sensors

MDPI

Article

RPC-Based Orthorectification for Satellite Images Using FPGA

Rongting Zhang [1,2,3], Guoqing Zhou [1,2,3,4,*], Guangyun Zhang [3], Xiang Zhou [2,3,4] and Jingjin Huang [1,2,3]

1 School of Precision Instrument and Opto-Electronic Engineering, Tianjin University, Tianjin 300072, China; zrt65@tju.edu.cn (R.Z.); jingjin_huang@tju.edu.cn (J.H.)
2 Guangxi Key Laboratory for Spatial Information and Geomatics, Guilin University of Technology, Guilin 541004, China; zqx0711@tju.edu.cn
3 The Center for Remote Sensing, Tianjin University, Tianjin 300072, China; guangyunzhang1@gmail.com
4 School of Microelectronics, Tianjin University, Tianjin 300072, China
* Correspondence: gzhou@glut.edu.cn; Tel.: +86-773-589-6073

Received: 25 May 2018; Accepted: 28 July 2018; Published: 1 August 2018

Abstract: Conventional rational polynomial coefficients (RPC)-based orthorectification methods are unable to satisfy the demands of timely responses to terrorist attacks and disaster rescue. To accelerate the orthorectification processing speed, we propose an on-board orthorectification method, i.e., a field-programmable gate array (FPGA)-based fixed-point (FP)-RPC orthorectification method. The proposed RPC algorithm is first modified using fixed-point arithmetic. Then, the FP-RPC algorithm is implemented using an FPGA chip. The proposed method is divided into three main modules: a reading parameters module, a coordinate transformation module, and an interpolation module. Two datasets are applied to validate the processing speed and accuracy that are achievable. Compared to the RPC method implemented using Matlab on a personal computer, the throughputs from the proposed method and the Matlab-based RPC method are 675.67 Mpixels/s and 61,070.24 pixels/s, respectively. This means that the proposed method is approximately 11,000 times faster than the Matlab-based RPC method to process the same satellite images. Moreover, the root-mean-square errors (RMSEs) of the row coordinate (ΔI), column coordinate (ΔJ), and the distance ΔS are 0.35 pixels, 0.30 pixels, and 0.46 pixels, respectively, for the first study area; and, for the second study area, they are 0.27 pixels, 0.36 pixels, and 0.44 pixels, respectively, which satisfies the correction accuracy requirements in practice.

Keywords: orthorectification; field-programmable gate array (FPGA); rational polynomial coefficient (RPC)

1. Introduction

Orthorectification is a process that orthorectifies an image onto its upright planimetry map and removes the perspective angle [1–3]. Orthorectification is a prerequisite for remotely sensed (RS) image applications in areas such as land resource investigation, disaster monitoring, forestry inventory, and environmental changes analysis. The RS image that is orthorectified not only contains the geometric accuracy of the map but also has the features of the remote sensing image. In the past 20 years, many orthorectification methods were proposed. For example, Zhou et al. [2] presented a comprehensive study on theories, algorithms, and methods of large-scale urban orthoimage generation. Zhou [3] proposed a near real-time orthorectification method for mosaic of video flow acquired by an unmanned aerial vehicle (UAV). Aguilar et al. [4] used rigorous model and rational function model to orthorectify GeoEye-1 and WorldView-2 images and assessed the

geometric accuracy of the orthophoto. The results showed that the best horizontal geo-positioning accuracies were acquired by using third order rational functions with vendor's RPC coefficients data. Marsetič et al. [5] presented an automatic processing chain for orthorectification of optical pushbroom sensors. Habib et al. [6] proposed an approach using generated orthophotos from frame camera to improve the orthorectification of hyperspectral pushbroom scanner imagery. However, these studies on orthorectification were based almost entirely on ground-image processing systems, which is unable to meet the demand with respect to time-critical disasters. Thus, it is important to determine how to improve the speed of the orthorectification process when used in the on-board processing of a spacecraft.

With the increasing demands in (near) real-time RS imagery applications for applications such as military deployments, quick response to terrorist attacks, and disaster rescue (e.g., flooding monitoring), the on-board implementation of orthorectification has attracted much research worldwide in recent years. To increase the speed of image processing, researchers have proposed multiple parallel-processing methods and employed hardware acceleration such as the approach by Warpenburg and Siegel [7], who performed resampling in a single instruction stream-multiple data stream environment. Wittenbrink et al. [8] presented optimal concurrent-read-exclusive-write and exclusive-read-exclusive-write parallel-random-access-machine algorithms for spatial image warping. Liu et al. [9] proposed a parallel algorithm that is focused on massive remotely sensed orthorectification. Dai and Yang [10] proposed a fast graphic processing unit (GPU)–central processing unit (CPU) cooperative processing algorithm that is based on computing unified device architecture for the orthorectification of RS images. Reguera-Salgado et al. [11] proposed a method for the real-time geocorrection of images from airborne pushbroom sensors using the hardware acceleration and parallel-computing characteristics of modern GPUs. Quan et al. [12] presented an optical aerial image orthorectification parallel algorithm that employs GPU acceleration. These ground-based parallel-processing systems have increased to an extent the processing speed for RS image orthorectification. However, the RS images still need to be sent back to the ground-based processing centers. However, this process is time consuming. In addition, most parallel-processing methods are based on the multiple task operating system of the GPU, which cannot essentially solve the problem of a serial instruction method.

To realize on-board orthorectification in (near) real-time, an efficient approach is to apply field-programmable gate array (FPGA) hardware architecture because FPGA chips offer a highly flexible design, scalable circuits, and a high efficiency in data processing for its pipeline structure and fine-grained parallelism. In recent decades, researchers have widely used FPGA for image processing applications. Examples are Halle can coworkers' [13] proposed on-board image data processing system based on the neural network processor NI100, digital signal processors, and FPGA. Eadie et al. [14] investigated the use of FPGA for the correction of geometric image distortion. Kumar et al. [15] realized the real-time correction of images using an FPGA under a dynamic environment. Escamilla-Hernández et al. [16] and Kate [17] used an FPGA to implement data compression. Tomasi et al. [18] proposed a stereo vision algorithm using an FPGA to perform the correction of video graphics array images (57 fps). Pal et al. [19], Wang et al. [20], and Zhang et al. [21] applied FPGAs to accelerate the image data and signal filtering processes. Ontiveros-Robles et al. [22,23] proposed FPGA-based hardware architectures for real-time edge detection using fuzzy logic algorithm. Li et al. [24,25] utilized FPGAs to realize the real-time processing of video images to remove snow and fog. Huang et al. [26] proposed an FPGA-based method for the on-board detection and matching of the feature points. Huang et al. [27] presented a new FPGA architecture of a fast and brief algorithm for on-board corner detection and matching.

To the best of our understanding, research into FPGA hardware systems has focused mainly on the real-time correction of video images, noise removal, edge detection, etc., and there are few studies related to on-board orthorectification. Zhou et al. [28] first presented the concept of "on-board geometric correction", but details pertaining to its on-board implementation were not given. Zhou [3]

proposed a method for a real-time mosaic of video flow acquired by a small low-cost unmanned aerial vehicle. However, the method was implemented based on software, a serial instruction system, which would affect the real-time processing efficiency. Thus, this paper proposes a FPGA-based method for the on-board implementation of orthorectification. The proposed method can be divided into three modules: reading parameters module, coordinates transformation module, and interpolation module.

The major contribution of this study is a FPGA-based method, in which a traditional orthorectification algorithm is modified for on-board image (near) real-time orthorectification.

The paper is organized as follows. Section 2 describes the proposed RPC algorithm, i.e., fixed-point-based RPC (FP-RPC) algorithm, and gives the FPGA implementation process of the FP-RPC algorithm. Section 3 provides an experimental comparison of the proposed method using IKONOS-2 data and SPOT-6 data. Section 4 discusses the rectification accuracy by FPGA and PC platforms, and processing speed and resource consumption of these two platforms. Finally, Section 5 gives some conclusions.

2. RPC-Based Orthorectification Using an FPGA Chip

High-resolution satellite sensors are different from conventional aerial frame perspective imaging, and generally apply linear-array CCD pushbroom imaging technology. To deal with various types of images, many geometric processing models and algorithms are presented. One of the most widely used models is the rational polynomial coefficient (RPC) model, which is a general imaging model that is independent of the satellite sensor and platform. Many modern satellite images are equipped with rational polynomial coefficients (RPCs). Unlike rigorous physical models that are based on the collinear equation, which uses the ephemeris, attitude information, etc., to establish the acquisition geometry of the sensors, the RPC model does not require knowledge of the interior orientation elements and exterior orientation elements, which are sometimes not provided by vendors. The RPC model can produce uniform accuracy with a rigorous physical model, and is a simple generalized model. The RPC model has been widely applied to orthorectify satellite images with the increasing utilization of high-resolution images, as in [29–35]. The details of RPC orthorectification are given in [29,30].

In this study, the RPC algorithm is implemented using FPGA. Usually, an FPGA chip can offer a highly flexible design, scalable circuits, and a high efficiency in data processing for its pipeline structure and fine-grained parallelism. Moreover, an FPGA chip has advantages in size, weight, and power (SWaP) compared to GPU and CPU, which is helpful to integrate the FPGA into the on-board system.

However, the traditional RPC algorithm for orthorectification is computationally costly because of floating-point operations in the RPC algorithm. To implement on-board orthorectification using an FPGA chip in (near) real-time, a fixed-point-based RPC (FP-RPC) algorithm is proposed that can reduce the computation cost significantly. The details of the proposed method are given below.

2.1. Proposed RPC Algorithm

FP processing is a method that accelerates the calculation [36,37]. To make the transformation between a fixed-point variable and a floating-point variable, multiplication by a constant is necessary to maintain the precision. When the constant is set to a power of 2, the multiplication can be seen as a single bit shift, i.e.,

$$F = \lfloor 2^{\tau} F' \rfloor \quad (1)$$

where F' is a floating-point variable, F is a fixed-point variable, and τ is a scale factor, which affects the binary accuracy of the resulting integer representation. A larger scale factor will produce a higher degree of the binary accuracy.

In the proposed FP-RPC algorithm, all of the variables and constants are transformed to integers using Equation (1). Table 1 gives the integer variables and their scale factors. In Table 1, a'_i, b'_i, c'_i, and d'_i (i = 1 to 20) are multinomial coefficients. Generally, the values of b'_1 and d'_1 are 1. Lon', Lat',

and Hei' are geodetic coordinates, which represent the longitude, latitude, and height, respectively. Lat'_{off}, Lat'_{scale}, Lon'_{off}, Lon'_{scale}, H'_{off}, H'_{scale}, $Line'_{off}$, $Line'_{scale}$, $Samp'_{off}$, and $Samp'_{scale}$ are the parameters for normalization. $Samp'$ and $Line'$ represent the image coordinates, sample and line.

Table 1. Scale factors and integer variables.

Variable Name	Scale Factor	Integer Variable Name
a'_i, b'_i, c'_i, and d'_i (i = 1 to 20)	τ_1	a_i, b_i, c_i, and d_i (i = 1 to 20)
Lon', Lat', Hei'	τ_2	Lon, Lat, Hei
Lat'_{off}, Lat'_{scale}, Lon'_{off}, Lon'_{scale}, H'_{off}, H'_{scale}	τ_3	Lat_{off}, Lat_{scale}, Lon_{off}, Lon_{scale}, H_{off}, H_{scale}
$Line'_{off}$, $Line'_{scale}$, $Samp'_{off}$, $Samp'_{scale}$	τ_3	$Line_{off}$, $Line_{scale}$, $Samp_{off}$, $Samp_{scale}$

According to Fraser et al. [29] and Grodecki et al. [30] and Equation (1), the normalized coordinates are converted into integers by

$$\begin{aligned} L &= 2^{\tau_2}\frac{2^{-\tau_2}Lon - 2^{-\tau_3}Lon_{off}}{2^{-\tau_3}Lon_{scale}} \\ P &= 2^{\tau_2}\frac{2^{-\tau_2}Lat - 2^{-\tau_3}Lat_{off}}{2^{-\tau_3}Lat_{scale}} \\ H &= 2^{\tau_2}\frac{2^{-\tau_2}Hei - 2^{-\tau_3}H_{off}}{2^{-\tau_3}H_{scale}} \end{aligned} \tag{2}$$

$$\begin{aligned} X &= 2^{\tau_2}\frac{2^{-\tau_2}Samp - 2^{-\tau_3}Samp_{off}}{2^{-\tau_3}Samp_{scale}} \\ Y &= 2^{\tau_2}\frac{2^{-\tau_2}Line - 2^{-\tau_3}Line_{off}}{2^{-\tau_3}Line_{scale}} \end{aligned} \tag{3}$$

Moreover, the polynomials are converted into integers by

$$2^{-\tau_2}\mathbf{ND}_{\mathrm{LS}} = (2^{-\tau_1}\mathbf{C})(2^{-\tau_2}\mathbf{N}^{\mathrm{T}}) \tag{4}$$

$$\mathbf{ND}_{\mathrm{LS}} = 2^{-\tau_1}\mathbf{C}\mathbf{N}^{\mathrm{T}} \tag{5}$$

where

$$\mathbf{ND}_{\mathrm{LS}} = \begin{bmatrix} Num_L & Den_L & Num_S & Den_S \end{bmatrix}^{\mathrm{T}},$$

$$\mathbf{C} = \begin{bmatrix} a_1 & a_2 & \dots & a_{19} & a_{20} \\ b_1 & b_2 & \dots & b_{19} & b_{20} \\ c_1 & c_2 & \dots & c_{19} & c_{20} \\ d_1 & d_2 & \dots & d_{19} & d_{20} \end{bmatrix},$$

$$\mathbf{N} = \begin{bmatrix} 1 & L & P & H & LP & LH & PH & LL & PP & HH & PLH & LLL & LPP & LHH & LLP & PPP & PHH & LLH & PPH & HHH \end{bmatrix}.$$

In addition, the normalized image coordinates (X', Y') are converted into integers by

$$2^{-\tau_2}Y = \frac{2^{-\tau_2}Num_L}{2^{-\tau_2}Den_L},\ 2^{-\tau_2}X = \frac{2^{-\tau_2}Num_S}{2^{-\tau_2}Den'_S} \tag{6}$$

$$Y = 2^{\tau_2}\frac{Num_L}{Den_L},\ X = 2^{\tau_2}\frac{Num_S}{Den_S} \tag{7}$$

Finally, the image coordinates ($Samp'$, $Line'$) are converted into integers by

$$\begin{cases} 2^{-\tau_3}Samp = (2^{-\tau_2}X)(2^{-\tau_3}Samp_{scale}) + 2^{-\tau_3}Samp_{off} \\ 2^{-\tau_3}Line = (2^{-\tau_2}Y)(2^{-\tau_3}Line_{scale}) + 2^{-\tau_3}Line_{off} \end{cases} \tag{8}$$

$$\begin{cases} Samp = (2^{-\tau_2}X)Samp_{scale} + Samp_{off} \\ Line = (2^{-\tau_2}Y)Line_{scale} + Line_{off} \end{cases} \tag{9}$$

2.2. Parallel Computation of Orthorectification Using an FPGA

Many factors affect the computation speed when an FPGA is adopted, such as the optimal design of algorithms and the logical resource of the utilized FPGA. By analyzing the structure of the FP-RPC algorithm and optimizing it, an FPGA-based hardware architecture for FP-RPC-based orthorectification is designed, as shown in Figure 1. As described in Equations (2)–(9), their structures are similar. It is convenient for FPGAs to be implemented in parallel. As shown in Figure 1, the FPGA-based FP-RPC module can be divided into three submodules, that is, Read_parameter_mod (RPM), which is used to send parameters to other modules; Coordinate_Transform_mod (CTM), which is applied to transform geodetic coordinates to image coordinates; and Interpolation_mod (IM), which is utilized to perform bilinear interpolation. The details of these modules are given as follows.

1. For RPM, the coefficients of RPC can be calculated by least-squares adjustment [38]. According to Tao et al. [38], the computing processes of the RPC coefficients are as follows. Equation (7) can be rewritten as

$$\mathrm{F}_X = Num_S(P,L,H) - 2^{\tau_2} XDen_S(P,L,H) = 0 \tag{10}$$

$$\mathrm{F}_Y = Num_L(P,L,H) - 2^{\tau_2} YDen_L(P,L,H) = 0 \tag{11}$$

Thus, the matrix form of error equation can be expressed as

$$\mathbf{V} = \mathbf{MA} - \mathbf{R} \tag{12}$$

where

$$\mathbf{M} = \begin{bmatrix} \frac{\partial F_X}{\partial a_i} & \frac{\partial F_X}{\partial b_j} & \frac{\partial F_X}{\partial c_i} & \frac{\partial F_X}{\partial d_j} \\ \frac{\partial F_Y}{\partial a_i} & \frac{\partial F_Y}{\partial b_j} & \frac{\partial F_Y}{\partial c_i} & \frac{\partial F_Y}{\partial d_j} \end{bmatrix} (i = 1,\ 2,\ \ldots,\ 20;\ j = 2,\ \ldots,\ 20)$$

$$\mathbf{R} = \begin{bmatrix} -\mathrm{F}_X^0 & -\mathrm{F}_Y^0 \end{bmatrix}^{\mathrm{T}}$$

$$\mathbf{A} = \begin{bmatrix} a_i & b_j & c_i & d_j \end{bmatrix}^{\mathrm{T}}$$

Equation (12) is solved by least-squares algorithm, and the solutions of RPC coefficients can be represented as

$$\mathbf{A} = (\mathbf{M}^{\mathrm{T}}\mathbf{M})^{-1}\mathbf{M}^{\mathrm{T}}\mathbf{R} \tag{13}$$

The solution for Equation (13) is acquired by an iterative process. The entire algorithm of Equation (13) has been implemented in our previous work [39], in which the detailed implementing process can be found. The normalization parameters can be calculated by the following equations.

$$\begin{gathered} Lat_{off} = \frac{\sum_{i=1}^{n} Lat_i}{n},\ Lon_{off} = \frac{\sum_{i=1}^{n} Lon_i}{n},\ H_{off} = \frac{\sum_{i=1}^{n} Hei_i}{n},\ Samp_{off} = \frac{\sum_{i=1}^{n} Samp_i}{n}, \\ Line_{off} = \frac{\sum_{i=1}^{n} Line_i}{n} \end{gathered} \tag{14}$$

$$\begin{aligned} Lat_{scale} &= \max\left(\left|Lat_{\max} - Lat_{off}\right|, \left|Lat_{\min} - Lat_{off}\right|\right), \\ Lon_{scale} &= \max\left(\left|Lon_{\max} - Lon_{off}\right|, \left|Lon_{\min} - Lon_{off}\right|\right) \end{aligned} \tag{15}$$

$$\begin{aligned} H_{scale} &= \max\left(\left|Hei_{\max} - H_{off}\right|, \left|Hei_{\min} - H_{off}\right|\right), \\ Line_{scale} &= \max\left(\left|Line_{\max} - Line_{off}\right|, \left|Line_{\min} - Line_{off}\right|\right) \end{aligned} \tag{16}$$

$$Samp_{scale} = \max\left(\left|Samp_{\max} - Samp_{off}\right|, \left|Samp_{\min} - Samp_{off}\right|\right) \tag{17}$$

where n is the number of ground control points (GCPs). When the enable signal is being received, the geodetic coordinates (*Lon*, *Lat*, *Hei*) stored in the RAMs are read, and sent to Coordinate_Transform_mod (CTM) with the attained parameters and the start signal (Start_Sig) in the same clock cycle.

2. When the Start_Sig, the constants, and the geodetic coordinates are being received in the CTM, the normalized coordinates (P, L, H) are first calculated in the regularization module (Regulation_mod, ReM). Then, the normalized coordinates and the done signal of ReM (ReM_Done_Sig) are sent to the polynomial module (Polynomial_mod, PM) with a_i, b_i, c_i, and d_i (i = 1 to 20) in the same clock cycle to compute the numerators and denominators (Num_L, Num_S, Den_L, and Den_S) of Equation (7). Subsequently, when the done signal (PM_Done_Sig) of PM, Num_L, Num_S, Den_L, and Den_S are being received, the normalized coordinates (X, Y) of the image coordinates are calculated. Finally, when the normalized coordinates (X, Y) and the done signal (RaM_Done_Sig) of the ratio module (Ratio_mod, RaM) are being received, the image coordinates (*Samp*, *Line*) and the done signal (RCM_Done_Sig) of the image coordinate calculation module (Row_Clm_mod, RCM) are acquired and sent to the interpolation module (Interpolation_mod, IM) in the same clock cycle.
3. When the image coordinates (*Samp*, *Line*) and RaM_Done_Sig are being received in IM, the gray of pixel (*Samp*, *Line*) is obtained by interpolating, and the done signal (IM_Done_Sig) of IM is produced.
4. When the posedge clk of the signal, ALL_Done_Sig is being detected, the processing is finished.

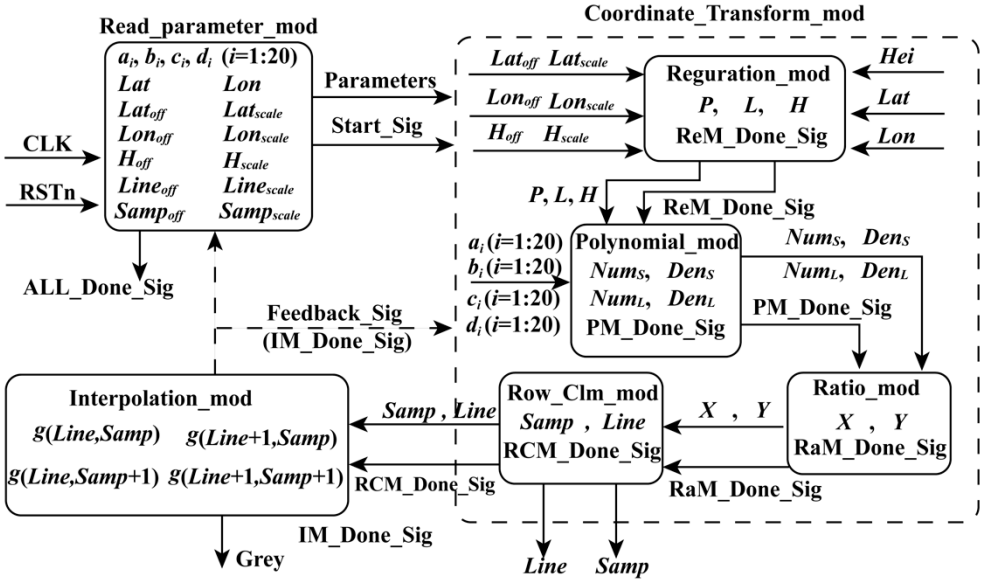


Figure 1. Flowchart showing implementation on an FPGA chip.

2.2.1. Read Parameter Module

To ensure that the constants, geodetic coordinates, and the start signal (Start_Sig) are sent in the same clock cycle, a parallel module (i.e., the RPM) is designed (see Figure 2). In the RPM, the constants are assigned corresponding values, while the geodetic coordinates are stored in RAM. In this design, all values are expressed using a fixed point of 32 bits to ensure computational accuracy.

In the RPM, the geodetic coordinates are sent to the next module according to the order of the column. First, the address of RAM is initialized as 0. When the enable signal is detected, the first group of geodetic coordinates (Lat_0, Lon_0, Hei_0) is read from the RAM and sent to the next module with the constants and the Start_Sig in the same clock cycle. Starting from the second group of geodetic

coordinates, the rules for reading and sending geodetic coordinates are changed. In other words, after the second group of geodetic coordinates (Lat_1, Lon_1, Hei_1), the geodetic coordinates will be read and sent unless the enable signal and the feedback signal (Feedback_Sig), which are sent by the interpolation module, are detected at the same time. After the final group of geodetic coordinates are read and sent, if the Feedback_Sig is received, the done signal (ALL_Done_Sig) of orthorectification is produced. When the ALL_Done_Sig is detected, the process of orthorectification is stopped.

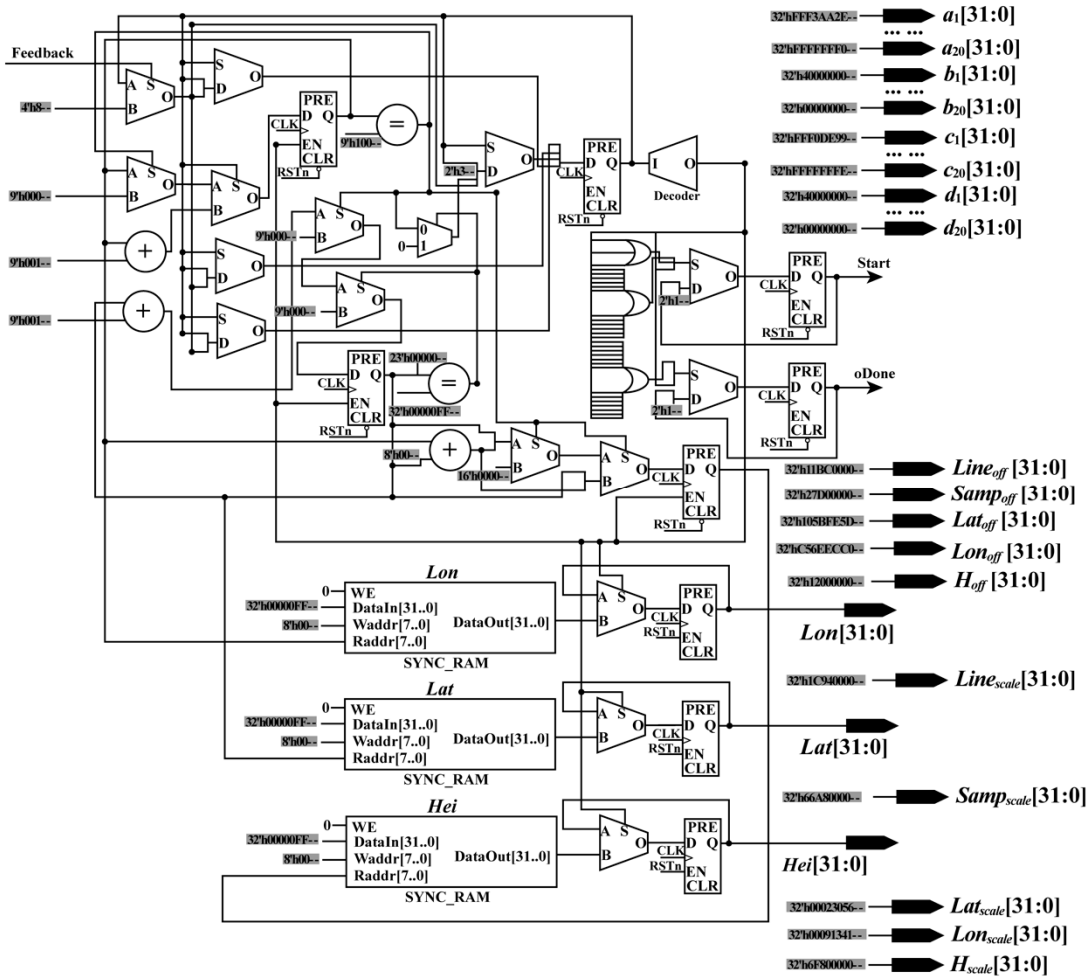


Figure 2. Schematic diagram of the read parameter module.

2.2.2. Coordinate Transformation Module

As shown in Figure 1, for the CTM, the inputs contain the constants, the geodetic coordinates, and the Start_Sig, while the outputs include image coordinates and the done signal of this module. The CTM can be divided into four submodules, namely ReM, PM, RaM, and RCM. Details regarding these four submodules are as follows.

- **Regulation Module**

According to Section 2.1, the geodetic coordinates (*Lat*, *Lon*, *Hei*) should be first transformed as the normalized coordinates (*L*, *P*, *H*) based on Equation (2) because this operation can minimize the introduction of errors during the computation of the numerical stability of equations [13]. As shown in Equation (2), the forms of these equations are uniform. In other words, they are suitable for implementation using FPGA. To obtain the normalized coordinates (*L*, *P*, *H*) of the geodetic coordinates (*Lat*, *Lon*, *Hei*) using an FPGA chip, a parallel computation architecture is presented in Figure 3.

In Figure 3, the structures of "Normalize *Lat*", "Normalize *Lon*", and "Normalize *Hei*" are similar. Thus, only the schematic diagram of "Normalize *Lat*" is presented (see Figure 4).

As shown in Figure 4, during the computation process, 1 divider, 10 adders, 10 flipflops, and 16 multiplexer units are mainly used to normalize the *Lat*. In this design, the relationship among "Normalize *Lat*", "Normalize *Lon*", and "Normalize *Hei*" is parallel. The normalized coordinates (*L*, *P*, *H*) are obtained in the same clock cycle as the done signal.

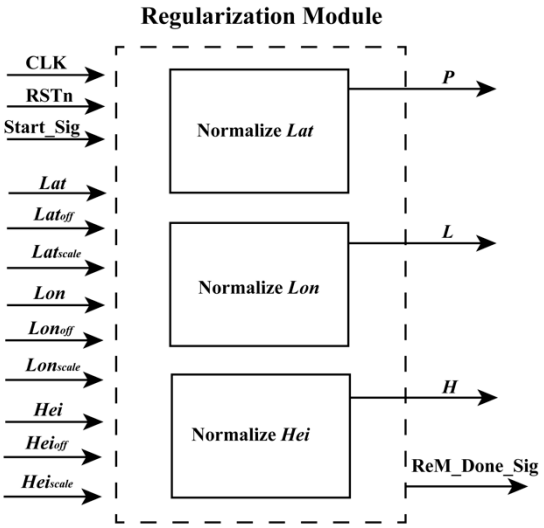


Figure 3. Schematic diagram of the ReM.

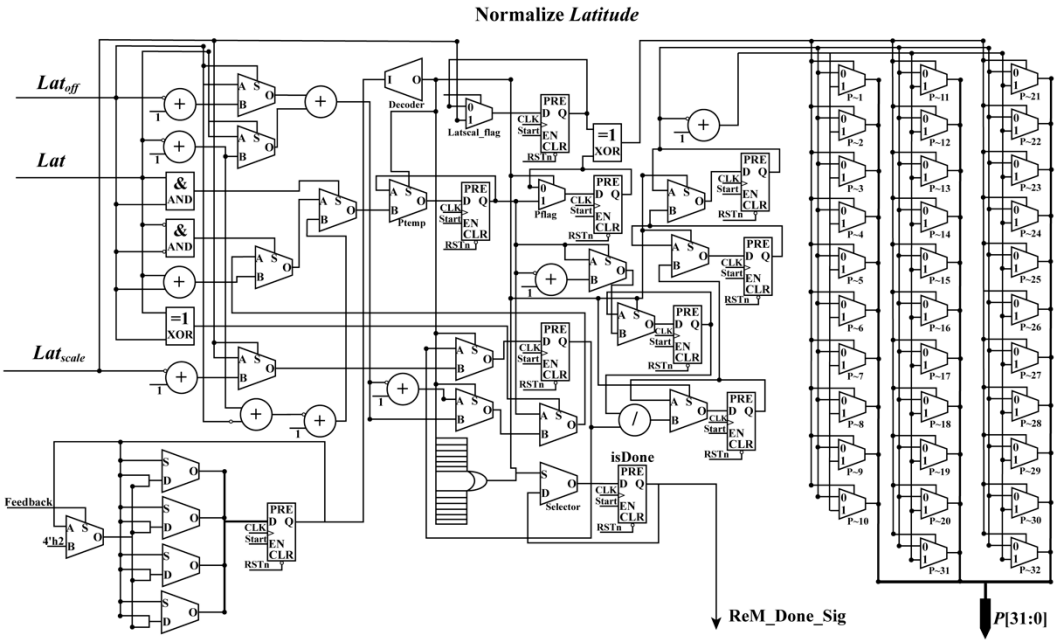


Figure 4. Schematic diagram of normalizing *Latitude*.

- **Polynomial Module**

When the ReM_Done_Sig and (*L*, *P*, *H*) are being received by the PM module, the PM module starts to work. As shown in Equations (4) and (5), these polynomials have a uniform form, which are suitable for the implementation of an FPGA chip in parallel. In these equations, variables such as *LH*,

LP, and *PH* are shared. To implement these polynomials in parallel using an FPGA chip, a parallel computation architecture is proposed in Figures 5 and 6. As shown in those figures, the PM module is divided into two parts: one is used to perform multiplication and the other is applied to manipulate addition. When performing addition, some special operations about the positive and negative sets of data should be considered. Thus, for the additions in Figure 6, each of them is extended to a similar form, as shown in Figure 7, taking the addition between a_3P and a_4H as an example. In the example, three situations are considered: (i) a_3P and a_4H are both positive; (ii) a_3P and a_4H are both negative; and (iii) a_3P and a_4H have opposite signs. The details for an extended addition are shown in Figure 7.

To implement each polynomial, 35 multipliers are utilized in the multiplication, and 19 extended additions are used. In each extended addition, three flipflops, four selectors, seven adders, and eleven multiplexers are applied. After processing the PM module, four sums, i.e., Num_L, Num_S, Den_L, and Den_S, are obtained with the done signal of the PM module, PM_Done_Sig, in the same clock cycle.

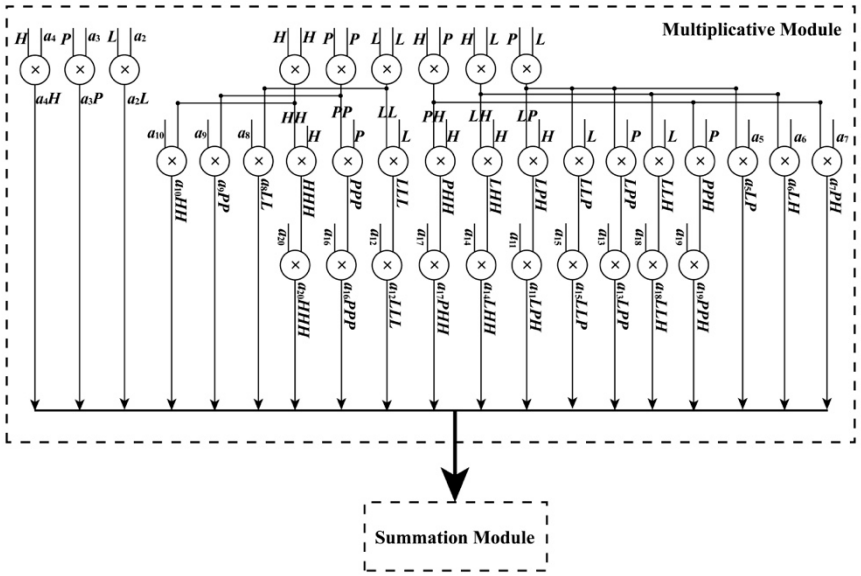


Figure 5. Schematic diagram of polynomial module.

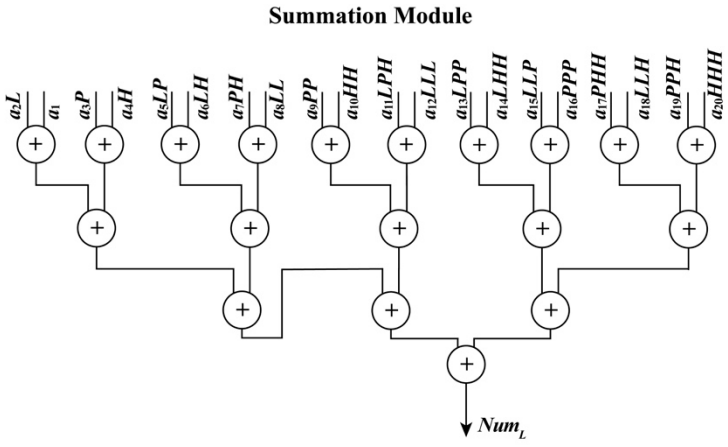


Figure 6. Schematic diagram of summation module.

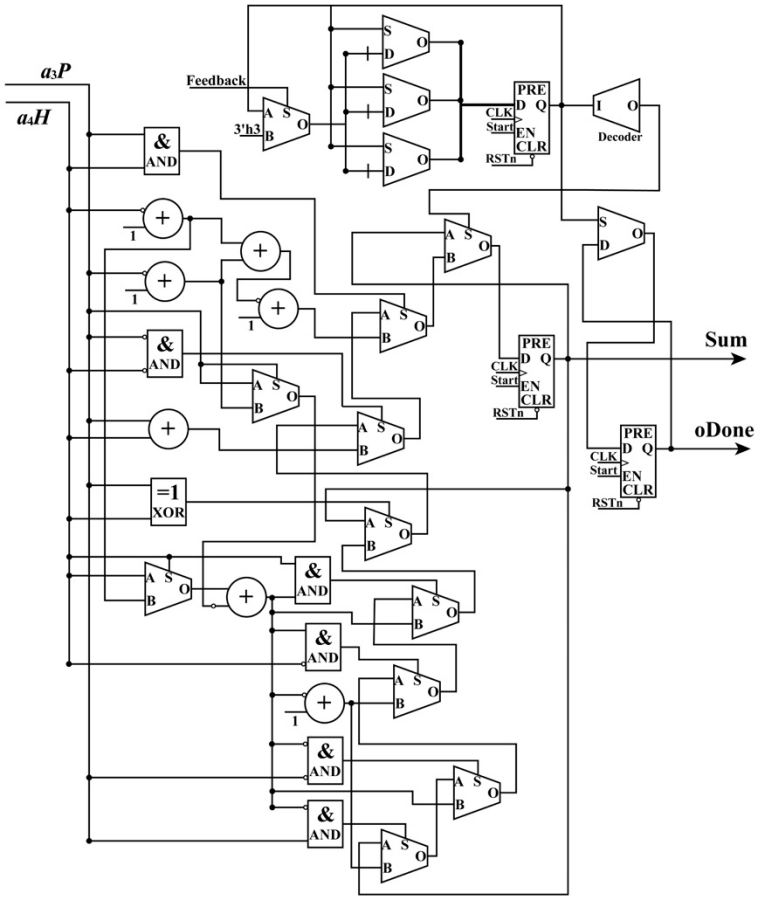


Figure 7. Details showing an example of extended addition.

- **Ratio Module**

When the PM_Done_Sig, Num_L, Num_S, Den_L and Den_S are being received, the RaM module starts to calculate the normalized coordinates (X, Y) of image coordinates. As shown in Equation (6), the forms for the two equations are the same. It is convenient to calculate X and Y in parallel using an FPGA chip. In Figure 8, a parallel-computing architecture that is used to calculate X is presented. In the same way, the Y coordinate can be obtained.

To obtain the X (or Y) coordinate, one divider, three adders, six multiplexers, six flipflops (two flipflops are public), and 32 selectors are applied. After the processing of the RaM module, the X coordinate and Y coordinate are acquired with the done signal, RaM_Done_Sig, in the same clock cycle.

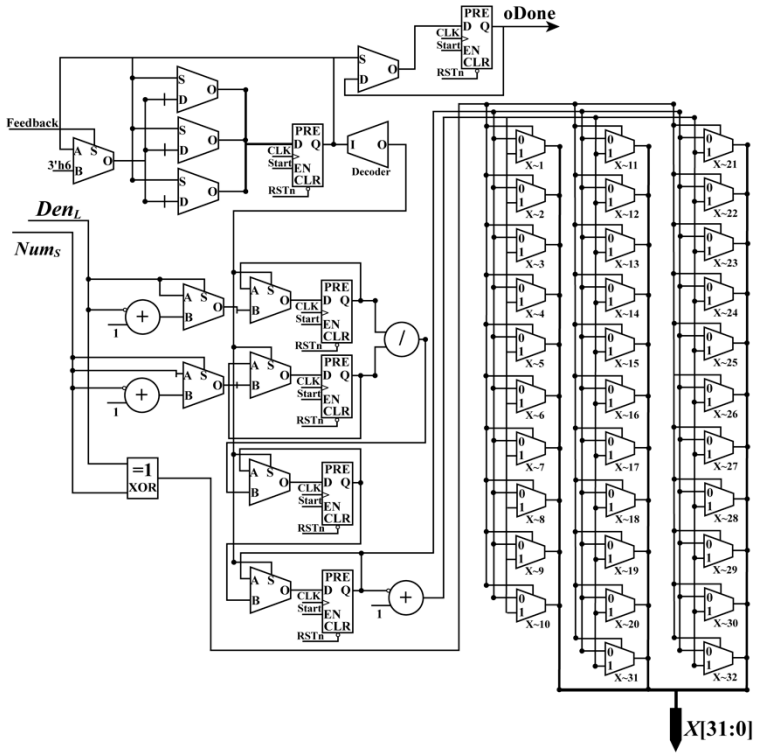


Figure 8. Schematic diagram of the ratio module.

- **Image Coordinate Calculation Module**

When the RaM_Done_Sig, X, and Y coordinates are being detected, the RCM module starts to calculate the image coordinates (*Samp*, *Line*), i.e., column and row indexes. As shown in Equation (9), the equations give the relationship between the normalized coordinates (X, Y) and image coordinates (*Samp*, *Line*).

As shown in Equation (9), the equations have a uniform form, which is helpful for implementation using an FPGA. To calculate the image coordinates (*Samp*, *Line*) in parallel, a parallel-computing hardware architecture is designed. Because the forms of the equations in Equation (9) are similar, only the schematic diagram used for calculating the *Samp* coordinate is given. As shown in Figure 9, there are one multiplier, four flipflops (two of them are shared when calculating *Line* coordinate), five selectors (MUX) shared when calculating *Line* coordinate, seven adders, and 136 multiplexers (MUX21).

After the processing of the RCM module, the image coordinates, that is, the column and row indexes (*Samp*, *Line*), and the done signal (RCM_Done_Sig) are obtained in the same clock cycle. Up to this point, the whole processing of the coordinate transformation is done. The obtained image coordinates are sent to the interpolation module to interpolate the grayscale.

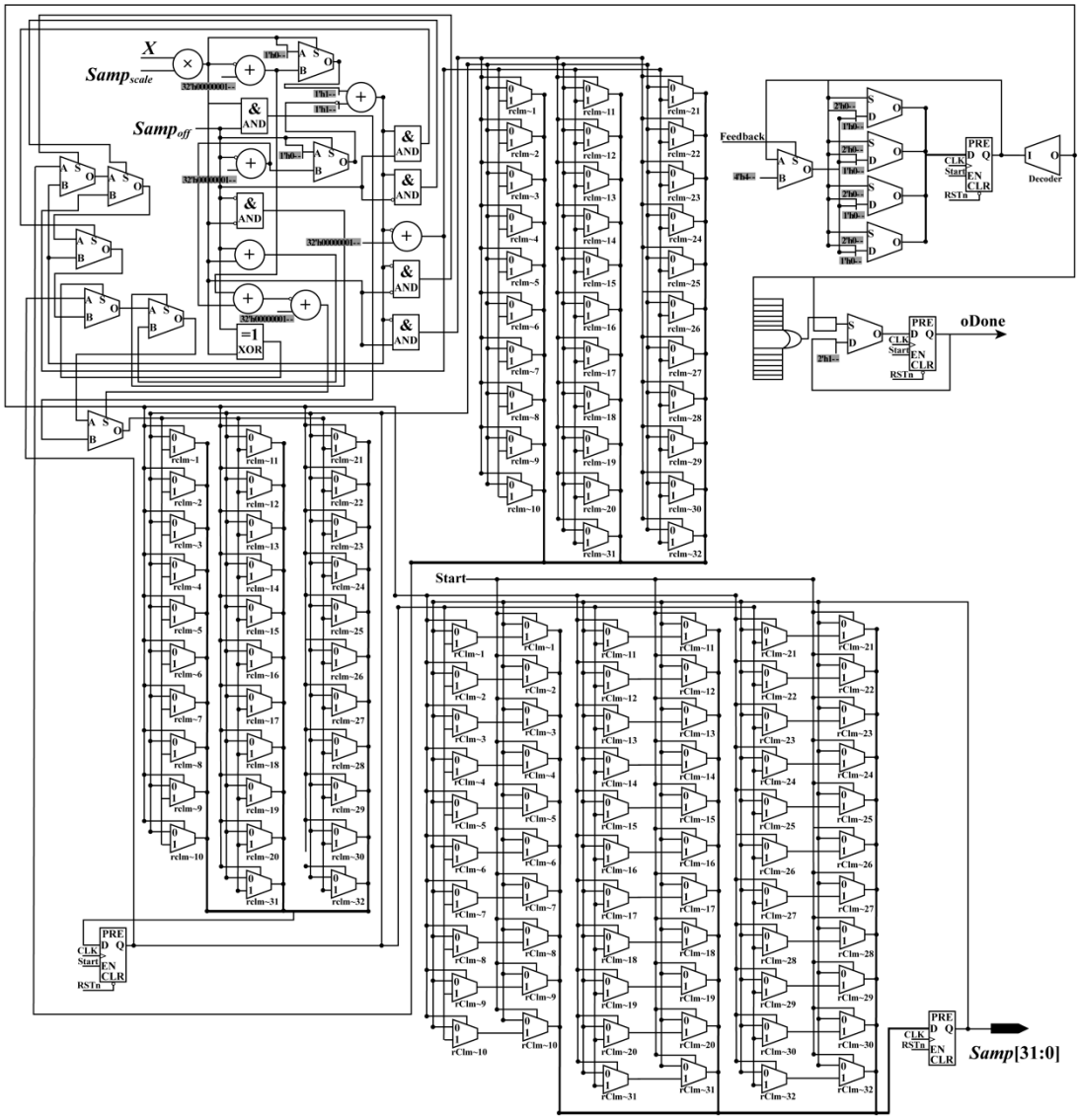


Figure 9. Schematic diagram used for calculating *Samp* coordinate.

2.2.3. Interpolation Module

Because the obtained column and row indexes may not exist only at the center of a pixel, it is necessary to use the interpolation method to obtain the grayscale in the obtained column and row indexes. Considering the interpolation effect, the complexity of an interpolation algorithm, and the resources of an FPGA, the bilinear interpolation method is selected to implement the interpolation for grayscale. Mathematically, the bilinear interpolation algorithm can be expressed by the following equation:

$$\begin{aligned} g(i+p,\, j+q) = \quad & (1-p)(1-q)g(i,\, j) + (1-p)qg(i,\, j+1) \\ & + p(1-q)g(i+1,\, j) + pqg(i+1,\, j+1) \end{aligned} \tag{18}$$

where i and j are nonnegative integers; the intermediates $p = |i - \text{int}(i)|$ and $q = |j - \text{int}(j)|$ are within the range of (0, 1); and $g(i, j)$ represents gray values.

To implement the bilinear interpolation algorithm in parallel using an FPGA chip, a parallel computation architecture was designed (see Figure 10). The designed hardware architecture contains four submodules/parts: (i) the subtract_mod, which is used to obtain the integer part (*iLine* and *iSamp*)

and fractional part (p and q) of *Line* and *Samp* indexes, and to calculate the subtraction (1_p and 1_q) in Equation (18); (ii) the get_gray_addr_mod, which is applied to obtain the address of gray in RAM; (iii) the multiplication part, which is utilized to calculate the multiplications in Equation (18); and (iv) the calculate_sum_mod, which is used to compute the sum in Equation (18). After the processing of the calculate_sum_mod, the results of interpolation in (*Samp*, *Line*) are obtained. The details of subtract_mod, get_gray_addr_mod, and calculate_sum_mod are described as follows.

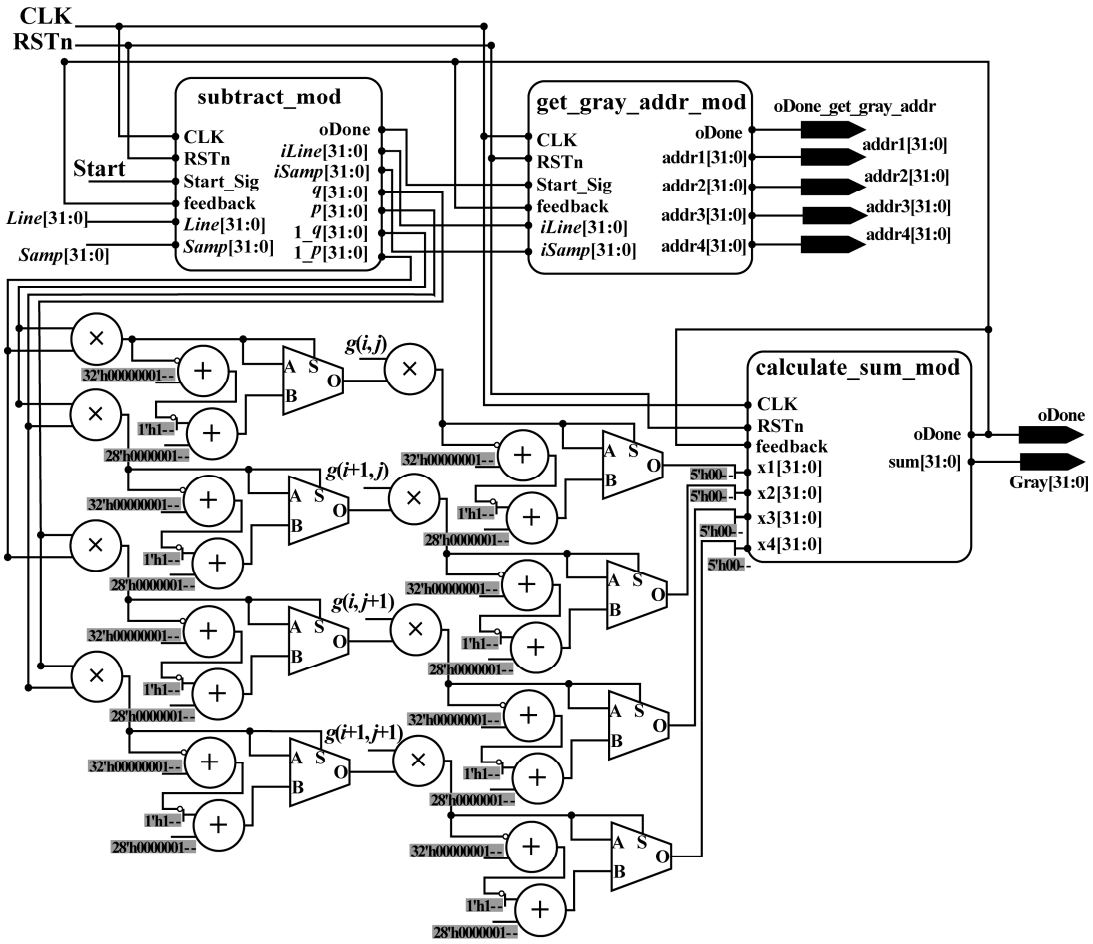


Figure 10. Schematic diagram of interpolation module.

- **subtract_mod**

As shown in Equation (18), to perform the bilinear interpolation method, the gray values of four neighbors around the acquired column and row indexes are required. Thus, the acquired column and row indexes should be pre-processed to obtained the integer part and fractional part, which are used to calculate $(1 - q)$ and $(1 - p)$. To implement the function using an FPGA chip, a parallel-computing architecture is proposed, named subtract_mod. In subtract_mod, the methods used to acquire *iSamp*, q and 1_q are similar to those for obtaining *iLine*, p and 1_p, respectively. Thus, in this section, only the schematic diagram for obtaining *iSamp*, q and 1_q is given (see Figure 11).

As shown in Figure 11, to obtain *iSamp*, q and 1_q, three adders, seven multiplexers (MUX21), and nine flipflops (three of which are shared when *iLine*), p and 1_p are used. In addition, three MUXs are public. After the processing of the whole subtract_mod, *iSamp*, q, 1_q, *iLine*, p, and 1_p are acquired with the done signal in the same clock cycle. When obtaining these variables, *iSamp* and *iLine* are sent to the next submodule to retrieve the address of gray for four neighbors in RAM. Meanwhile, q, 1_q, *iLine*, p, and 1_p are sent to another part to perform multiplication.

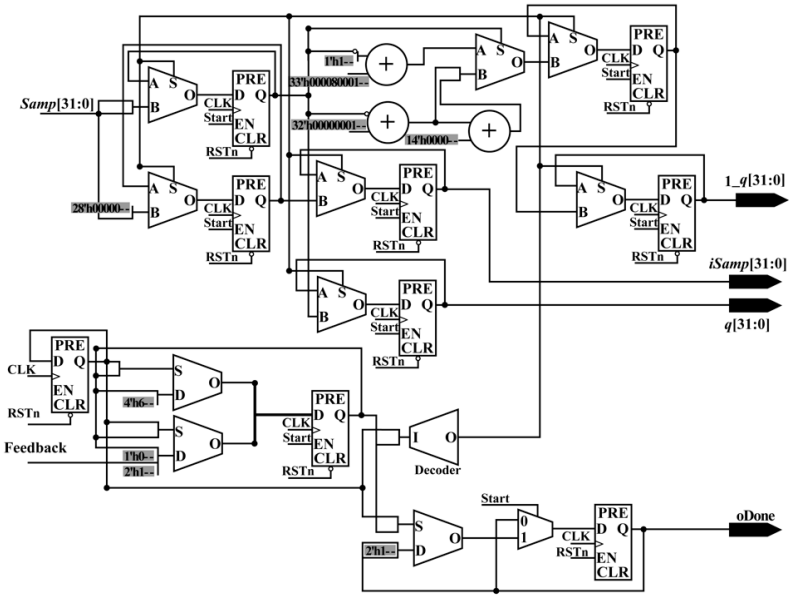


Figure 11. Schematic diagram of the subtract_mod module.

- **get_gray_addr_mod**

The grayscale of a pixel can be obtained according to the corresponding address. To obtain the gray values of four neighbors around the obtained column and row indexes in parallel, a parallel-computing hardware architecture is proposed (see Figure 12), called get_gray_addr_mod. In the get_gray_addr_mod, 3 LESS-THAN comparators, 9 adders, 10 MUX21, 12 flipflops, and 70 MUX are applied. After the processing of the get_gray_addr_mod, four addresses are obtained with the done signal in the same clock cycle. According to the obtained addresses, the gray values can be acquired from RAM. Then, they are sent to the multiplication part to perform the multiplication.

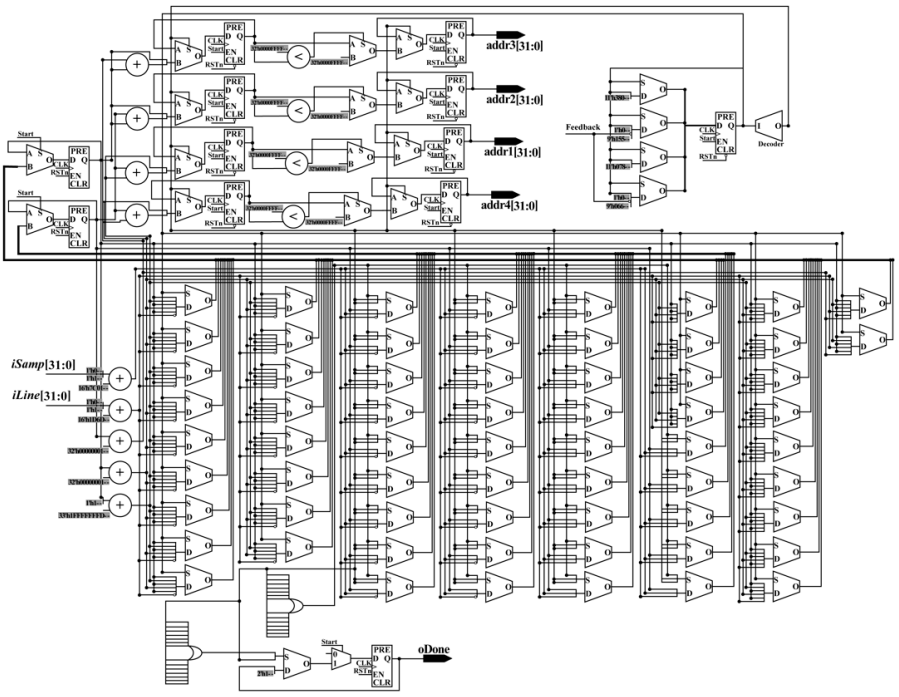


Figure 12. Schematic diagram of the get_gray_addr_mod module.

- **calculate_sum_mod**

As shown in Figure 10, after the multiplication process, four variables, x_1, x_2, x_3, and x_4, are obtained in the same clock cycle. To implement the addition for four variables, two levels of additions are needed. Each addition corresponds to an extended addition that has an architecture that is similar to Figure 7. The details can be found in Section 2.2.2 and Figure 7. After the processing of the calculate_sum_mod, the result of interpolation in (*Samp*, *Line*) can be obtained.

2.3. Integration On-Board System

An FPGA device can be integrated into the on-board system as a part of the system, because FPGA has advantages in size, weight, and power. After completing the proposed algorithm design using Verilog language, the designed algorithm can be programmed into the selected FPGA device.

3. Experiments

3.1. Software and Hardware Environment

In this study, an Altera FPGA was used. The version of the FPGA is Kintex-7 XC7K325TFFG900-1 (see Figure 13), the design tool is Vivado 2016.4 (Xilinx, San Jose, CA, USA), and the simulation tool is ModelSim SE10.1d (Mentor, Santa Barbara, CA, USA). The PC uses a Windows 7 (64 bit) operating system, and has an Intel® Core™ i7-4790 CPU @ 3.6 GHz processor with 8 GB RAM. To validate the proposed method, the orthorectification algorithm was also implemented using Matlab 2012a (MathWorks, 1 Apple Hill Drive, Natick, MA, USA).

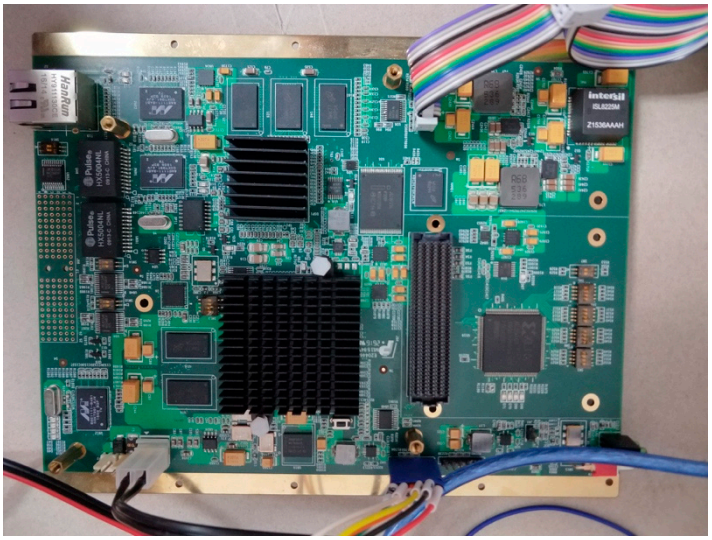


Figure 13. Photograph of the FPGA platform.

3.2. Dataset

To validate the correction accuracy and processing speed of the proposed FPGA-based orthorectification method, two test datasets (as shown in Figure 14) were used in this study. The first study area is located in San Diego, CA, USA. The IKONOS-2 PAN image with the resolution of 1.0 m was collected on 7 February 2000. The wavelength range of IKONOS-2 PAN image is 450–900 nm. The second study area is located in Genhe, Inner Mongolia, China. The SPOT-6 PAN image with the resolution of 1.5 m was acquired on 29 September 2013. The wavelength range of SPOT-6 PAN image is 450–745 nm. The known parameters are listed in Tables 2–4.

(**a**) (**b**)

Figure 14. (**a**) Original IKONOS image; and (**b**) original SPOT6 image.

Table 2. Normalized parameters.

#	First Area	Second Area
$Line_{off}$ (pixels)	1135	24,874.5
$Samp_{off}$ (pixels)	2548	17,962.5
Lat_{off} (degrees)	32.718700	50.737358
Lon_{off} (degrees)	−117.13340	121.44648
H_{off} (meters)	36.000	500
$Line_{scale}$ (pixels)	1829	24,874.5
$Samp_{scale}$ (pixels)	6570	17,962.5
Lat_{scale} (degrees)	0.01710000	0.41058849
Lon_{scale} (degrees)	0.07090000	0.45794952
H_{scale} (meters)	223	500

Table 3. Rational function polynomial coefficients of the first study area.

#	Values	#	Values	#	Values	#	Values
a_1	$-7.52883250 \times 10^{-4}$	b_1	1	c_1	$-9.23491680 \times 10^{-4}$	d_1	1
a_2	$4.60115225 \times 10^{-3}$	b_2	$-1.68736561 \times 10^{-3}$	c_2	1.01134804	d_2	$-1.68736561 \times 10^{-3}$
a_3	−1.03642070	b_3	$1.88384395 \times 10^{-3}$	c_3	$3.57115249 \times 10^{-4}$	d_3	$1.88384395 \times 10^{-3}$
a_4	$-3.93943040 \times 10^{-2}$	b_4	$-6.55340329 \times 10^{-4}$	c_4	$-1.17541741 \times 10^{-2}$	d_4	$-6.55340329 \times 10^{-4}$
a_5	$1.75874570 \times 10^{-3}$	b_5	$2.11928788 \times 10^{-7}$	c_5	$1.71162003E \times 10^{-3}$	d_5	$2.11928788 \times 10^{-7}$
a_6	$2.25762210 \times 10^{-4}$	b_6	$-2.15886792 \times 10^{-7}$	c_6	$-2.77384658 \times 10^{-4}$	d_6	$-2.15886792 \times 10^{-7}$
a_7	$6.47497342 \times 10^{-4}$	b_7	$6.60194370 \times 10^{-8}$	c_7	$-4.67286564 \times 10^{-5}$	d_7	$6.60194370 \times 10^{-8}$
a_8	$-1.22344418 \times 10^{-3}$	b_8	$1.29969058 \times 10^{-6}$	c_8	$-1.70712175 \times 10^{-3}$	d_8	$1.29969058 \times 10^{-6}$
a_9	$-1.95386510 \times 10^{-3}$	b_9	$-6.96485750 \times 10^{-7}$	c_9	$7.61699396 \times 10^{-7}$	d_9	$-6.96485750 \times 10^{-7}$
a_{10}	$2.70799645 \times 10^{-5}$	b_{10}	$3.41030606 \times 10^{-7}$	c_{10}	$2.98181047 \times 10^{-6}$	d_{10}	$3.41030606 \times 10^{-7}$
a_{11}	$5.10672113 \times 10^{-7}$	b_{11}	$5.17265975 \times 10^{-10}$	c_{11}	$8.68077245 \times 10^{-7}$	d_{11}	$5.17265975 \times 10^{-10}$
a_{12}	$2.05965613 \times 10^{-6}$	b_{12}	$2.71171743 \times 10^{-10}$	c_{12}	$1.42413537 \times 10^{-6}$	d_{12}	$2.71171743 \times 10^{-10}$
a_{13}	$-2.18726634 \times 10^{-7}$	b_{13}	$-1.47633205 \times 10^{-10}$	c_{13}	$-1.11312088 \times 10^{-6}$	d_{13}	$-1.47633205 \times 10^{-10}$
a_{14}	$5.40855097 \times 10^{-8}$	b_{14}	$3.59570414 \times 10^{-10}$	c_{14}	$2.35665930 \times 10^{-7}$	d_{14}	$3.59570414 \times 10^{-10}$
a_{15}	$-3.96996732 \times 10^{-6}$	b_{15}	$2.28588675 \times 10^{-10}$	c_{15}	$5.40107978 \times 10^{-7}$	d_{15}	$2.28588675 \times 10^{-10}$
a_{16}	$7.19308892 \times 10^{-7}$	b_{16}	$-1.11864088 \times 10^{-10}$	c_{16}	$-1.10872161 \times 10^{-10}$	d_{16}	$-1.11864088 \times 10^{-10}$
a_{17}	$-3.89372910 \times 10^{-7}$	b_{17}	$-1.37823694 \times 10^{-10}$	c_{17}	$-4.86264967 \times 10^{-10}$	d_{17}	$-1.37823694 \times 10^{-10}$
a_{18}	$-4.18443985 \times 10^{-6}$	b_{18}	$-3.32951199 \times 10^{-9}$	c_{18}	$-8.43531861 \times 10^{-7}$	d_{18}	$-3.32951199 \times 10^{-9}$
a_{19}	$4.50802285 \times 10^{-8}$	b_{19}	$6.33689691 \times 10^{-10}$	c_{19}	$-3.64346611 \times 10^{-8}$	d_{19}	$6.33689691 \times 10^{-10}$
a_{20}	$-1.57227534 \times 10^{-8}$	b_{20}	$-5.49482473 \times 10^{-11}$	c_{20}	$-2.38643375 \times 10^{-9}$	d_{20}	$-5.49482473 \times 10^{-11}$

Table 4. Rational function polynomial coefficients of the second study area.

#	Values	#	Values	#	Values	#	Values
a_1	0.00207581	b_1	1	c_1	−0.01727220	d_1	1
a_2	0.05939323	b_2	$5.06562471 \times 10^{-9}$	c_2	1.01955596	d_2	$-2.43637767 \times 10^{-6}$
a_3	−1.06139835	b_3	$-2.23329117 \times 10^{-9}$	c_3	0.00149223	d_3	$1.76928700 \times 10^{-6}$
a_4	0.00300505	b_4	$-2.21235982 \times 10^{-11}$	c_4	−0.00498582	d_4	$-1.29701506 \times 10^{-7}$
a_5	$9.99447200 \times 10^{-5}$	b_5	$-3.85166222 \times 10^{-8}$	c_5	−0.01544990	d_5	$-4.50015117 \times 10^{-5}$
a_6	$4.48210655 \times 10^{-6}$	b_6	$7.22875706 \times 10^{-11}$	c_6	0.00070933	d_6	$1.03746933 \times 10^{-6}$
a_7	−0.00011601	b_7	$1.96276656 \times 10^{-9}$	c_7	−0.00021711	d_7	$-2.16896140 \times 10^{-6}$
a_8	−0.00285285	b_8	$1.03082157 \times 10^{-8}$	c_8	0.01454522	d_8	$2.27253718 \times 10^{-5}$
a_9	−0.00025849	b_9	$4.58273834 \times 10^{-8}$	c_9	0.00146642	d_9	$2.71848287 \times 10^{-5}$
a_{10}	$8.66439267 \times 10^{-8}$	b_{10}	$4.12786890 \times 10^{-12}$	c_{10}	$-2.61417544 \times 10^{-6}$	d_{10}	$-1.20465997 \times 10^{-7}$
a_{11}	$1.94846265 \times 10^{-7}$	b_{11}	$-8.77450090 \times 10^{-11}$	c_{11}	$-2.38142758 \times 10^{-5}$	d_{11}	$-8.45078827 \times 10^{-9}$
a_{12}	$7.22766464 \times 10^{-7}$	b_{12}	$-1.25925035 \times 10^{-9}$	c_{12}	$4.37422992 \times 10^{-5}$	d_{12}	$-2.25373082 \times 10^{-9}$
a_{13}	$-2.45655192 \times 10^{-5}$	b_{13}	$-1.41399102 \times 10^{-9}$	c_{13}	−0.00012373	d_{13}	$1.66156835 \times 10^{-7}$
a_{14}	$-2.15776820 \times 10^{-10}$	b_{14}	$1.09626013 \times 10^{-12}$	c_{14}	$3.1682470454 \times 10^{-7}$	d_{14}	$-2.87815787 \times 10^{-10}$
a_{15}	$3.53191253 \times 10^{-5}$	b_{15}	$1.82192638 \times 10^{-9}$	c_{15}	0.00022374	d_{15}	$-1.35493121 \times 10^{-7}$
a_{16}	$3.58935305 \times 10^{-5}$	b_{16}	$-8.74976255 \times 10^{-10}$	c_{16}	$1.88815906 \times 10^{-5}$	d_{16}	$-2.31068042 \times 10^{-8}$
a_{17}	$-3.00220333 \times 10^{-9}$	b_{17}	$2.05074275 \times 10^{-13}$	c_{17}	$-1.37072926 \times 10^{-7}$	d_{17}	$5.66675066 \times 10^{-10}$
a_{18}	$-3.32028434 \times 10^{-7}$	b_{18}	$8.02116204 \times 10^{-11}$	c_{18}	$1.95362054 \times 10^{-5}$	d_{18}	$-2.37161908 \times 10^{-9}$
a_{19}	$-1.44250161 \times 10^{-7}$	b_{19}	$-1.15845372 \times 10^{-10}$	c_{19}	$2.68897003 \times 10^{-6}$	d_{19}	$6.41890372 \times 10^{-9}$
a_{20}	$1.88636696 \times 10^{-12}$	b_{20}	$5.66352056 \times 10^{-16}$	c_{20}	$-1.15446432 \times 10^{-9}$	d_{20}	$-7.23372539 \times 10^{-12}$

According to the proposed method, input parameters should be transformed into fixed-point data. As shown in Tables 2–4, the values of parameters of two study areas are in different range. To ensure computation accuracy, all parameters are transformed into fixed-point of 32 bits using different scale factor, τ. The details are given in Tables 5 and 6. In addition, the clock frequency is 100 MHz.

Table 5. Scale factors for parameters of First Area.

	τ	Range	Accuracy
Lat', Lon', Hei'	23	(−256, 255.999999881)	0.000000119
Lat'_{off}, Lat'_{scale}, Lon'_{off}, Lon'_{scale}	23	(−256, 255.999999881)	0.000000119
H'_{off}, H'_{scale}	23	(−256, 255.999999881)	0.000000119
$Line'_{off}$, $Line'_{scale}$, $Samp'_{off}$, $Samp'_{scale}$	18	(−8192, 8191.999996185)	0.000003815
a'_i, b'_i, c'_i, and d'_i (i = 1 to 20)	30	(−2, 1.999999999)	0.000000001

Table 6. Scale factors for parameters of Second Area.

	τ	Range	Accuracy
Lat', Lon', Hei'	23	(−256, 255.999999881)	0.000000119
Lat'_{off}, Lat'_{scale}, Lon'_{off}, Lon'_{scale}	23	(−256, 255.999999881)	0.000000119
H'_{off}, H'_{scale}	21	(−1024, 1023.999999523)	0.000000477
$Line'_{off}$, $Line'_{scale}$, $Samp'_{off}$, $Samp'_{scale}$	16	(−32,768, 32,767.999984741)	0.000015259
a'_i, b'_i, c'_i, and d'_i (i = 1 to 20)	30	(−2, 1.999999999)	0.000000001

3.3. Results

As shown in Figures 15a and 16a, after the processing of the proposed method, the orthorectified results (orthophoto) were obtained. To validate the accuracy and speed of the proposed rectification method, orthorectification for the same datasets was also implemented by applying the PC-based platform. On the PC-based platform, the proposed FP-RPC orthorectification was implemented using Matlab codes.

The orthorectification results obtained using the PC-based software are shown in Figures 15b and 16b. The contrast-enhanced difference images for two study areas are shown in Figures 15c and 16c, respectively. As shown in Figures 15c and 16c, the contrast-enhanced difference images show the few discrepancies that are present. The orthorectification images obtained by FPGA and PC are not visually

different by inspection. The numerical differences between FPGA and PC become apparent when observing the difference images shown in Figures 15c and 16c. These pixel position differences are mainly caused by the used bit wide and scale factor of fixed-point data [1]. According to [1], the pixel position difference can be decreased with the increasing of bit wide and scale factor. Error analysis between the proposed method and the FP-RPC algorithm implemented on PC are provided in the next section.

(**a**) (**b**) (**c**)

Figure 15. Orthoimages for the first study area: (**a**) by FPGA; and (**b**) by Matlab; and (**c**) the difference of image (**a**) and image (**b**).

(**a**) (**b**) (**c**)

Figure 16. Orthoimages for the second study area: (**a**) by FPGA; and (**b**) by Matlab; and (**c**) the difference of image (**a**) and image (**b**).

4. Discussion

4.1. Error Analysis

To quantitatively evaluate the accuracy of the proposed orthorectification method, the root-mean-square error (RMSE) [40,41] was utilized. Mathematically, the RMSEs of the image coordinates along the vertical axis (ΔI) and horizontal axis (ΔJ), and distance (ΔS) can be calculated using the following equations, respectively,

$$\Delta I = \sqrt{\frac{\sum_{h=1}^{n} (I'_h - I_h)^2}{n-1}} \quad \Delta J = \sqrt{\frac{\sum_{h=1}^{n} (J'_h - J_h)^2}{n-1}} \tag{19}$$

$$\Delta S = \sqrt{\frac{\sum_{h=1}^{n} ((I'_h - I_h)^2 + (J'_h - J_h)^2}{n-1}} \tag{20}$$

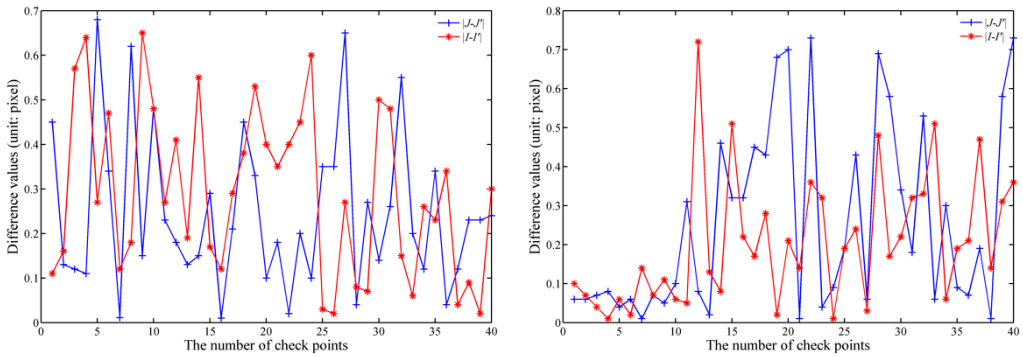
where I'_h and J'_h are the image coordinates rectified by the proposed orthorectification method; I_h and J_h are the reference image coordinates; and n is the number of check points.

To compute the RMSEs, 40 check points for each study area were selected randomly (as shown in Figure 17). As shown in Figure 18, the differences in the values of image coordinates for the Matlab-based and FPGA-based methods are given. Based on Equations (19) and (20), the RMSEs (ΔI, ΔJ, and ΔS) are 0.35 pixels, 0.30 pixels, and 0.46 pixels, respectively, for the first study area; meanwhile, they are 0.27 pixels, 0.36 pixels, and 0.44 pixels, respectively, for the second study area. Moreover, other statistics were also calculated (as shown in Table 7).

According to the calculation results of Equations (19) and (20), the orthorectification results obtained using the proposed method are considered acceptable because the RMSEs are less than one pixel [42–44]. However, as shown in Figure 18, differences still exist in the image coordinates acquired by the FPGA-based and Matlab-based methods. These differences may be caused by the algorithms implemented by FPGA hardware, for example, the fixed-point computation, which propagate and accumulate.

(**a**) (**b**)

Figure 17. Check-point distribution in: (**a**) the first study area; and (**b**) the second study area.



(**a**) (**b**)

Figure 18. Different statistical analyses for the FPGA-based method and Matlab-based method for: (**a**) the first study area; and (**b**) the second study area.

Table 7. Statistical analysis for the different image coordinates obtained by Matlab and FPGA (unit: pixel).

Study Area No.		Maximum	Minimum	Mean	STD
1st	$\|I'-I\|$	0.65	0.02	0.29	0.19
	$\|J'-J\|$	0.68	0.01	0.25	0.18
2nd	$\|I'-I\|$	0.72	0.01	0.20	0.16
	$\|J'-J\|$	0.73	0.01	0.26	0.24

4.2. Processing Speed Comparison

This section presents the processing time as the size of satellite image increases, and evaluates processing speed of the FPGA-based orthorectification method and the Matlab-based method.

The processing time have been recorded as an average of 10 runs of the RPC orthorectification algorithm for each image. The average processing time for difference size of image is presented in Table 8. A plot of the image size vs. processing time is shown in Figure 19. The speed-up of the method can be defined as the Matlab time taken divided by the time taken on the FPGA for the performance of the RPC algorithm [45]. In the image case considered, the maximum speed-up is acquired from a size of 1024 × 1024 pixels, where the speed-up is about 11,095.8709. From the results in Table 8, it can be demonstrated that the speed increases with the size of image.

Table 8. Average processing time for RPC orthorectification implementation on Matlab and FPGA.

No.	Image Size (Pixels)	Matlab Time (s)	FPGA Time (ms)	Speed-Up
1	256 × 256	1.0515	0.09686	10,855.8745
2	512 × 512	4.2461	0.3875	11,008.8151
3	1024 × 1024	17.1986	1.5500	11,095.8709

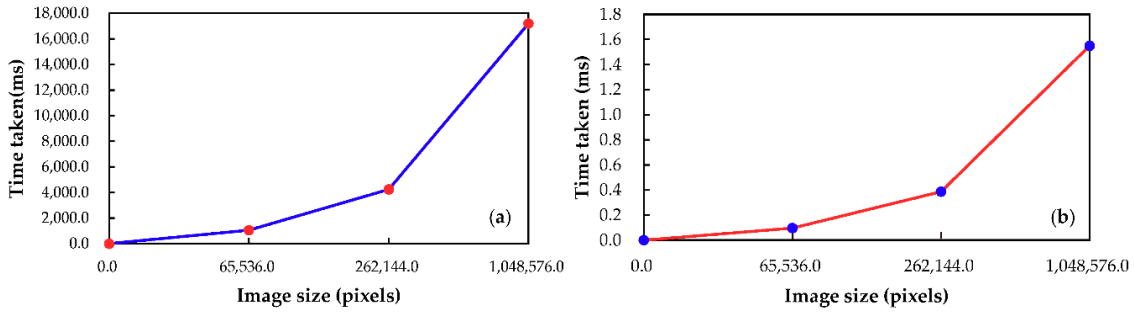


Figure 19. Image size vs. time taken to perform RPC orthorectification algorithm: (**a**) Matlab-based platform; and (**b**) FPGA-based platform.

The processing speed is one of the most importance indicators for evaluating on-board processing. To evaluate and compare the speed of the proposed FPGA-based orthorectification method and the Matlab-based method, the throughput, which is a normalized metric, is used, and represents the capacity in terms of the number of pixels processed per second. For the proposed method, the average throughput is approximately 675.67 Mpixels/s. However, for the Matlab-based method, the average throughput is approximately 61,677.49 pixels/s. This means that the proposed FPGA-based method has higher processing capacity than the Matlab-based method.

4.3. Resource Consumption

Besides the speed of processing, the utilization ratio of each type of resource is also a key indicator when assessing the quality of a method. As is well known, it can be determined whether a selected

device meets the requirement of a design scheme by analyzing the utilization ratio of hardware resource. If the utilization ratio of a type of resource reaches 60–80%, the selected device satisfies the requirement of the design scheme.

Thus, after implementing the proposed method, some important resources, such as look-up tables (LUTs), registers, and total pins are analyzed. As shown in Table 9, the slice logics contain slice LUTs and slice registers. The utilization ratios of LUTs and registers are 44.42% and 5.59%, respectively. The utilization of input and output (IO) is 368, which is 73.60% of the total IOs.

In short, according to the above comprehensive utilization ratios for resources, it can be demonstrated that the resources of the selected FPGA can meet the design requirement of the proposed FPGA-based orthorectification method.

Table 9. Utilization ratio of resources for the proposed FPGA-based orthorectification.

#		Utilization	Available	Utilization Ratio (%)
Slice logic	Slice LUTs	90,634	203,800	44.42
	Slice registers	22,798	407,600	5.59
IO		368	500	73.60

5. Conclusions

This paper proposes an orthorectification method, namely, the field-programmable gate array (FPGA)-based fixed-point (FP) rational polynomial coefficient (RPC) method (FPGA-based FP-RPC method) to perform the process of orthorectification on board spacecraft/satellite to accelerate the orthorectification processing speed for remotely sensed (RS) images. The proposed FPGA-based FP-RPC method contains three main submodules, Read_parameter_mod, Coordinate_transform_mod, and Interpolation_mod, based on the bilinear interpolation algorithm.

To validate the orthorectification accuracy, an orthophoto that was orthorectified by a PC-based platform (Matlab 2012a) was used as a reference. Two datasets, IKONOS and SPOT-6 images, were used to validate the proposed FPGA-based FP-RPC method. The root-mean-square error (RMSE), which is associated with the maximum, minimum, standard deviation (STD), and mean of row and column coordinates' differences, was used. The experimental results show that the STD of the row and column coordinates' differences are 0.19 pixels and 0.18 pixels, respectively, for the first study area, while they were 0.16 pixels and 0.24 pixels, respectively, for the second study area. The RMSE of the row coordinate (ΔI), column coordinate (ΔJ) and the distance ΔS are 0.35 pixels, 0.30 pixels, and 0.46 pixels, respectively, for the first study area, while they are 0.27 pixels, 0.36 pixels, and 0.44 pixels, respectively, for the second study area. It can be concluded from these quantitative analyses that the proposed method can meet the demand of orthorectification in practice.

Moreover, a comparison of the processing speed was also performed for the proposed FPGA-based FP-RPC method and PC-based RPC methods. The throughput of the FPGA-based FP-RPC method and PC-based RPC method are 675.67 Mpixels/s and 61,070.24 pixels/s, respectively. Therefore, it can be shown that the processing speed of the FPGA-based FP-RPC method is faster (by approximately 11,000 times) than the processing speed of the Matlab-based RPC method. In terms of the resource consumptions, it can be found that the utilization ratios of ALUTs, registers, and IO are 44.42%, 5.59%, and 73.60%, respectively.

Author Contributions: G.Z. (Guoqing Zhou) contributed the most to the manuscript. He contributed the whole idea of this manuscript, designed the experiments and reviewed and revised the manuscript. R.Z. performed the experiments and analyzed the data. R.Z. wrote the paper. G.Z. (Guangyun Zhang), X.Z. and J.H. contributed to editing and reviewed the manuscript.

Funding: This paper was financially supported by the National Natural Science of China under Grant numbers 41431179, 41601365, ; Guangxi Innovative Development Grand Grant, entitled, "Research and Development of Bathymetric Mapping LiDAR under the number 2018AA13005, the National Key Research and Development Program of China under Grant numbers 2016YFB0502501; GuangXi Natural Science Foundation under grant

numbers 2015GXNSFDA139032; Guangxi Science & Technology Development Program under the Contract number GuiKeHe 14123001-4; GuangXi Key Laboratory of Spatial Information and Geomatics Program (Contract Nos. GuiKeNeng 163802506 and 163802530); the Natural Science Foundation of Tianjin (Grant No. #: 14JCYBJC41700); the National Natural Science Foundation of China (grant No. 41601446); and Tianjin Natural Science Foundation (grant No.16JCQNJC01200).

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

References

1. French, J.C.; Balster, E.J. A fast and accurate orthorectification algorithm of aerial imagery using integer arithmetic. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2014**, *7*, 1826–1834. [CrossRef]
2. Zhou, G.Q.; Chen, W.; Kelmelis, J.; Zhang, D.Y. A comprehensive study on urban true orthorectification. *IEEE Trans. Geosci. Remote Sens.* **2005**, *43*, 2138–2147. [CrossRef]
3. Zhou, G.Q. Near real-time orthorectification and mosaic of small UAV video flow for time-critical event response. *IEEE Trans. Geosci. Remote Sens.* **2009**, *47*, 739–747. [CrossRef]
4. Aguilar, M.A.; Saldaña, M.D.M.; Aguilar, F.J. Assessing geometric accuracy of the orthorectification process from GeoEye-1 and WorldView-2 panchromatic images. *Int. J. Appl. Earth Obs.* **2013**, *21*, 427–435. [CrossRef]
5. Marsetič, A.; Oštir, K.; Fras, M.K. Automatic orthorectification of high-resolution optical satellite images using vector roads. *IEEE Trans. Geosci. Remote Sens.* **2015**, *53*, 6035–6047. [CrossRef]
6. Habib, A.; Xiong, W.; Yang, F.; HeH, L.; Crawford, M. Improving orthorectification of UAV-based pushbroom scanner imagery using derived orthophotos from frame cameras. *IEEE J.-STARS* **2017**, *10*, 262–276. [CrossRef]
7. Warpenburg, M.R.; Siegel, L.J. SIMD image resampling. *IEEE Trans. Comput.* **1982**, *31*, 934–942. [CrossRef]
8. Wittenbrink, C.M.; Somani, A.K. 2D and 3D optimal parallel image warping. In Proceedings of the Seventh International Parallel Processing Symposium, Newport, CA, USA, 13–16 April 1993; pp. 331–337.
9. Liu, H.; Yang, J.; Liu, H.; Zhang, J. A new parallel ortho-rectification algorithm in a cluster environment. In Proceedings of the Third International Congress on Image and Signal Processing, Yantai, China, 16–18 October 2010; pp. 2080–2084.
10. Dai, C.; Yang, J. Research on orthorectification of remote sensing images using GPU-CPU cooperative processing. In Proceedings of the International Symposium on Image and Data Fusion, Tengchong, China, 9–11 August 2011; pp. 1–4.
11. 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]
12. Quan, J.; Wang, P.; Wang, H. Orthorectification of optical aerial images by GPU acceleration. *Opt. Precis. Eng.* **2016**, *24*, 2863–2871. [CrossRef]
13. Halle, W.; Venus, H.; Skrbek, W. Thematic data processing on board the satellite BIRD. In Proceedings of the SPIE 4132, Imaging Spectrometry VI, Toulouse, France, 15 November 2000; pp. 412–419.
14. Eadie, D.; Shevlin, F.; Nisbet, A. Correction of geometric image distortion using FPGAs. In Proceedings of the SPIE—The International Society for Optical Engineering, Galway, Ireland, 19 March 2003.
15. Kumar, P.R.; Sridharan, K. VLSI-efficient scheme and FPGA realization for robotic mapping in a dynamic environment. *IEEE Trans. VLSI Syst.* **2007**, *15*, 118–123. [CrossRef]
16. Escamilla-Hernández, E.; Kravchenko, V.; Ponomaryov, V.; Robles-Camarillo, D.; Ramos, L.E. Real time signal compression in radar using FPGA. *Científica* **2008**, *12*, 131–138.
17. Kate, D. Hardware implementation of the huffman encoder for data compression using Altera DE2 board. *Int. J. Adv. Eng. Sci.* **2012**, *2*, 11–15. [CrossRef]
18. Tomasi, M.; Vanegas, M.; Barranco, F.; Diaz, J.; Ros, E. Real-time architecture for a robust multi-scale stereo engine on FPGA. *IEEE Trans. VLSI Syst.* **2012**, *20*, 2208–2219. [CrossRef]
19. Pal, C.; Kotal, A.; Samanta, A.; Chakrabarti, A.; Ghosh, R. An efficient FPGA implementation of optimized anisotropic diffusion filtering of images. *Int. J. Reconfig. Comput.* **2016**, *2016*, 3020473. [CrossRef]
20. Wang, E.; Yang, F.; Tong, G.; Qu, P.; Pang, T. Particle filtering approach for GNSS receiver autonomous integrity monitoring and FPGA implementation. *TELKOMNIKA* **2016**, *14*, 1321–1328. [CrossRef]
21. Zhang, C.; Liang, T.; Mok, P.K.T.; Yu, W. FPGA implementation of the coupled filtering method. In Proceedings of the 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Shenzhen, China, 15–18 December 2016; pp. 435–442.

22. Ontiveros-Robles, E.; Gonzalez-Vazquez, J.L.; Castro, J.R.; Castillo, O. A hardware architecture for real-time edge detection based on interval type-2 fuzzy logic. In Proceedings of the 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Vancouver, BC, Canada, 24–29 July 2016; pp. 804–810.
23. Ontiveros-Robles, E.; Vázquez, J.G.; Castro, J.R.; Castillo, O. A FPGA-based hardware architecture approach for real-time fuzzy edge detection. In *Nature-Inspired Design of Hybrid Intelligent Systems*; Melin, P., Castillo, O., Kacprzyk, J., Eds.; Springer International Publishing: Basel, Switzerland, 2017; pp. 519–540, ISBN 978-3-319-47054-2.
24. Li, H.H.; Liu, S.; Piao, Y. Snow removal of video image based on FPGA. In *Proceedings of the 5th International Conference on Electrical Engineering and Automatic Control*; Huang, B., Yao, Y., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 207–215, ISBN 978-3-662-48768-6.
25. Li, H.; Xiang, F.; Sun, L. Based on the FPGA video image enhancement system implementation. In Proceedings of the International Conference on Electronic Information Technology and Intellectualization, Hong Kong, China, 24–26 June 2016; pp. 427–434.
26. Huang, J.; Zhou, G. On-board detection and matching of feature points. *Remote Sens.* **2017**, *9*, 601. [CrossRef]
27. Huang, J.; Zhou, G.; Zhou, X.; Zhang, R. A new FPGA architecture of fast and BRIEF algorithm for on-board corner detection and matching. *Sensors* **2018**, *18*, 1014. [CrossRef] [PubMed]
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. Fraser, C.S.; Hanley, H.B.; Yamakawa, T. Three-dimensional geopositioning accuracy of IKONOS imagery. *Photogramm. Rec.* **2002**, *17*, 465–479. [CrossRef]
30. Grodecki, J.; Dial, G. Block adjustment of high-resolution satellite images described by rational polynomials. *Photogramm. Eng. Remote Sens.* **2003**, *69*, 59–68. [CrossRef]
31. Wang, H.; Ellis, E.C. Spatial accuracy of orthorectified IKONOS imagery and historical aerial photographs across five sites in China. *Int. J. Remote Sens.* **2005**, *26*, 1893–1911. [CrossRef]
32. Hoja, D.; Schneider, M.; Müller, R.; Lehner, M.; Reinartz, P. Comparison of orthorectification methods suitable for rapid mapping using direct georeferencing and RPC for optical satellite data. In Proceedings of the ISPRS Conference 2008, Peking, China, 3–11 July 2008; pp. 1617–1624.
33. Zhang, G.; Qiang, Q.; Luo, Y.; Zhu, Y.; Gu, H.; Zhu, X. Application of RPC model in orthorectification of spaceborne SAR imagery. *Photogramm. Rec.* **2012**, *27*, 94–110. [CrossRef]
34. Yang, G.D.; Zhu, X. Ortho-rectification of SPOT 6 satellite images based on RPC models. *Appl. Mech. Mater.* **2013**, *392*, 808–814. [CrossRef]
35. Yang, G.; Xin, X.; Wu, Q. A study on ortho-rectification of SPOT6 image. In Proceedings of the 2017 International Conference on Mechanical and Mechatronics Engineering (ICMME 2017), Bangkok, Thailand, 26–27 March 2017.
36. Ferrer, M.A.; Alonso, J.B.; Travieso, C.M. Offline geometric parameters for automatic signature verification using fixed-point arithmetic. *IEEE Trans. Pattern Anal.* **2005**, *27*, 993–997. [CrossRef] [PubMed]
37. Balster, E.J.; Fortener, B.T.; Turri, W.F. Integer computation of lossy JPEG2000 compression. *IEEE Trans. Image Process.* **2011**, *20*, 2386–2391. [CrossRef] [PubMed]
38. Tao, C.V.; Hu, Y. A comprehensive study of the rational function model for photogrammetric processing. *Photogramm. Eng. Remote Sens.* **2001**, *67*, 1347–1357.
39. Zhou, G.; Jiang, L.; Huang, J.; Zhang, R.; Liu, D.; Zhou, X.; Baysal, O. FPGA-based on-board geometric calibration for linear CCD array sensors. *Sensors* **2018**, *18*, 1794. [CrossRef] [PubMed]
40. Shi, W.; Shaker, A. Analysis of terrain elevation effects on IKONOS imagery rectification accuracy by using non-rigorous models. *Photogramm. Eng. Remote Sens.* **2003**, *69*, 1359–1366. [CrossRef]
41. Reinartz, P.; Müller, R.; Lehner, M.; Schroeder, M. Accuracy analysis for DSM and orthoimages derived from SPOT HRS stereo data using direct georeferencing. *ISPRS J. Photogramm. Remote Sens.* **2006**, *60*, 160–169. [CrossRef]
42. Schowengerdt, R.A. CHAPTER 7—Correction and Calibration. In *Remote Sensing*, 3rd ed.; Academic Press: Cambridge, MA, USA, 2007; p. 285-XXIII, ISBN 978-0-12-369407-2.
43. Schowengerdt, R.A. CHAPTER 8—Image Registration and Fusion. In *Remote Sensing*, 3rd ed.; Academic Press: Cambridge, MA, USA, 2007; ISBN 978-0-12-369407-2.

44. Richards, J.A.; Jia, X. Remote sensing digital image analysis: An introduction. In *Remote Sensing Digital Image Analysis: An Introduction*; Richards, J.A., Jia, X., Eds.; Springer: Berlin/Heidelberg, Germany, 1999; pp. 39–74, ISBN 978-3-642-30062-2.
45. Senthilnath, J.; Sindhu, S.; Omkar, S.N. GPU-based normalized cuts for road extraction using satellite imagery. *J. Earth Syst. Sci.* **2014**, *123*, 1759–1769. [CrossRef]