦 | Device : | 0.379W (57%) |

**Figure 26.** Power consumption

## 5. Conclusions

This paper presents a novel scheme for on-board georeferencing using FPGA optimized second-order polynomial equation and bilinear interpolation scheme, which consists of five modules: input data, coordinate transformation, bilinear interpolation and output data. The main contributions of this paper are as follows.

First, a comprehensive framework has been developed to optimize the georeferencing algorithm based on an FPGA. (1) A floating-point block LU decomposition is used to inverse the matrix. Compared with the traditional LU decomposition, the multiplication and division operations are reduced by 1.02 and 1.25 times, respectively. The block LU decomposition method can reduce the complexity for inverting matrix and speed up the operation. (2) To reduce resource consumption of an FPGA, some strategies are adopted in programming, i.e., 32-bit integer and floating-point mixed operation and serial-parallel data communication.

Second, the performances of the proposed algorithm are evaluated by error analysis, gray value comparison, resource occupation analysis, processing speed comparison, and power consumption. The *RMSE*s are less than one pixel, and other statistics, such as maximum, minimum, and mean error are less than one pixel. The gray values of the georeferenced image when implemented using the FPGA have the same accuracy as those implemented using MATLAB and Visual Studio (C++), and have a very close accuracy implemented using ENVI software. The processing speed using the proposed algorithm is 8 times faster than that based on PC computer. The on-chip power consumption is 0.659 W.

Therefore, it can be concluded that the proposed georeferencing algorithm implemented using FPGA with second-order polynomial model and bilinear interpolation algorithm can achieve real-time geographic referencing for remote sensing images.

**Author Contributions:** G.Z. conceives and designs the experiments; D.L. performs the experiments and writes the paper; J.H. offers the help of FPGA implementation; L.S. analyzes the data; R.Z. and X.Z. revise the manuscript. X.C. improves the English language and style.

**Funding:** This work was supported by grants from the Natural Science Foundation of China (grant No. 41431179, 41601365); GuangXi Key Laboratory for Spatial Information and Geomatics Program (Contract No.17-259-16-09); GuangXi innovative Development Grand under Grant 2018AA13005; GuangXi Key Research and development Program of China under Grant number 2016YFB502501; GuangXi innovative Development Grand Grant, (the grant number: GuiKe AA18118038). The State Oceanic Administration under Grant number [2014]#58; GuangXi Natural Science Foundation under grant number 2015GXNSFDA139032; and Guangxi Key Laboratory of Spatial Information and Geomatics Program under grant numbers 15-140-07-01 and 16-380-25-12.

**Acknowledgments:** The author would like to thank the reviewers for their constructive comments and suggestions.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Joyce, K.E.; Belliss, S.E.; Samsonov, S.V.; Mcneill, S.J.; Glassey, P.J. A review of the status of satellite remote sensing and image processing techniques for mapping natural hazards and disasters. *Prog. Phys. Geogr.* **2009**, *33*, 183–207. [CrossRef]
2. Tralli, D.M.; Blom, R.G.; Zlotnicki, V.; Donnella, A.; Evans, D.L. Satellite remote sensing of earthquake, volcano, flood, landslide and coastal inundation hazards. *J. Photogr. Remote Sens.* **2005**, *59*, 185–198. [CrossRef]
3. Sanyal, J.; Lu, X.X. Application of remote sensing in flood management with special reference to monsoon Asia: A review. *Nat. Hazards* **2004**, *33*, 283–301. [CrossRef]
4. Zhou, G. Near real-time orthorectificatoin and nosaic of small UAV-based video flow for time-critical event response. *IEEE Trans. Geosci. Remote Sens.* **2009**, *47*, 739–747. [CrossRef]
5. Zhou, G.; Zhang, R.; Liu, N.; Huang, J.; Zhou, X. On-board ortho-rectification for images based on an FPGA. *Remote Sens.* **2017**, *9*, 874. [CrossRef]
6. González, G.; Bernabé, S.; Mozos, D.; Plaza, A. FPGA Implementation of an algorithm for automatically detecting targets in remotely Sensed hyperspectral images. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2016**, *9*, 4334–4343. [CrossRef]
7. Qi, B.; Shi, H.; Zhuang, Y.; Chen, H.; Chen, L. On-board, real-time preprocessing system for optical remote-sensing imagery. *Sensors* **2018**, *18*, 1328. [CrossRef] [PubMed]
8. Dawood, A.S.; Visser, S.J.; Williams, J.A. Reconfigurable FPGAs for real time image processing in space. In Proceedings of the 2002 14th International Conference on Digital Signal Processing Proceedings, DSP 2002 (Cat. No. 02TH8628), Santorini, Greece, 1–3 July 2002; pp. 845–848.
9. Zhang, Y.; Kerle, N. Satellite remote sensing for near-real time data collection. In *Geospatial Information Technology for Emergency Response*; CRC Press: Taylor & Francis, London, UK, 2008; pp. 91–118.
10. Zhou, G.; Chen, W.; Kelmelis, J.A.; Zhang, D. A comprehensive study on urban true orthorectification. *IEEE Trans. Geosci. Remote Sens.* **2005**, *43*, 2138–2147. [CrossRef]
11. Zhou, G. Geo-Referencing of video flow from small low-cost civilian UAV. *IEEE Trans. Autom. Eng. Sci.* **2010**, *7*, 156–166. [CrossRef]
12. Ziboon, A.R.T.; Mohammed, I.H. Accuracy assessment of 2D and 3D geometric correction models for different topography in Iraq. *Eng. Technol. J. Part A Eng.* **2013**, *31*, 2076–2085.
13. Kartal, H.; Sertel, E.; Alganci, U. Comperative analysis of different geometric correction methods for very high resolution pleiades images. In Proceedings of the 2017 8th International Conference on Recent Advances in Space Technologies (RAST), Istanbul, Turkey, 19–22 June 2017; pp. 171–175.
14. Wang, T.; Zhang, G.; Li, D.; Tang, X.; Pan, H.; Zhu, X.; Chen, C. Geometric accuracy validation for ZY-3 satellite imagery. *IEEE Geosci. Remote Sens. Lett.* **2014**, *11*, 1168–1171. [CrossRef]

15. Chen, J.; Joang, T.; Lu, W.; Han, M. The geometric correction and accuracy assessment based on Cartosat-1 satellite image. In Proceedings of the 2010 3rd International Congress on Image and Signal Processing, Yantai, China, 16–18 October 2010; pp. 1253–1257.
16. Lee, C.A.; Gasster, S.D.; Plaza, A.; Chang, C.I.; Huang, B. Recent developments in high performance computing for remote sensing: A review. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2011**, *4*, 508–527. [CrossRef]
17. Plaza, A.; Valencia, D.; Plaza, J.; Martinez, P. Commodity cluster-based parallel processing of hyperspectral imagery. *J. Parallel Distrib. Comput.* **2006**, *66*, 345–358. [CrossRef]
18. Plaza, A.; Du, Q.; Chang, Y.L.; King, R.L. High performance computing for hyperspectral remote sensing. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2011**, *4*, 528–544. [CrossRef]
19. Fang, L.; Wang, M.; Li, D.; Pan, J. CPU/GPU near real-time preprocessing for ZY-3 satellite images: Relative radiometric correction, MTF compensation, and geocorrection. *ISPRS J. Photogr. Remote Sens.* **2014**, *87*, 229–240. [CrossRef]
20. Van der Jeught, S.; Buytaert, J.A.N.; Dirckx, J.J. Real-time geometric lens distortion correction using a graphics processing unit. *Opt. Eng.* **2012**, *51*. [CrossRef]
21. Thomas, O.; Trym, V.H.; Ingebrigt, W.; Ingebrigt, W. Real-time georeferencing for an airborne hyperspectral imaging system. *Algorithms Technol. Multispectral Hyperspectral Ultraspectral Imagery XVII* **2011**, *8048*. [CrossRef]
22. Reguera-Salgado, J.; Calvino-Cancela, M.; Martin-Herrero, J. GPU geocorrection for airborne pushbroom imagers. *IEEE Trans. Geosci. Remote Sens.* **2012**, *50*, 4409–4419. [CrossRef]
23. López-Fandiño, J.; Barrius, P.Q.; Heras, D.B.; Argüello, F. Efficient ELM-based techniques for the classification of hyperspectral remote sensing images on Commodity GPUs. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2015**, *8*, 2884–2893. [CrossRef]
24. Lu, J.; Zhang, B.; Gong, Z.; Li, E.; Liu, H. The remote-sensing image fusion based on GPU. In Proceedings of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Beijing, China, 23 October 2008; pp. 1233–1238.
25. Zhu, H.; Cao, Y.; Zhou, Z.; Gong, M. Parallel multi-temporal remote sensing image change detection on GPU. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IEEE Computer Society, Shanghai, China, 21–25 May 2012; pp. 1898–1904.
26. Ma, Y.; Chen, L.; Liu, P.; Lu, K. Parallel programing templates for remote sensing image processing on GPU architectures: Design and implementation. *Computing* **2016**, *98*, 7–33. [CrossRef]
27. Lopez, S.; Vladimirova, T.; Gonzalez, G.; Resano, J.; Mozos, D. The promise of reconfigurable romputing for hyperspectral imaging onboard systems: A review and trends. *Proc. IEEE* **2013**, *101*, 698–722. [CrossRef]
28. Zhou, G.; Baysal, O.; Kaye, J.; Habib, S.; Wang, C. Concept design of future intelligent earth observing satellites. *Int. J. Remote Sens.* **2004**, *25*, 2667–2685. [CrossRef]
29. Huang, J.; Zhou, G.; Zhang, D.; Zhang, G.; Zhang, R.; Baysal, O. An FPGA-based implementation of corner detection and matching with outlier rejection. *Int. J. Remote Sens.* **2018**, 1–20. [CrossRef]
30. Pakartipangi, W.; Darlis, D.; Syihabuddin, B.; Wijanto, H. Analysis of camera array on board data handling using FPGA for nano-satellite application. In Proceedings of the International Conference on Telecommunication Systems Services and Applications, Bandung, Indonesia, 25–26 November 2015; pp. 1–6.
31. Huang, J.; Zhou, G.; Zhou, X. A new FPGA architecture of fast and brief algorithm for on-board corner detection and matching. *Sensors* **2018**, *18*, 1014. [CrossRef] [PubMed]
32. Yu, G.; Vladimirova, T.; Sweeting, M.N. Image compression systems on board satellites. *Acta Astronautica* **2009**, *64*, 988–1005. [CrossRef]
33. Long, T.; Jiao, W.; He, G.; Zhang, Z. A fast and reliable matching method for automated georeferencing of remotely-sensed imagery. *Remote Sens.* **2016**, *8*, 56. [CrossRef]
34. Williams, J.A.; Dawood, A.S.; Visser, S.J. FPGA-based cloud detection for real-time onboard remote sensing. In Proceedings of the 2002 IEEE International Conference on Field-Programmable Technology (FPT), Hong Kong, China, 16–18 December 2002; pp. 110–116.
35. González, C.; Mozos, D.; Resano, J.; Plaza, A. FPGA implementation of the N-FINDR algorithm for remotely sensed hyperspectral image analysis. *IEEE Trans. Geosci. Remote Sens.* **2012**, *50*, 374–388. [CrossRef]
36. Swann, R.; Hawkins, D.; Westwellroper, A.; Johnstone, W. The potential for automated mapping from geocoded digital image data. *Photogr. Eng. Remote Sens.* **1988**, *54*, 187–193.

37. Savoy, F.M.; Dev, S.; Lee, Y.H.; Winkler, S. Geo-referencing and stereo calibration of ground-based Whole Sky Imagers using the sun trajectory. In Proceedings of the 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Beijing, China, 10–15 July 2016; pp. 7473–7476.
38. Jensen, J.R.; Lulla, K. Introductory Digital image processing-A remote sensing perspective. *Environ. Eng. Geosci.* **2007**, *13*, 89–90. [CrossRef]
39. Chen, L.C.; Teo, T.A.; Liu, C.L. The geometrical comparisons of RSM and RFM for FORMOSAT-2 satellite images. *Photogr. Eng. Remote Sens.* **2006**, *72*, 573–579. [CrossRef]
40. Bannari, A.; Morin, D.; Bénié, G.B.; Bonn, F.J. A theoretical review of different mathematical models of geometric corrections applied to remote sensing images. *Remote Sens. Rev.* **1995**, *13*, 27–47. [CrossRef]
41. Toutin, T. Geometric processing of remote sensing images: Models, algorithms and methods. *Int. J. Remote Sens.* **2004**, *25*, 1893–1924. [CrossRef]
42. Raffa, M.; Mercogliano, P.; Galdi, C. Georeferencing raster maps using vector data: A meteorological application. In Proceedings of the 2016 IEEE Metrology for Aerospace (MetroAeroSpace), Florence, Italy, 22–23 June 2016; pp. 102–107.
43. Zhou, G.; Ron, L. Accuracy evaluation of ground points from IKONOS high-resolution satellite imagery. *Photogr. Eng. Remote Sens.* **2000**, *66*, 1103–1112.
44. Zhou, G.; Yue, T.; Shi, Y.; Zhang, R.; Huang, J. Second-order polynomial equation-based block adjustment for orthorectification of DISP imagery. *Remote Sens.* **2016**, *8*, 680. [CrossRef]
45. Shlien, S. Geometric correction, registration, and resampling of Landsat imagery. *Can. J. Remote Sens.* **1979**, *5*, 74–89. [CrossRef]
46. Gribbon, K.T.; Bailey, D.G. A novel approach to real-time bilinear interpolation. In Proceedings of the DELTA 2004 Second IEEE International Workshop on Electronic Design, Test and Applications, Perth, WA, Australia, 28–30 January 2004; pp. 126–131.
47. Bailey, D.G. *Design for Embedded Image Processing on FPGAs*; John Wiley & Sons (Asia) Pte Ltd.: solaris south tower, Singapore, 2011; pp. 275–305. ISBN 978-0-470-82849-6.
48. Huang, J. FPGA-Based Optimization and Hardware Implementation of P-H Method for Satellite Relative Attitude and Absolute Attitude Solution. Ph.D. Thesis, Tianjin University, Tianjin, China, 2019.
49. Zhou, G.; Huang, J.; Shu, L. An FPGA-based P-H method on-board solution for satellite relative altitude. *Geomat. Inf. Sci. Wuhan University.* **2018**, *43*, 1–9. [CrossRef]
50. Daga, V.; Govindu, G.; Prasanna, V.; Gangadharapalli, S.; Sridhar, V. Efficient floating-point based block LU decomposition on FPGAs. In Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (Ersa'04), Las Vegas, NV, USA, 21–24 June 2004; pp. 276–279.
51. Gill, T.; Collett, L.; Armston, J.; Eustace, A.; Danaher, T.; Scarth, P.; Flood, N.; Phinn, S. Geometric correction and accuracy assessment of landsat-7 etm+ and landsat-5 tm imagery used for vegetation cover monitoring in queensland, Australia from 1988 to 2007. *Surveyor* **2010**, *55*, 273–287. [CrossRef]
52. Chen, J.; Ji, K.; Shi, Z.; Liu, W. Implementation of block algorithm for LU factorization. In Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering, Los Angeles, CA, USA, 31 March–2 April 2009; pp. 569–573.
53. Richards, J.A.; Richards, J.A. *Remote Sensing Digital Image Analysis*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 39–74. ISBN 978-3-642-30062-2.
54. Schowengerdt, R.A. Chapter 7-correction and calibration. In *Remote Sensing*, 3rd ed.; Academic Press: Cambridgem, MA, USA, 2007; p. 285-XXII. ISBN 978-0-12-369407-2.
55. French, J.C.; Balster, E.J.; Turri, W.F. A 64-bit orthorectification algorithm using fixed-point arithmetic. *High-Perform. Comput. Remote Sens.* **2013**, *8895*. [CrossRef]
56. Shaffer, D.A. An FPGA Implementation of Large-Scale Image Orthorectification. Ph.D. Thesis, University of Dayton, Dayton, OH, USA, 2018.