sensors

MDPI

Article

# On-Board, Real-Time Preprocessing System for Optical Remote-Sensing Imagery

**Baogui Qi [1], Hao Shi [1,2,*], Yin Zhuang [1], He Chen [1] and Liang Chen [1]**

[1] Beijing Key Laboratory of Embedded Real-Time Information Processing Technology, Beijing Institute of Technology, Beijing 100081, China; qibaogui@bit.edu.cn (B.Q.); zhuangyin640829@163.com (Y.Z.); chenhe@bit.edu.cn (H.C.); chenl@bit.edu.cn (L.C.)

[2] Department of Electronic Engineering, Tsinghua University, Beijing 100084, China

* Correspondence: shihao@tsinghua.edu.cn; Tel.: +86-186-1166-1399

Received: 28 March 2018; Accepted: 18 April 2018; Published: 25 April 2018


**Abstract:** With the development of remote-sensing technology, optical remote-sensing imagery processing has played an important role in many application fields, such as geological exploration and natural disaster prevention. However, relative radiation correction and geometric correction are key steps in preprocessing because raw image data without preprocessing will cause poor performance during application. Traditionally, remote-sensing data are downlinked to the ground station, preprocessed, and distributed to users. This process generates long delays, which is a major bottleneck in real-time applications for remote-sensing data. Therefore, on-board, real-time image preprocessing is greatly desired. In this paper, a real-time processing architecture for on-board imagery preprocessing is proposed. First, a hierarchical optimization and mapping method is proposed to realize the preprocessing algorithm in a hardware structure, which can effectively reduce the computation burden of on-board processing. Second, a co-processing system using a field-programmable gate array (FPGA) and a digital signal processor (DSP; altogether, FPGA-DSP) based on optimization is designed to realize real-time preprocessing. The experimental results demonstrate the potential application of our system to an on-board processor, for which resources and power consumption are limited.

**Keywords:** remote sensing; preprocessing; relative radiation correction; geometric correction; real-time

## 1. Introduction

Remote-sensing techniques are increasingly used in geological exploration, natural disaster prevention, monitoring, etc. [1–5]. They usually require very high-resolution satellite images, which are texturally-rich and may raise the running costs of systems. However, many remote-sensing satellites must rapidly respond to emergencies, such as fires and earthquakes, and quickly return the region of interest (ROI) of the emergency to the ground station [6]. In general processing procedure, the satellite image data are downlinked to the ground station for processing and analysis. The data size of Earth's observation satellites often exceeds 10 GB. So, the process of data downlink causes a long delay time, and it severely affects rapid response to emergencies [7–9]. On-board processing is a way to effectively improve response speed and provide immediate products for rapid decision-making [10–12]. After processing and sending the data about which we are most concerned, the amount of data can be reduced several times. Therefore, by processing the data on-board and downlinking the processing results only, the communication bandwidth of downlink can be reduced. At the same time, the data processing flow of the ground station can simultaneously be accelerated and simplified. Consequently,

on-board processing can reduce the cost and complexity of ground processing systems and solve the delay problem in image acquisition, analysis, and application.

The acquired remote-sensing images may contain uneven radiation brightness stripes and deformation areas, due to the defects of the sensors and the relative movement between satellite platforms and the Earth [13–15]. Therefore, the acquired raw data from sensors on satellite platforms cannot be used directly. So, image preprocessing is a necessary step to solve such crucial problems. There are several necessary steps for preprocessing within charge coupled device (CCD) camera images, such as relative radiation correction (RRC), geometric correction (GC), and multi-CCD stitching (MCCDS).

Numerous studies have been performed to satisfy the needs of on-board processing. Cong Li et al. [16] introduced a new volume calculation formula and developed a new real-time implementation of a maximum simplex volume algorithm, which is suitable for real-time, on-board processing. Qian Du et al. [8] employed a small portion of pixels in the evaluation of data statistics to accelerate the real-time implementation of detection and classification. This design achieved fast, real-time, on-board processing by reducing computational complexity and simplifying hardware implementation.

Scholars have also conducted related studies of architecture implementation and efficient algorithm mapping. El-Araby et al. [10] presented a reconfigurable computing real-time cloud detection system for satellite on-board processing. Kalomiros et al. [17] designed a hardware/software field-programmable gate array (FPGA) system for fast image processing, which can be utilized for an extensive range of custom applications. Winfried et al. [18] designed an on-board, bispectral infrared detection system, which is based on the neural network processor NI1000, a digital signal processor (DSP), and a FPGA. The system can perform on-board radiometric correction, geometric correction, and texture extraction. Botella et al. [19] proposed an architecture for a neuromorphic, robust optical flow based on a FPGA, which was applied in a complicated environment. Multi-core processors and graphic processing units (GPUs) for achieving real-time performance of the Harsanyi–Farrand–Chang (HFC) method for a virtual dimensionality (VD) algorithm was proposed for unmixing [20]. Carlos et al. presented the first FPGA design for the HFC-VD algorithm to realize unmixing [21].

The previously mentioned methods—GPU, FPGA, and DSP—are the most common processors for implementing these algorithms in real time. In a ground processing system, a GPU is the popular choice for a preprocessing system. Although a GPU can provide high computing performance, it consumes considerable energy and cannot achieve the radiation tolerance required for an on-board environment. Therefore, a GPU cannot be adapted to an on-board processing system. To satisfy the requirements of on-board processing, this system should be implemented using a FPGA, which has low power consumption and high radiation resistance [22–24]. Considering the computational complexity of a preprocessing algorithm, the use of a DSP as a co-processor is common to perform processes that are not computationally demanding and need to be sporadically executed. Although some publications have designed GC systems based on a FPGA, these systems are not suitable for remote-sensing images [25–27] or cannot achieve the complete process [28]. To the best of our knowledge, no such hardware systems have been proposed for remote image preprocessing, probably because of the complex computations and data management required. However, such a preprocessing step should be executed on this platform to achieve higher performance.

The process of image preprocessing can be decomposed into two parts. The first step calculates the model parameters. This step processes small amounts of data but involves complex calculations (such as sine and cosine functions), making it suitable for a DSP. The second step uses the model parameters to perform a pixel-by-pixel, gray-scale calculation and obtain the output image. When the pixels are calculated in this step, parallel calculations are appropriate, because the calculation forms of all the pixels are similar. However, due to the irregularity of the image deformation and other issues, there are several problems in the pixel calculation step. First, the calculation of each pixel coordinate requires many parameters and a large amount of hardware computing resources.

Some parameters are involved in each pixel coordinate calculation and must be repeatedly calculated many times, thus wasting considerable time. Therefore, it is necessary to optimize the algorithm to improve computational efficiency. Second, due to the irregularity of the image deformation, the input and output data cannot be strictly correlated with each other, which makes it difficult to implement the pipeline process. Therefore, it is necessary to design the methods for reading and storing the data according to the characteristics of the geometric deformation. Third, existing algorithms use floating-point data for calculations. Compared with fixed-point calculations, floating-point calculations require more resources and more time. Because the amount of image data is large, it is very important to design a fixed-point solution to speed up the process.

Therefore, we optimized the design of the preprocessing algorithm regarding these aspects of the hardware implementation. First, a hierarchical decomposition mapping method based on coordinate transformation is proposed, which can effectively reduce the computation burden of on-board processing. Second, according to the characteristics of the data read and write irregularities, a block mapping design is implemented to avoid wasting time when reading and writing data. Finally, we design a fixed-point algorithm for the RRC and pixel resampling parts. The design can reduce resources and ensure accuracy. Using these technologies, an optical image preprocessing system based on FPGA and DSP coprocessors is designed and implemented. Because our system is designed for on-board processing, we chose processors with high radiation tolerance for space environments.

Thus, our contributions can be summarized as follows: first, we proposed a hierarchical optimization and mapping method to realize the preprocessing algorithm in a hardware structure, which can effectively reduce the computation burden of on-board processing. Second, a FPGA-DSP co-processing system based on optimization is designed to realize real-time preprocessing.

The remainder of this paper is structured as follows. The second section describes the preprocessing algorithm. The third section describes a mapping strategy and optimizing method. The fourth section describes the hardware realization and parallel accelerating design. The fifth section presents the experimental results and comparison with related studies. The last section provides conclusions and plans for future research.

## 2. Preprocessing Method

The complete process for optical remote-sensing CCD image data preprocessing is shown in Figure 1. The process we implemented consists of three parts: RRC, MCCDS, and GC. The input of the preprocessing chain is a raw image with its corresponding ancillary information (imaging time, orbit, attitude, and other necessary information). The output of the preprocessing chain is the georeferenced image. We call the image after the RRC the Level 0 image; the image after the MCCDS is the Level 1 image, and the image after the GC is the Level 2 image.

The RRC is used to remove the systematic noise introduced by the discrepancy in the optical-electronic responses between different detectors and can be described as follows:

$$y_i = k_i \times x_i + b_i, \tag{1}$$

where $b_i$ and $k_i$ represent the bias and gain coefficients, respectively, of the $i$th detector, which are provided by the manufacturer or calibration laboratory, and $x_i$ and $y_i$ correspond to the digital number value and the at-sensor radiance of the $i$th detector, respectively [29].

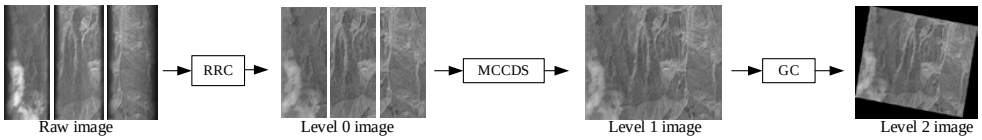



**Figure 1.** Preprocessing chain. RRC: relative radiation correction; MCCDS: multi-charge-coupled device (CCD) stitching; GC: geometric correction; Level 0: the image after the RRC; Level 1: the image after the MCCDs; and Level 2: the image after the GC.

The MCCDS is based on the virtual CCD and rational function model (RFM). We summarize the process in two steps. First, the image coordinates of the Level 1 image corresponding to a certain number of points in the Level 0 image are solved using the rigorous imaging model and the orbit, attitude, and auxiliary information. The Level 1 image rational polynomial coefficients (RPCs) for the RFM are calculated based on these coordinate relationships. Second, for each coordinate in the required Level 1 image, the corresponding coordinate in the Level 0 image is calculated via the RPCs, and the gray value is obtained by resampling. The RFM that is employed in this process is expressed as follows:

$$\begin{array}{l} s = (x_0 + a \times x_1 + b \times x_2 + a \times b \times x_3) \times s_{scale} + s_{off} \\ l = (y_0 + a \times y_1 + b \times y_2 + a \times b \times y_3) \times l_{scale} + l_{off} \end{array}, \quad (2)$$

where $a$ and $b$ are the row coordinates and column coordinates, respectively, of the Level 1 image; $s$ and $l$ are the row coordinates and column coordinates, respectively, of the Level 0 image; $x_0$, $x_1$, $x_2$, $x_3$, $y_0$, $y_1$, $y_2$, and $y_3$ are the respective polynomial coefficients; $s_{scale}$ and $l_{scale}$ are the scale factors; and $s_{off}$ and $l_{off}$ are the offsets.

The purpose of the GC is to correct the deformations that occur during imaging [30]. GC methods are divided into parametric and non-parametric models [31]. For on-board processing, it is more suitable to choose the parametric model, because the orbital information of the satellite platform can be obtained. The GC is based on the RFM. We summarize the process in two steps. First, the geographic coordinates in the Level 2 image that correspond to a certain number of points in the Level 1 image are solved using the rigorous imaging model, the RFM of the Level 1 image, and other information. Then, the RPCs for the RFM are solved based on the coordinate relationships. Second, for each geographic coordinate of the requested region in the Level 2 image, the corresponding image coordinate in the Level 1 image is calculated via the RPCs, and the gray value is obtained by resampling. The RFM used in this process is expressed as follows:

$$\begin{array}{l} s = \frac{x_0 + lon \times x_1 + lat \times x_2 + h \times x_3}{lon \times x_4 + lat \times x_5 + h \times x_6 + 1} \times s_{scale} + s_{off} \\ l = \frac{y_0 + lon \times y_1 + lat \times y_2 + h \times y_3}{lon \times y_4 + lat \times y_5 + h \times y_6 + 1} \times l_{scale} + l_{off} \end{array}, \quad (3)$$

where $s$ and $l$ are the pixel coordinates of the Level 1 image, $x_0$–$x_6$ and $y_0$–$y_6$ are RPCs, $lon$ is the longitude, $lat$ is the latitude, $h$ is the elevation, $s_{scale}$ and $l_{scale}$ are the scale factors, and $s_{off}$ and $l_{off}$ are the offsets.

After the coordinate transformation, we obtain the coordinates ($s$ and $l$) of the image pixels. Because the image is a discrete space grid, resampling is required to obtain the image gray values using the interpolation method. Because the bi-cubic interpolation method yields the best performance, we chose this method for our preprocessing algorithm. The bi-cubic interpolation method is shown in Figure 2, which can be described as

$$\begin{array}{rl} Q(u,v) & = [a_1 p_{11} + a_2 p_{21} + a_3 p_{31} + a_4 p_{41}] \times b_1 + [a_1 p_{12} + a_2 p_{22} + a_3 p_{32} + a_4 p_{42}] \times b_2 \\ & + [a_1 p_{13} + a_2 p_{23} + a_3 p_{33} + a_4 p_{43}] \times b_3 + [a_1 p_{14} + a_2 p_{24} + a_3 p_{34} + a_4 p_{44}] \times b_4 \end{array}, \quad (4)$$

where

$$\begin{array}{llll} a_1 = -t + 2t^2 - t^3 & a_2 = 1 - 2t^2 + t^3 & a_3 = t + t^2 - t^3 & a_4 = -t^2 + t^3 \\ b_1 = -s + 2s^2 - s^3 & b_2 = 1 - 2s^2 + s^3 & b_3 = s + s^2 - s^3 & b_4 = -s^2 + s^3 \end{array}, \quad (5)$$

and $Q(u,v)$ is the output pixel gray value, $(u,v)$ is the sample position, $p_{11}$ to $p_{44}$ are the original sample pixel gray values, $t = v - \lfloor v \rfloor$, and $s = u - \lfloor u \rfloor$.

More descriptions of the image preprocessing are provided in [32–34].

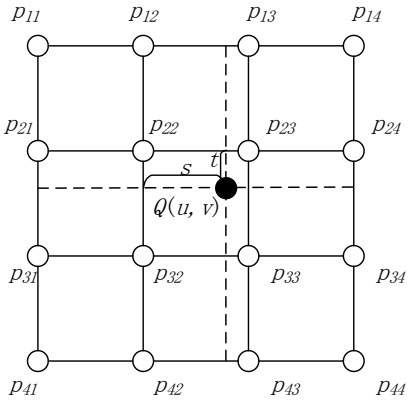



**Figure 2.** Bi-cubic interpolation method. $Q(u,v)$ is the output pixel gray value, $(u,v)$ is the sample position, $p_{11}$ to $p_{44}$ are the original sample pixel gray values, $t = v - \lfloor v \rfloor$, and $s = u - \lfloor u \rfloor$.

## 3. Parallel Accelerating Architecture

The preprocessing algorithm can be divided into two parts for hardware processing. The linear part, which contains a large number of rapid but repetitive computations, is the largest computational burden of on-board, real-time implementation. Because the linear part is a per-image pixel operation, communication with a mass storage resource must be considered. The nonlinear part consists of slower and more complex computations that determine the image quality.

### *3.1. Nonlinear Part Mapping Strategy*

#### 3.1.1. Hierarchical Decomposition Mapping Strategy

The nonlinear parts include the calculation of the RPCs and the coordinates. Although the method in the last section can complete the preprocessing of images, point-by-point calculation renders the hardware system complicated and time-consuming. To satisfy the needs of on-board processing, optimization of the algorithm is important. To balance the accuracy and complexity of the preprocessing algorithm, we split the image during the MCCDS and GC steps, as shown in Figure 3. $P_1P_2P_3P_4$ is the input image (Level 0 image or Level 1 image), $p_1p_2p_3p_4$ is the corresponding output image (Level 1 image or Level 2 image), and $P_1{}'P_2{}'P_3{}'P_4{}'$ is the image range for image storage. *abcd* is one of the image blocks after the output image is divided, and the corresponding image block in the input image is *ABCD*. For each image block, a corresponding set of RPCs exists. A smaller image block produces more accurate image correction and higher computational complexity. Therefore, the size of the image block is an important parameter.

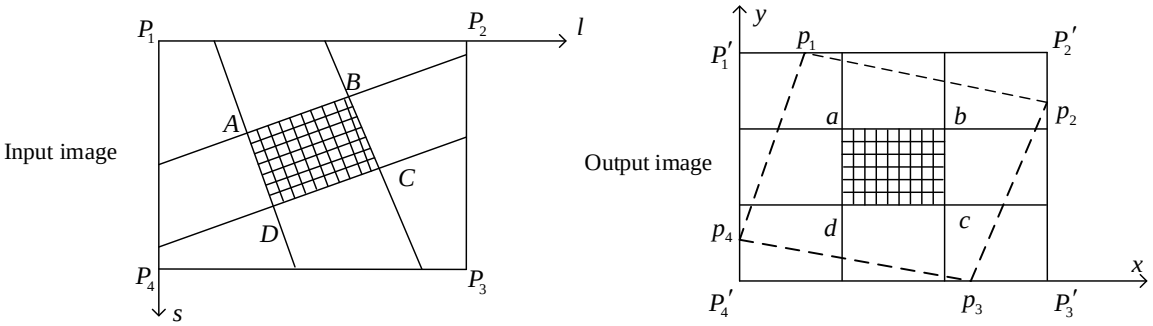



**Figure 3.** Image block correction method. $P_1P_2P_3P_4$ is the input image (Level 0 image or Level 1 image), $p_1p_2p_3p_4$ is the corresponding output image (Level 1 image or Level 2 image), $P_1{}'P_2{}'P_3{}'P_4{}'$ is the image range for image storage, *abcd* is one of the image blocks after the output image is divided, and *ABCD* is the corresponding image block in the input image.

In image block correction processing, the coordinate calculation in each image block involves many parameters, which enables a reduction in the number of computations.

The RFM in Section 2 can be simplified to the following formula when converting coordinates from a Level 1 image to a Level 0 image:

$$\begin{array}{l} s = s_0 + (t_0 + n \times t_1) \times s_{scale} \\ l = l_0 + (t_2 + n \times t_3) \times l_{scale} \end{array}, \tag{6}$$

where $n$ is the column number of one block in the Level 1 image. The remaining parameters are expressed as follows:

$$\begin{array}{ll} t_0 = m \times (s_1 + s_4) & t_1 = s_2 + m \times s_3 + s_5 \\ t_2 = m \times (l_1 + l_4) & t_3 = l_2 + m \times l_3 + l_5 \end{array}, \tag{7}$$

where $m$ is the row number of one block in the Level 1 image. The remaining parameters are expressed as

$$\begin{array}{ll} s_0 = (x_0 + a_0 \times x_1 + b_0 \times x_2 + a_0 \times b_0 \times x_3) \times s_{scale} + s_{off} & s_1 = \Delta a \times x_1 \\ s_2 = \Delta b \times x_2 s_3 = \Delta a \times \Delta b \times x_3 s_4 = a_0 \times \Delta b \times x_3 & s_5 = \Delta a \times b_0 \times x_3 \\ l_0 = (y_0 + a_0 \times y_1 + b_0 \times y_2 + a_0 \times b_0 \times y_3) \times l_{scale} + l_{off} & l_1 = \Delta a \times y_1 \\ l_2 = \Delta b \times y_2 l_3 = \Delta a \times \Delta b \times y_3 l_4 = a_0 \times \Delta b \times y_3 & l_5 = \Delta a \times b_0 \times y_3 \end{array}, \tag{8}$$

where $a_0$ and $b_0$ are the initial row number of the block and the initial column number of the block, respectively. $\Delta a$ and $\Delta b$ are the row step and column step, respectively; both are set to one in this algorithm. The remaining parameters are described in Section 2.

So, we divide these parameter calculations into three levels. The relationships among the different levels are shown in Figure 4. The first-level parameters only need to be calculated once during an image block. The second-level parameters need to be calculated once during each line of a block. The third-level parameters have to be calculated per pixel.

The three levels are also shown in Figure 5. The block phase, the line phase, and the point phase correspond to Expression (6), Expression (5), and Expression (4), respectively. The block phase is processed only when the initial parameters are provided for each image block. The results calculated by the block phase are sent to the line operation phase. After receiving the block calculation data, each line of the image block is calculated in the line phase, and the results are sent to the point phase. In the point phase, a point-by-point calculation occurs according to the received parameters. By optimizing the process, we can reduce the number of additions and multiplications when performing the point-by-point calculations, which reduces the use of many resources.

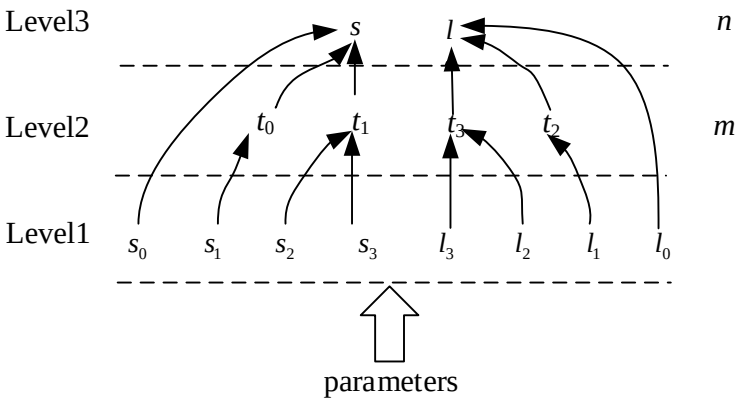



**Figure 4.** Hierarchical parameters calculation flow of Level 0 to Level 1.

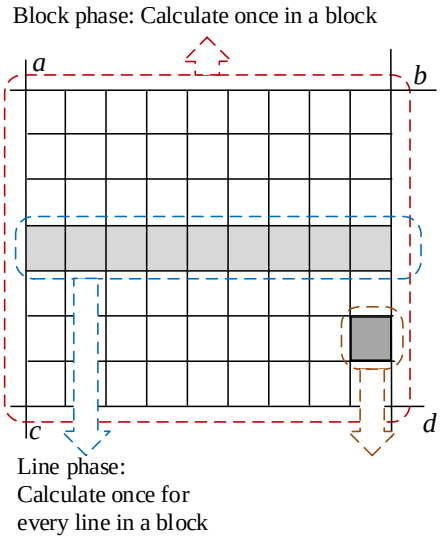



**Figure 5.** Different calculation phases in an image block.

The RFM in Section 2 that performs a coordinate transformation from a Level 2 image to a Level 1 image can be simplified. Because *lon* and *lat* are incremented by a fixed-step size and are redundant in the calculations, the process can be transformed into

$$\begin{array}{l} s = \frac{t_0+n\times s_2+h\times x_3}{t_1+n\times s_5+h\times x_6} \times s_{scale} + s_{off} \\ l = \frac{t_2+n\times l_2+h\times y_3}{t_3+n\times l_5+h\times y_6} \times l_{scale} + l_{off} \end{array}, \tag{9}$$

where $n$ is the number of steps in the longitude in the Level 2 image. The remaining parameters are expressed as

$$\begin{array}{ll} t_0 = s_0 + m \times s_1 & t_1 = s_3 + m \times s_4 \\ t_2 = l_0 + m \times l_1 & t_3 = l_3 + m \times l_4 \end{array}, \tag{10}$$

where $m$ is the number of steps in the latitude in the Level 2 image. The remaining variables are described in the following formula:

$$\begin{array}{lll} s_0 = x_0 + lon_0 \times x_1 + lat_0 \times x_2 & s_1 = \Delta lon \times x_1 & s_2 = \Delta lat \times x_2 \\ s_3 = lon_0 \times x_4 + lat_0 \times x_5 + 1 & s_4 = \Delta lon \times x_4 & s_5 = \Delta lat \times x_5 \\ l_0 = y_0 + lon_0 \times y_1 + lat_0 \times y_2 & l_1 = \Delta lon \times y_1 & l_2 = \Delta lat \times y_2 \\ l_3 = lon_0 \times y_4 + lat_0 \times y_5 + 1 & l_4 = \Delta lon \times y_4 & l_5 = \Delta lat \times y_5 \end{array}, \tag{11}$$

where $\Delta lon$ and $\Delta lat$ are the steps in the latitude and the longitude, respectively. For the calculation order of each parameter, we divide the coordinate transformation process into three phases, as shown in Figure 5, with the same order employed for the Level 1 image to the Level 0 image. The block phase, the line phase, and the point phase correspond to Expression (9), Expression (8), and Expression (7), respectively.

After optimization, we can obtain the new preprocessing chain, as shown in Figure 6, in which the parallelograms represent data and the rectangles represent the processing phases. The image data flow is shown in solid red lines, and the attitude and ancillary data flows are shown in solid blue lines. The data from the camera is separated into the image and the auxiliary data. The auxiliary data are used to calculate the RPCs of the Level 1 image and the Level 2 image. After processing the raw images using the RRC, the images are divided into blocks. Each image block undergoes the block phase, line phase, point phase, and resampling processing based on the respective RPCs. After this processing, we obtain the Level 2 image.

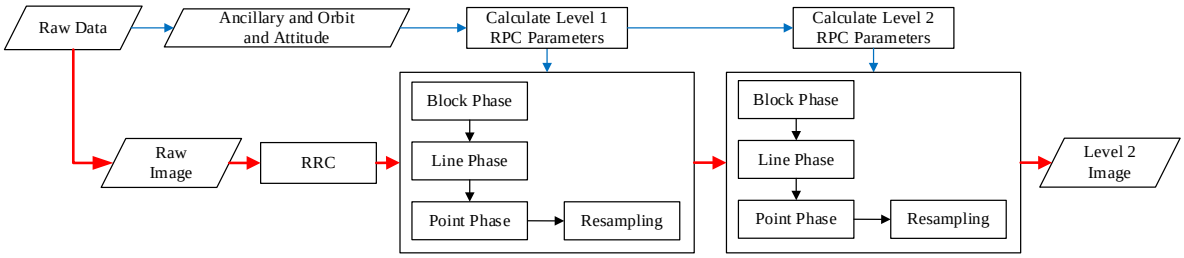



**Figure 6.** Preprocessing system work flow. The parallelograms represent data, and the rectangles represent the processing phases. The image data flow is shown in solid red lines, and the attitude and ancillary data flows are shown in solid blue lines.

#### 3.1.2. Complexity Analysis

After optimizing the calculation process, the computation times are reduced. For an image of 4096 × 4096 pixels, if we divide it into 32 × 32 blocks (each block is 128 × 128 pixels), then we can eliminate 133,892,672 additions and 166,692,864 multiplications. Detailed information is listed in Table 1.

**Table 1.** Computation times before and after optimization.

| Operation | Additions | Multiplications |
|---|---|---|
| *before* | 369,098,752 | 402,653,184 |
| *after* | 236,206,080 | 235,960,320 |

### *3.2. Linear Part Mapping Strategy*

The linear part primarily includes pixel grey calculations and data access. These operations need to calculate the grey values of each block of image. To satisfy the on-board processing needs, we needed to improve the efficiency of data access and calculation.

#### 3.2.1. Data Access Pattern

The principle of the mapping storage method is to optimize and balance the line and block data access rate. The MCCDS and GC must adopt the block correction method. If an image is stored in a normal sequence in the dual data rate (DDR) synchronous dynamic random-access memory (SDRAM), then the read and write processes will involve cross-banking, which lowers the efficiency. We needed to design a high-efficiency pattern based on the row-major format, which is a common method for storing multidimensional arrays. Therefore, we designed a form of mapping memory locations according to the image block, as shown in Figure 7, in which a single image block is mapped, as shown in Figure 8. The image is divided into $m \times n$ blocks, and each block has 64 × 64 pixels. The image is stored in the DDR with $m \times n$ rows, and each row stores the data of one image block. In the data read process, reading each image block corresponds to reading a row in the DDR. Thus, we can achieve the maximum efficiency for reading and writing data.

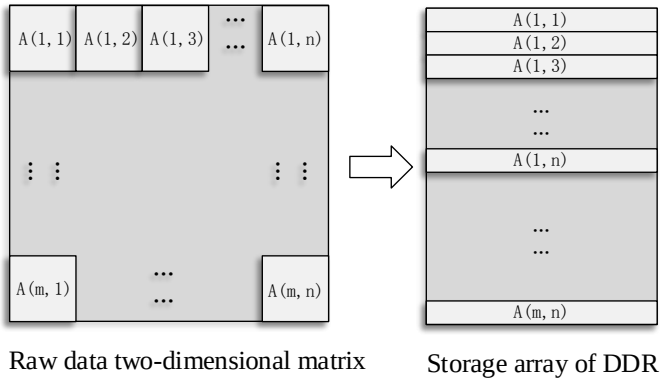



**Figure 7.** Block mapping method. DDR: dual data rate.

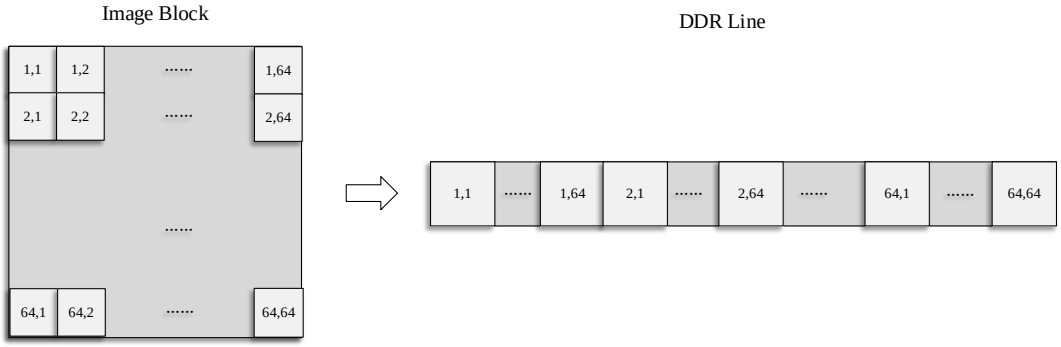



**Figure 8.** Single image block mapping method.

During reading and writing, the decoding must be performed according to the corresponding address. Because each operation is performed according to the image block, the decoding is divided into two steps. First, the image coordinates are mapped to the image block number. Second, the image block number is mapped to the DDR address. The relationship between the image coordinates $(x, y)$ and the image block coordinates $(m, n)$ is as follows:

$$\begin{cases} m = \lfloor x/64 \rfloor \\ n = \lfloor y/64 \rfloor \end{cases}. \tag{12}$$

The relationship between the image block coordinates $(m, n)$ and the row number $s$ of a single bank in the DDR is

$$s = m \times N + n. \tag{13}$$

When external data are written into the DDR, the data controller transforms the image according to the above method. In the subsequent image preprocessing process, the data controller only needs to read and write data using an image block according to the coordinate relationship. The inverse transformation of the image data only has to be done once in the final output process.

### 3.2.2. Parallel Processing Data Access

To realize parallel processing, we needed to analyze the data processing procedure. The image blocks can be sequentially read and written during the RRC process. When performing the MCCDS and GC processes, it is necessary to calculate the coordinates using the RFM and then perform resampling calculations according to the image coordinates. Due to the irregularity of coordinate transformation, it is difficult to predict the pixel positions required for each participating operation, which will affect the efficiency of the pipeline processing performance. Researchers [35] have proposed a parallel

computing strategy that can weaken the influence of the above characteristics, but that strategy is not suitable for implementation with a FPGA. When using a FPGA for data processing, a suitable rule for data reading and storage methods can make the processor perform better. Therefore, we designed rules for data reading and writing for the preprocessing algorithm.

During the MCCDS and GC steps, the input image grid position corresponding to each output image changes after the grid is divided. Therefore, the amount of data read from the DDR cannot be consistent every time. To solve this problem, we analyzed the positions of the grids. As shown in Figure 9, the output grids are primarily mapped to the input grids in four situations. Therefore, the maximum number of input grids corresponding to each output grid is nine (but not all blocks will be calculated). Therefore, we designed nine random-access memory (RAM) areas for reading data in this module; each area is 1 k $\times$ 64 bit and stores 64 $\times$ 64 pixels. We also designed two RAM areas for the output data, which can ensure that the pipeline writes the output image block. We designed 16 blocks instead of nine to read the data to ensure that the demand was still met in the event of a large deformation.

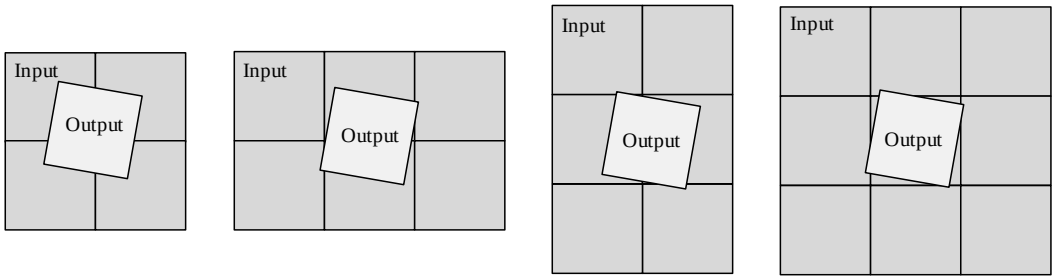



**Figure 9.** Positional relationship of output and input data.

Because coordinate transformations and resampling calculations require more time than the reading and writing of data, we can allow data reading and writing during the calculations to ensure the functioning of the pipeline. When the current computing module uses the RFM for coordinate transformation, the output of the previous sample block is written into the DDR, and the data required for the next sample block is read from the DDR. To reduce the amount of redundant data reads, only 2–3 input data blocks are read at one time. The input data block ranges must cover the output block corner. As shown in Figure 10, we read different data in different situations: label 1 in the gray-colored block is the input data that has been read, label 1 in the white-colored block is the currently calculated output image, and label 2 in the white-colored block is the next output image block to be calculated. The next output block coordinates will decide the next input image blocks, and the data that must be read is represented by label 2 in the gray-colored blocks.

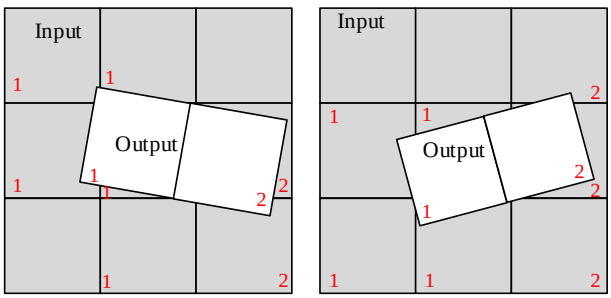



**Figure 10.** Image block read order. Label 1 in the gray-colored block is the input data that has been read, label 1 in the white-colored block is the currently calculated output image, label 2 in the white-colored block is the next output image block to be calculated, and label 2 in the gray-colored blocks is the data that must be read.

#### 3.2.3. Fixed-Point Calculation Design

When calculating the gray value of the image pixel, both fixed-point and floating-point data formats can be used. The floating-point format has high precision, but is resource-intensive, complex, and slower. The fixed-point format can be performed quickly and requires fewer resources, but the results are less accurate. The optical CCD image pixel gray scale is always 12 bits; therefore, the accuracy of calculation only has to be better than 12 bits. A fixed-point design for the calculation can thus be achieved without damaging effects. By performing fixed-point processing of the data in the calculation process, it is possible to optimize the use of resources and improve the calculation speed while ensuring data accuracy.

The RCC formula is

$$y_i = k_i * x_i + b_i, \tag{14}$$

where $x_i$ is the original pixel gray with a 12-bit integer and $y_i$ is the corrected pixel gray, which also must be an integer of 12 bits. To ensure a corrected pixel gray accuracy better than one gray level, both $k_i \times x_i$ and $b_i$ should have accuracies that are better than 0.1 gray level, which is a 4-bit fractional part. Therefore, $b_i$ is 16 bits, the first 12 bits are the integer, and the last 4 bits are the fractional part. Because $k_i \times x_i$ should have the same accuracy as $b_i$, $k_i$ is 28 bits, the first 12 bits are the integer, and the last 16 bits are the fractional part.

Because the results of the coordinate transformation in the MCCDS and GC are not an integer, it is necessary to perform a bi-cubic interpolation on the 16 points around the target pixel to obtain the gray value of the required point. Due to the pixel-by-pixel calculation and the large number of computations, a fixed-point design similar to the RRC is used. Table 2 lists the data structure in the fixed-point format that we have employed in this module. Those parameters have been described in Section 2.

**Table 2.** Data structures of fixed-point design.

| Variable | Sign | Integer | Fractional |
|---|---|---|---|
| $t, s$ | 0 | 0 | 32 |
| $t^2, s^2$ | 0 | 0 | 30 |
| $t^3, s^3$ | 0 | 0 | 28 |
| $a_1, a_2, a_3, a_4$ $b_1, b_2, b_3, b_4$ | 1 | 0 | 25 |
| $Q(u, v)$ | 0 | 12 | 0 |

## 4. Realization of the FPGA-DSP Accelerating Platform

To test and verify the functionality and performance of the proposed architecture, we developed a prototype system for preprocessing and conducted a parallel processing analysis.

This preprocessing system is designed based on a FPGA and a DSP co-processor. The main architectural modules of this preprocessing system are shown in Figure 11. The FPGA receives all the raw data, sends the image to the DDR for storage, and sends the remaining data to the DSP. Then, the FPGA processes the image data, whereas the DSP calculates the parameters of the two RFMs. Because the computations (such as sine and cosine functions) of the RFMs are complicated but utilize few data, the DSP is suitable for this purpose. All image data are processed by the FPGA, which ensures efficient parallelization of the algorithm.

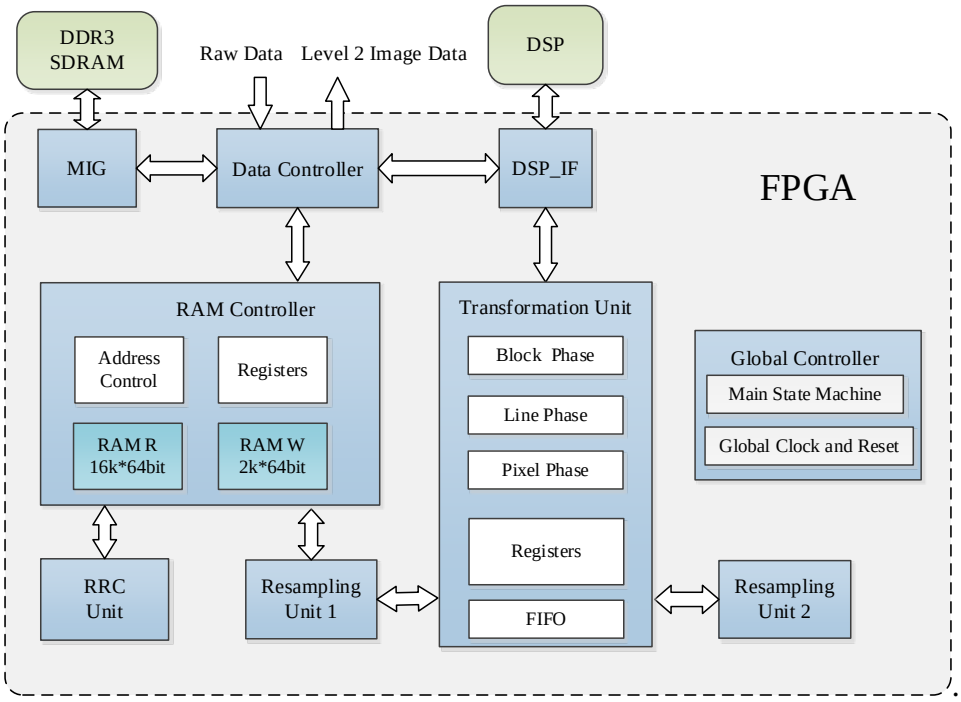



**Figure 11.** Preprocessing system architecture. FPGA: field-programmable gate array; DSP: digital signal processor; MIG: memory interface generator; RAM: random-access memory; FIFO: first-in-first-out; and SDRAM: synchronous dynamic random-access memory.

The data controller is responsible for receiving external data and achieving data interactions among the DDR, FPGA, and DSP. The memory interface generator (MIG) is used to control the DDR SDRAM. The RAM controller caches the data that are needed for the RRC unit and resampling unit 1. The RRC unit achieves the RRC process for the entire image. The transformation unit and resampling unit 1 realize the coordinate transformation and resampling processes of the MCCDS and the GC. Resampling unit 2 is applied when a more accurate elevation is required. The DSP_IF unit is used to exchange data between the FPGA and the DSP. We set the FPGA as the main controller in the proposed system. The FPGA will send an interrupt signal to change the work state of the DSP. After receiving the interrupt signal, the DSP will first read the register of the FPGA through external buses. Then, the DSP executes the corresponding process algorithm according to the register value. During this procedure, the DSP reads data from the RAM of the FPGA and then writes the results back to the RAM of the FPGA. When finishing this procedure, the DSP modifies the register value of the FPGA, and the FPGA will perform the specific operation according to the register value, such as reading and writing data from RAM or changing the state machine. The global controller contains the main state machine, which is responsible for the phase transition, global clock, and reset. Global information is propagated to all modules in the form of broadcasts.

The transformation unit performs coordinate transformations based on the RPCs that are sent by the DSP_IF and then sends the coordinate transformation results to resampling unit 1. This module is designed based on the optimization algorithm of Section 3. We designed the block phase, line phase, and point phase in this module. The block phase only needs to be run one time for each image block. The line phase runs once for each line of an image block.

The processing timeline is shown in Figure 12, which illustrates the working sequence of the different modules. For each procedure, after sending the data address by the DSP_IF or transformation unit, the data controller and MIG will read or store data for different purposes. Because the speed of reading is substantially higher than the speed of processing, the data controller and MIG consume less time. Because the RAM controller is designed for simultaneously reading and writing data, it can

perform different functions during each procedure. As shown in Figure 12, each processing unit (RRC unit, Transformation unit, and Resampling unit) starts working after obtaining data and does not stop until the procedure is ended. All units work on a pipeline and do not waste time waiting for other units.

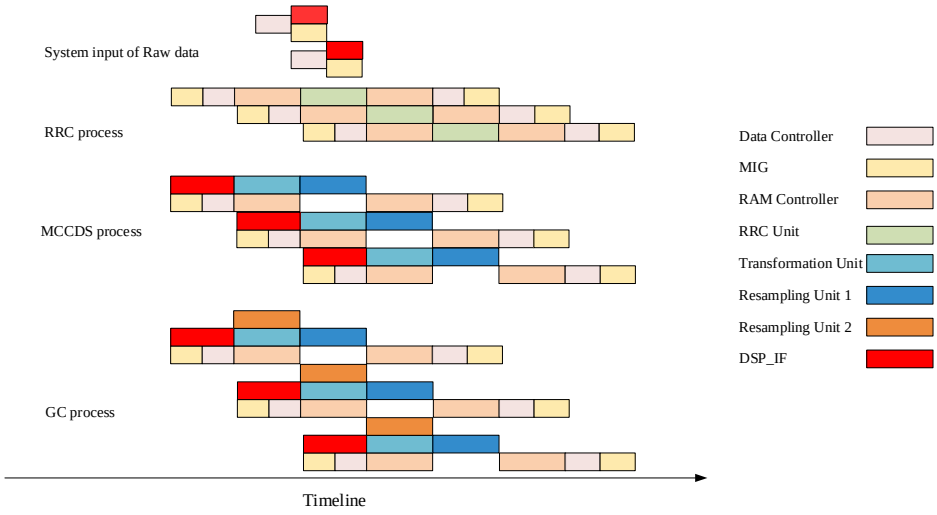



**Figure 12.** Processing timeline of the system.

## 5. Experimental Results

This section uses remote-sensing images to validate the preprocessing system. The verification in this section has two main goals. The first goal is to test and evaluate the effects of the system optimization methods. The second goal is to verify the function of the system and determine whether the system can realize the task of preprocessing. To address an on-board environment, the FPGA in this system was a Xilinx (San Jose, CA, United States) XC6VLX240T, and the DSP was a Texas Instruments (Dallas, TX, United States) TMS320C6701. We mainly used Verilog language to develop the system. In addition, we also used C language and a high-level synthesis tool to develop some computation units, such as the transformation unit and resampling unit. We employed synthetic and real data in our experiments. The synthetic data in this experiment consisted of three CCD push-scan images; the size of each CCD was 12,000 × 30,000 pixels. The real data in this experiment consisted of an image produced by the Gaofen-2 (GF-2) satellite. The image size was 29,200 × 27,620 pixels.

A photo of the hardware system that was employed for the preprocessing is shown in Figure 13. In this system, there were two parallel processing units. Each processing unit contained the FPGA and DSP processors and the independent DDR and rapid data transport channel. Thus, we could easily extend the processing ability for different data volumes.

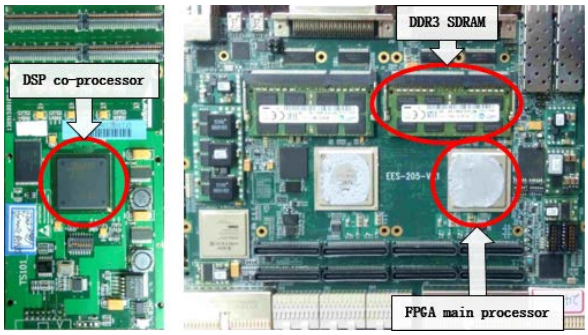



**Figure 13.** Photo of the hardware system.

### *5.1. Processing Performance*

This section tests the effectiveness of the algorithmic optimization approach that was employed. To evaluate the optimization of the algorithms and structures, we compared the effects of the calculation units (RRC unit, transformation unit, and resampling Unit) before and after optimization.

To ensure the comparison of identities, we designed the pipeline mode of each unit such that each unit expended the same amount of time for the same image data. The Flip-Flop (FF), Look-Up-Table (LUT), and DSP48 are the most important resources that determine the resource consumption of a FPGA. So, we verified the resource consumption before and after the calculation optimization. The comparison of the resource results is shown in Figure 14. After algorithm optimization and fixed-point calculation design, the consumption of all the calculation resources was lower. Therefore, the design of the hierarchical mapping and fixed-point calculations can reduce the use of resources more than the design with no optimization. Table 3 shows the FPGA resource occupation. The maximal frequency of this design is approximately 163 MHz.

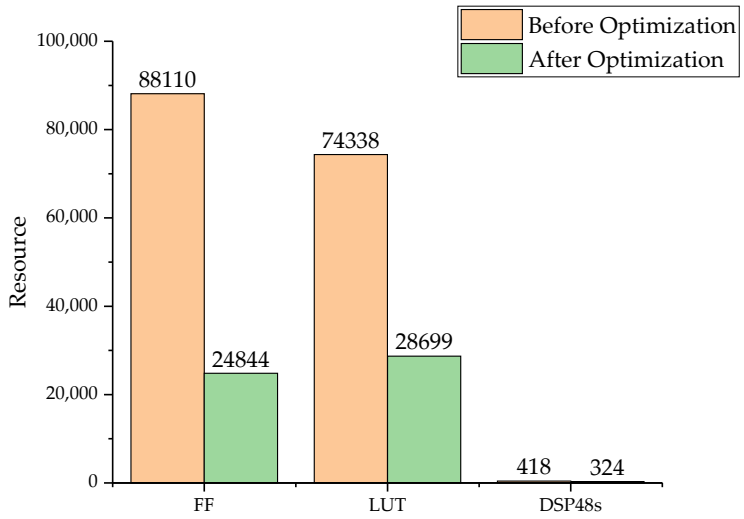



**Figure 14.** Calculation units' resource consumption before and after optimization.

To the best of our knowledge, similar hardware systems for remote image preprocessing have not been proposed; thus, we compare our system with central processing unit (CPU) based and GPU-based systems. The total system processing time for 2.01 GB of data is 11.6 s. For comparison purposes, we also processed 1.12 GB of data and recorded the time. The processing time of each processor in our system (FPGA-DSP co-processor) was compared with the processing times for other systems (CPU and GPU) [36]. Table 4 lists the processing speeds of the different systems. The processing time of an RRC in our system is more than the processing time of a GPU; however, the processing time of a GC is less than the processing time of a GPU. The FPGA design can reach higher speeds, because the FPGA can be more flexible in implementing pipelined and parallel process. Thus, the total processing speed is faster. Due to the relatively slow processing speed of the model parameters calculation by the DSP, the acceleration of the RRC process by increasing the resource usage and waiting for the parameters is unnecessary. Although the system based on a GPU can realize rapid development, it is not suitable for an on-board environment. The power consumption of our system is about 33 W, which includes two pairs of FPGAs and DSPs and the corresponding memory and Input/Output (I/O) devices. In contrast, the power consumption of the traditional GPU-based system is about 200 W. However, NVIDIA has released the embedded GPU, such as Jetson TX2, and the power consumption of an embedded GPU is nearly 8 W per processor. In order to process the same data volume, the power consumption of an embedded GPU system is close to the power consumption of our system. But these embedded GPUs cannot be adapted for an on-board processing system, which needs radiation tolerance. So, our system

is more suitable for an energy-constrained and high radiation space environment. Using the FPGA and the DSP enables greater flexibility in configuration and development at higher speeds. Therefore, the advantage of using the FPGA and DSP systems for on-board data preprocessing is irreplaceable.

**Table 3.** FPGA resources occupation (Xilinx xc6vlx240t). LUTs: Look-Up-Table; and FF: Flip-Flop

| Parameter | Used | Available |
|---|---|---|
| Number of slice registers | 41,061 | 301,440 |
| Number of slice LUTs | 39,072 | 150,720 |
| Number of fully used LUT-FF pairs | 20,674 | 59,459 |
| Number of block RAM/FIFO | 230 | 416 |
| Number of DSP48s | 324 | 768 |

**Table 4.** Processing times of different systems. CPU: central processing unit; and GPU: graphic processing unit.

| Platform Model | CPU (seconds) Intel Xeon E5650 CPU | GPU (seconds) Tesla M2050 GPU | Co-Processor (seconds) XC6VLX240T& TMS320C6701 |
|---|---|---|---|
| RRC | 3.64 | 0.23 | 0.67 |
| MCCDS | - | - | 1.67 |
| GC | 424.23 | 8.49 | 5.40 |

### 5.2. Real-Time Assessment

To assess the real-time performance, we present the following formula:

$$p = \frac{T_{in} + T_{pro} + T_{out}}{N * T_{in}}, \tag{15}$$

where *Tin* and *Tout* represent the time of raw-data input and the processing result output of the processing node, respectively. *Tpro* is the processing delay. *N* is the number of processing nodes. When *p* is less than one, the system can satisfy the real-time requirement. If *p* is larger than one, the system cannot satisfy the real-time requirement. Because one processing node can process an image, the speed of all data processing is positively related to the number of nodes. For the real-time, on-board task, if we only need to obtain a determined area, then one node is sufficient. If we need to process all data that are acquired, two solutions are available. The first solution is to establish additional processing nodes. The second solution is to establish additional memory when the processing time is less than the input time. Then, the system can process the first image when the second image is inputting into the memory. Our system employs the second solution to cope with the low-speed condition. For the GF-2 satellite, the data input time of the 2 GB image data is 1 s. Our system requires 0.89 s to process and output the same data. Thus, our system can satisfy the needs of real-time processing. For actual processing, only part of the image needs to be preprocessed and downlinked. Thus, the processing time will be substantially shorter. Therefore, our system can satisfy the needs of on-board, real-time processing.

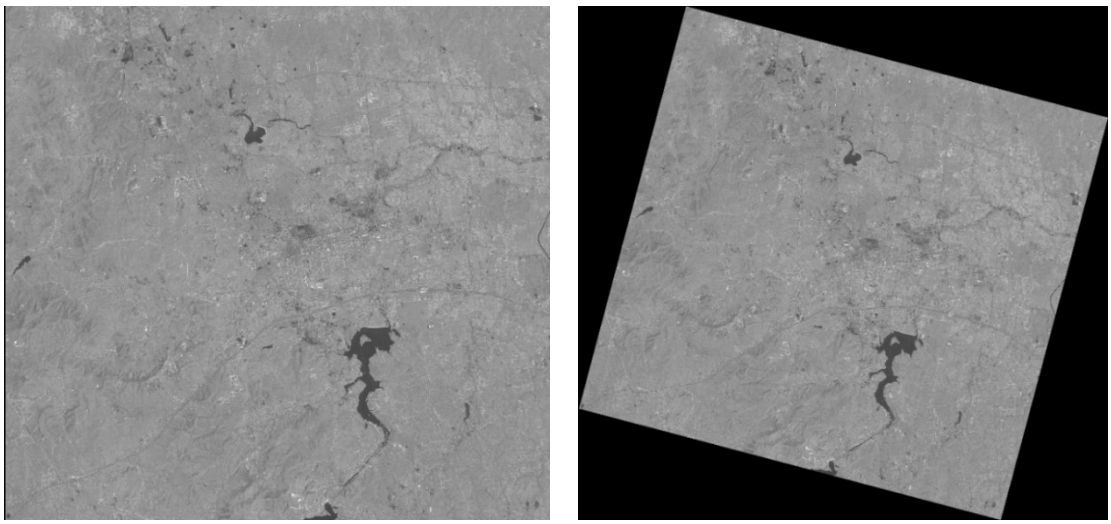
### 5.3. Correctness Results

To verify the correctness of our preprocessing system, we compared the results of this system with the results of the personal computer (PC) platform using the root-mean-square error (RMSE) of the output data of the two platforms as the evaluation criteria. The RMSE is expressed as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{w}\sum_{j=1}^{h}\left(DN_{ij}^{FPGA} - DN_{ij}^{PC}\right)^2}{w * h}}, \tag{16}$$

where $ND_{ij}{}^{FPGA}$ and $DN_{ij}{}^{PC}$ are the 16 bit integer values of the image pixels that are processed on the on-board platform and PC, respectively. *w* and *h* are the width and height of the Level 2 image.

Because the results of the CPU calculation are floating-point data and the results of the FPGA output are fixed-point data, we first compared the RMSE between the output of the FPGA and the floating-point data of the CPU. Then, we compared the RMSE between the FPGA output and the rounding of the CPU output. Table 5 lists the results. As we can see, the maximum RMSE is 0.2934 before the data are rounded. However, after rounding, the RMSE of both becomes zero, which means the corresponding resultant images are perfectly matched. However, the task of image preprocessing needs to obtain only integer-type image data to meet the requirements. Therefore, the fixed-point optimization method adopted by the system satisfies the precision requirements while improving the computational efficiency. Figure 15 provides an example of the input and output of the GC processing.



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

**Figure 15.** Example of the input and output of the GC processing: (**a**) raw image and (**b**) georeferenced image.

**Table 5.** Root-mean-square errors (RMSEs) between the CPU processors and the DSP/FPGA co-processors.

| Processor | RMSE (before Rounding) | RMSE (after Rounding) |
|---|---|---|
| RRC | 0.2886 | 0 |
| MCCDS | 0.2934 | 0 |
| GC | 0.2869 | 0 |

## 6. Conclusions

This paper presents a FPGA and DSP co-processing system for an optical remote-sensing image preprocessing algorithm. The design can be applied to the rapid responses required for the on-board processing of remote-sensing images. The main contributions of this paper are as follows.

First, we optimized a mapping methodology for the preprocessing algorithm. For the linear part, hierarchical coordinate transformation optimization, a block mapping design, and fixed-point calculation are proposed. The hierarchical optimization can reduce the complexity, the block mapping can prevent the problem of geometric deformation, and the fixed-point design can reduce the time consumption and simplify the design.

Second, we designed a parallel acceleration architecture for real-time requirements. An optical image preprocessing system that is based on a FPGA and DSP coprocessor was designed and implemented. Because our system is designed for on-board processing, we chose processors with a high radiation tolerance for space environments. The experimental results of this system demonstrate that our system has the potential for application on an on-board processor, for which the resources and power consumption are limited.

Although the current system can achieve the task of preprocessing, it requires the DSP to calculate the RPCs, which limits potential applications. In future research, a preprocessing algorithm based on a full FPGA design will be investigated. By using the FPGA to implement all the processes, the computational efficiency can be further improved and wider applications can also be realized.

**Author Contributions:** B.Q. and H.S. conceived of and designed the framework, performed the experiments and analyzed the data; H.C. and L.C. contributed to the hardware platform implementation; B.Q. wrote the paper; and Y.Z. and H.S. reviewed and revised the paper.

**Acknowledgments:** This work was supported by the National Natural Science Foundation of China under Grant 91438203, the Chang Jiang Scholars Program under Grant T2012122, and the Hundred Leading Talent Project of Beijing Science and Technology under Grant Z141101001514005.

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

## References

1. Goddijn-Murphy, L.; Peters, S.; van Sebille, E.; James, N.A.; Gibb, S. Concept for a hyperspectral remote sensing algorithm for floating marine macro plastics. *Mar. Pollut. Bull.* **2018**, *126*, 255–262. [CrossRef] [PubMed]
2. Percivall, G.S.; Alameh, N.S.; Caumont, H.; Moe, K.L.; Evans, J.D. Improving Disaster Management Using Earth Observations—GEOSS and CEOS Activities. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2013**, *6*, 1368–1375. [CrossRef]
3. Panteras, G.; Cervone, G. Enhancing the temporal resolution of satellite-based flood extent generation using crowdsourced data for disaster monitoring. *Int. J. Remote Sens.* **2018**, *39*, 1459–1474. [CrossRef]
4. Serge, A.; Berny, S.; Philippe, G.; Riza, F.A. INDESO project: Results from application of remote sensing and numerical models for the monitoring and management of Indonesia coasts and seas. *Mar. Pollut. Bull.* **2018**. [CrossRef] [PubMed]
5. Tralli, D.M.; Blom, R.G.; Zlotnicki, V.; Donnellan, A.; Evans, D.L. Satellite remote sensing of earthquake, volcano, flood, landslide and coastal inundation hazards. *ISPRS J. Photogramm. Remote Sens.* **2005**, *59*, 185–198. [CrossRef]
6. 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]
7. Chang, C.I.; Ren, H.; Chiang, S.S. Real-time processing algorithms for target detection and classification in hyperspectral imagery. *IEEE Trans. Geosci. Remote Sens.* **2001**, *39*, 760–768. [CrossRef]
8. Du, Q.A.; Nekovei, R. Fast real-time onboard processing of hyperspectral imagery for detection and classification. *J. Real-Time Image Process.* **2009**, *4*, 273–286. [CrossRef]
9. Subramanian, S.; Gat, N.; Ratcliff, A.; Eismann, M. Real-time Hyperspectral Data Compression Using Principal Components Transformation. In *Proceedings of the Aviris Earth Science & Applications Workshop*; NASA Technical Reports Server: Cleveland, OH, USA, 2000.
10. El-Araby, E.; El-Ghazawi, T.; Le Moigne, J.; Irish, R. Reconfigurable processing for satellite on-board automatic cloud cover assessment. *J. Real-Time Image Process.* **2009**, *4*, 245–259. [CrossRef]
11. Du, Q.; Nekovei, R. Implementation of real-time constrained linear discriminant analysis to remote sensing image classification. *Pattern Recognit.* **2005**, *38*, 459–471. [CrossRef]
12. Visser, S.J.; Dawood, A.S. Real-Time Natural Disasters Detection and Monitoring from Smart Earth Observation Satellite. *J. Aerosp. Eng.* **2004**, *17*, 10–19. [CrossRef]
13. Wang, M.; Zhu, Y.; Jin, S.; Pan, J.; Zhu, Q. Correction of ZY-3 image distortion caused by satellite jitter via virtual steady reimaging using attitude data. *ISPRS. J. Photogramm.* **2016**, *119*, 108–123. [CrossRef]
14. Guo, J.N.; Yu, J.; Zeng, Y.; Xu, J.; Pan, Z.Q.; Hou, M.H. Study on the relative radiometric correction of CBERS satellite CCD image. *Sci. China Ser. E* **2005**, *48*, 12–28. [CrossRef]
15. Toutin, T. Review article: Geometric processing of remote sensing images: Models, algorithms and methods. *Int. J. Remote Sens.* **2004**, *25*, 1893–1924. [CrossRef]
16. Li, C.; Gao, L.; Plaza, A.; Zhang, B. FPGA implementation of a maximum simplex volume algorithm for endmember extraction from remotely sensed hyperspectral images. *J. Real-Time Image Process.* **2017**. [CrossRef]

17. Kalomiros, J.A.; Lygouras, J. Design and evaluation of a hardware/software FPGA-based system for fast image processing. *Microprocess. Microsyst.* **2008**, *32*, 95–106. [CrossRef]
18. Halle, W.; Venus, H.; Skrbek, W. Thematic data processing on board the satellite BIRD. In *2000 International Symposium on Optical Science and Technology*; Digital Library: San Diego, CA, USA, 2000; Volume 4540, pp. 412–419. [CrossRef]
19. Botella, G.; Garcia, A.; Rodriguez-Alvarez, M.; Ros, E.; Meyer-Baese, U.; Molina, M.C. Robust Bioinspired Architecture for Optical-Flow Computation. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2010**, *18*, 616–629. [CrossRef]
20. Torti, E.; Fontanella, A.; Plaza, A. Parallel real-time virtual dimensionality estimation for hyperspectral images. *J. Real-Time Image Process.* **2017**. [CrossRef]
21. Gonzalez, C.; Lopez, S.; Mozos, D.; Sarmiento, R. A novel FPGA-based architecture for the estimation of the virtual dimensionality in remotely sensed hyperspectral images. *J. Real-Time Image Process.* **2015**. [CrossRef]
22. Li, B.; Shi, H.; Chen, L.; Yu, W.; Yang, C.; Xie, Y.; Bian, M.; Zhang, Q.; Pang, L. Real-Time Spaceborne Synthetic Aperture Radar Float-Point Imaging System Using Optimized Mapping Methodology and a Multi-Node Parallel Accelerating Technique. *Sensors (Basel)* **2018**, *18*. [CrossRef] [PubMed]
23. Yang, C.; Li, B.; Chen, L.; Wei, C.; Xie, Y.; Chen, H.; Yu, W. A Spaceborne Synthetic Aperture Radar Partial Fixed-Point Imaging System Using a Field- Programmable Gate Array-Application-Specific Integrated Circuit Hybrid Heterogeneous Parallel Acceleration Technique. *Sensors (Basel)* **2017**, *17*. [CrossRef] [PubMed]
24. Hauck, S. The roles of FPGAs in reprogrammable systems. *Proc. IEEE* **1998**, *86*, 615–638. [CrossRef]
25. Qiang, L.; Allinson, N.M. FPGA Implementation of Pipelined Architecture for Optical Imaging Distortion Correction. In Proceedings of the 2006 IEEE Workshop on Signal Processing Systems Design and Implementation, Banff, AB, Canada, 2–4 October 2006; pp. 182–187. [CrossRef]
26. Zemčík, P.; Přibyl, B.; Herout, A.; Seeman, M. Accelerated image resampling for geometry correction. *J. Real-Time Image Process.* **2011**, *8*, 369–377. [CrossRef]
27. Shevlin, F.P. Correction of geometric image distortion using FPGAs. In Proceedings of the Opto-Ireland 2002: Optical Metrology, Imaging, and Machine Vision, Galway, Ireland, 19 March 2003; Volume 4877, pp. 28–37.
28. Zhou, G.Q.; Zhang, R.T.; Liu, N.; Huang, J.J.; Zhou, X. On-Board Ortho-Rectification for Images Based on an FPGA. *Remote Sens.-Basel.* **2017**, *9*. [CrossRef]
29. Gehrke, S.; Beshah, B.T. Radiometric Normalization of Large Airborne Image Data Sets Acquired by Different Sensor Types. *ISPRS Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2016**, *XLI-B1*, 317–326. [CrossRef]
30. Nguyen, T. Optimal Ground Control Points for Geometric Correction Using Genetic Algorithm with Global Accuracy. *Eur. J. Remote Sens.* **2015**, *48*, 101–120. [CrossRef]
31. Maras, E.E. Improved Non-Parametric Geometric Corrections For Satellite Imagery Through Covariance Constraints. *J. Indian Soc. Remote Sens.* **2015**, *43*, 19–26. [CrossRef]
32. Schowengerdt, R.A. *Remote Sensing: Models and Methods for Image Processing*, 3rd ed.; Elsevier Academic Press Inc.: San Diego, CA, USA, 2007; pp. 1–515.
33. 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]
34. Tao, C.V.; Hu, Y. A Comprehensive study of the rational function model for photogrammetric processing. *Photogramm. Eng. Remote Sens.* **2001**, *67*, 1347–1357.
35. Zhou, H.F.; Yang, X.J.; Liu, H.Z.; Tang, Y. GPGC: A Grid-enabled parallel algorithm of geometric correction for remote-sensing applications. *Concurr. Comput.-Pract. Exp.* **2006**, *18*, 1775–1785. [CrossRef]
36. 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. Photogramm.* **2014**, *87*, 229–240. [CrossRef]