IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Traitement du Signal
Vol. 37, No. 5, October, 2020, pp. 745-752
Journal homepage: http://iieta.org/journals/ts

Design and Implementation of a Lossless Compression System for Hyperspectral Images

Qizhi Fang*, Yuxuan Liu, Lili Zhang

College of Electronic and Information Engineering, Shenyang Aerospace University, Shenyang 110136, China

Corresponding Author Email: arinc2006@sau.edu.cn

https://doi.org/10.18280/ts.370506

Received: 8 May 2020
Accepted: 19 September 2020

Keywords:
field programmable gate array (FPGA), hyperspectral image, lossless compression, forward prediction, full-pipeline construction

ABSTRACT

Despite its popularity, the hyperspectral image compression algorithm recommended by the Consultative Committee for Space Data Systems (CCSDS) faces a long delay of the feedback loop and complex computations in the modes of band sequential (BSQ) and band interleaved by line (BIL). After analyzing the features of the CCSDS algorithm, this paper proposes a forward prediction method based on the xc7k325tffg9000 field programmable gate array (FPGA) chip (Xilinx Inc.), and adjusts the calculation flow of the CCSDS algorithm, aiming to shorten the time delay in the feedback loop. In addition, full-pipeline construction was implemented on FPGA board to realize real-time processing of data, and dynamic configuration of image parameters. Through functional simulation and off-board test, it is learned that, for the speed-insensitive path, the optimized algorithm can realize the complex operations of the original algorithm with less hardware resources; for hyperspectral image data with an effective input bit width of 12bit, the proposed method can reach a maximum operating frequency of 103MHz, and the data throughput of 103M samples per second (1.237Gbps).

1. INTRODUCTION

The resolution of spectral images has been increasing, owing to the advancement of aerospace science. As a result, there is an exponential growth in the amount of information in optical remote sensing images [1, 2]. The massive amount of data overloads the limited transmission bandwidth and hardware resources of satellite-borne devices. Thus, it is particularly important to effectively compress the massive data.

Lossless compression is a suitable way to reduce the size of hyperspectral images [3]. For this reason, the Consultative Committee for Space Data Systems (CCSDS) has recommended a lossless compression standard for satellite-borne hyperspectral images (CCSDS123.0-B-1) [4]. With low computing complexity, this standard is very conducive to hardware implementation [5, 6], and internationally adopted for lossless compression of satellite-borne hyperspectral images [7].

The compression of hyperspectral images has long been a research hotspot. Many hyperspectral image compression algorithms have emerged, roughly falling into prediction method, transform method and vector quantization method [8, 9]. The following are some representative studies on multi- and hyper-spectral image compression. Valsesia and Magli [3] effectively controlled the onboard predictive coding of hyperspectral images, using multiple spectrum bands and Kalman filter. Wu and Memon [10] improved the context-based adaptive lossless image coding (CALIC) algorithm to compress hyperspectral images. To reduce complexity, Song et al. [11] introduced least squares filtering to the prediction process. Aiazzi et al. [12] proposed a multi-spectral prediction method based on a lookup table. Lin and Hwang [13] divided the prediction into two steps: computing the initial value of the prediction, and calculating the final predicted value. Mielikainen and Huang [14] implemented adaptive prediction length in linear prediction.

On field programmable gate array (FPGA), the CCSDS algorithm is mostly employed based on the mode of band interleaved by pixel (BIP). In this mode, the feedback loop is not necessarily related to the running speed of the hardware, and prone to a long calculation delay. But the large throughput of the mode can meet the needs of real-time processing. In both BIP mode and band sequential (BSQ) mode, the speed of hardware operation hinges on the calculation delay of the feedback loop, which is lengthened by the huge amount of internal calculations [15].

In fact, the traditional hardware structure can no longer satisfy the current demand of satellite communications with data throughput greater than 1Gbps [16]. BSQ and band interleaved by line (BIL) are two inevitable data formats of hyperspectral images. A huge amount of hardware resources will be consumed by processing BSQ and BIL images, whether the two formats are processed in parallel or converted into BIP before processing [17]. Considering the extremely limited resources of space-borne equipment, it is significant to develop a real-time compression technology for hyperspectral images in BSQ and BIL modes [18].

After detailed analysis on CCSDS algorithm, this paper proposes a forward prediction method based on the features of the feedback loop, which reduces the computing load inside the critical path. Then, the authors realized the full-pipeline construction of the hardware. Achieving a core data throughput of 1.237Gbps, the proposed method meets the needs of real-time compression, and supports dynamic configuration of the parameters.

2. ALGORITHM ANALYSIS

The CCSDS algorithm divides the columns, rows, and spectral segments of hyperspectral image data according to their coordinates (x, y, z) in a three-dimensional (3D) space. In the light of spectrum band, the image data are arranged in the z direction, forming a data cube. As shown in Figure 1, the data in BSQ, BIP, and BIL are processed in different orders: x, y, z for BSQ data, z, x, y for BIP data, and x, z, y for BIL data.

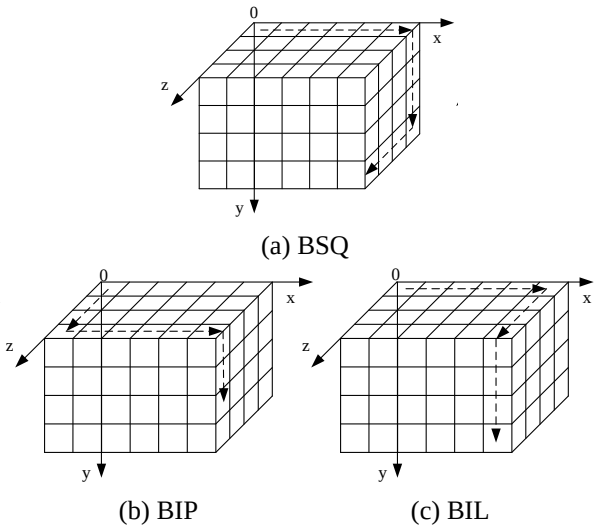

Figure 1. The processing orders of BSQ, BIP, and BIL images

Let N_x, N_y, and N_z be the number of image data in x, y, and z directions, respectively. That is, the value ranges of x, y, and z coordinates are [0, N_x-1], [0, N_y-1], and [0, N_z-1], respectively. In addition, it is defined that $t=y*N_x+x$. Then, each pixel can be described as:

$$S_{z,y,x} = S_z(t) \tag{1}$$

The CCSDS compression consists of two parts: prediction and encoding. Since the entire system only has one prediction module with a feedback structure, this paper only analyzes the prediction part (Figure 2). The current pixel $S_{z,y,x}$ needs to be processed, using its peripheral pixels and the corresponding pixels of the previous P spectrum bands.

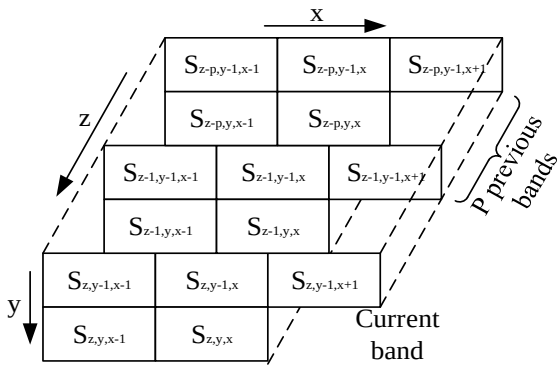

Figure 2. The 3D prediction model

The sum $\sigma_{z,y,x}$ of pixel weights in each spectral band can be computed pixel by pixel. The CCSDS algorithm defines two compression modes, namely, the proximity mode and the line mode. The calculation methods of $\sigma_{z,y,x}$ in the two modes can be respectively described as:

$$\sigma_{z,y,x} = \begin{cases} S_{z,y-1,x-1} + S_{z,y-1,x} + S_{z,y-1,x+1} + S_{z,y,x-1}, & y > 0, 0 < x < N_x - 1 \\ 4 * S_{z,y,x-1}, & y = 0, x > 0 \\ 2 * \left(S_{z,y-1,x} + S_{z,y-1,x+1}\right), & y > 0, x = 0 \\ S_{z,y,x-1} + S_{z,y-1,x-1} + 2 * S_{z,y-1,x}, & y > 0, x = N_x - 1 \end{cases} \tag{2}$$

$$\sigma_{zyx} = \begin{cases} 4 * S_{z,y-1,x}, & y > 0 \\ 4 * S_{z,y,x-1}, & y = 0, x > 0 \end{cases} \tag{3}$$

In each spectrum band, local differences in all directions are defined by the pixel positions, including central local differences $d_{z,y,x}$, N-direction local differences $d^N_{z,y,x}$, W-direction local differences $d^W_{z,y,x}$, and NW-direction local differences (Figure 3).

NW $S_{z,y-1,x-1}$	*N* $S_{z,y-1,x}$	$S_{z,y-1,x+1}$
W $S_{z,y,x-1}$	*central* $S_{z,y,x}$	

Figure 3. The calculation of local differences

The above local differences can be respectively computed by:

$$d_{z,y,x} = 4 * S_{z,y,x} - \sigma_{z,y,x}, \quad x + y \neq 0 \tag{4}$$

$$d^N_{z,y,x} = \begin{cases} 4 * S_{z,y-1,x} - \sigma_{z,y,x}, & y > 0 \\ 0, & y = 0 \end{cases} \tag{5}$$

$$d^W_{z,y,x} = \begin{cases} 4 * S_{z,y,x-1} - \sigma_{z,y,x}, & x > 0, y > 0 \\ 4 * S_{z,y-1,x} - \sigma_{z,y,x}, & x = 0, y > 0 \\ 0, & y = 0 \end{cases} \tag{6}$$

$$d^{NW}_{z,y,x} = \begin{cases} 4 * S_{z,y-1,x-1} - \sigma_{z,y,x}, & x > 0, y > 0 \\ 4 * S_{z,y-1,x} - \sigma_{z,y,x}, & x = 0, y > 0 \\ 0, & y = 0 \end{cases} \tag{7}$$

After obtaining the local differences of each spectrum band, the central local differences $\hat{d}_z(t)$ can be predicted based on the difference vector $U_z(t)$ and weight vector $W_z(t)$, under $P_z^* = \min\{z, P\}$:

$$\hat{d}_z(t) = W_z^T(t)U_z(t) = \begin{bmatrix} \omega_z^N(t) \\ \omega_z^W(t) \\ \omega_z^{NW}(t) \\ \omega_z^{(1)}(t) \\ \omega_z^{(2)}(t) \\ \vdots \\ \omega_z^{(P_z^*)}(t) \end{bmatrix}^T * \begin{bmatrix} d_z^N(t) \\ d_z^W(t) \\ d_z^{NW}(t) \\ d_{z-1}(t) \\ d_{z-2}(t) \\ \vdots \\ d_{z-P_z^*}(t) \end{bmatrix} \tag{8}$$

The initial value of $W_z(t)$ can be expressed as:

$$\omega_z^{(1)}(1) = \frac{7}{8}2^{\Omega},\ \omega_z^{(i)}(1) = \left\lfloor \frac{1}{8}\omega_z^{(i-1)}(1) \right\rfloor, \quad i = 2,3,\ldots,P_z^* \tag{9}$$

$$\omega_z^N(1) = \omega_z^W(1) = \omega_z^{NW}(1) = 0 \tag{10}$$

Then, $\tilde{S}(t)$ is derived from $\hat{d}_z(t)$ and $\sigma_{z,y,x}$, provided that its value falls in the specified range, and the error $e_z(t)$ is obtained by subtracting $\hat{S}(t)$ from the current pixel value $S_z(t)$. Next, the value of *sgn* can be determined by:

$$sgn = \begin{cases} 1, e_z(t) \geq 0 \\ -1, e_z(t) < 0 \end{cases} \tag{11}$$

Let $\rho(t)$ be the weight update factor. The weight vector $W_z(t)$ can be updated by:

$$W_z(t+1) = clip\left(W_z(t) + \left\lfloor \frac{1}{2}\left(sgn * 2^{-\rho(t)} * U_z(t) + 1\right)\right\rfloor, \{\omega_{min}, \omega_{max}\}\right) \tag{12}$$

The clip function can be defined as:

$$clip(x, \{x_{min}, x_{max}\}) = \begin{cases} x_{min}, & x < x_{min} \\ x, x_{min} \leq x \leq x_{max} \\ x_{max}, & x > x_{max} \end{cases} \tag{13}$$

The update of the weight vector $W_z(t)$ directly bears on the adaptability of the CCSDS algorithm.

If the predicted value is greater than the actual value, then the sgn is positive, and the weight vector needs to be properly increased; If the predicted value is smaller than the actual value, the weight vector needs to be properly decreased to reduce the predicted value; If the predicted value is equal to the actual value, the parameters are in line with image information, but the weight will still grow, making the compression unstable.

The sgn and $W_z(t+1)$ can be respectively expressed as:

$$sgn = \begin{cases} 1, e_z(t) > 0 \\ 0, e_z(t) = 0 \\ -1, e_z(t) < 0 \end{cases} \tag{14}$$

$$W_z(t+1) = clip\left(W_z(t) + \left\lfloor \frac{1}{2}\left(sgn * 2^{-\rho(t)} * U_z(t)\right)\right\rfloor, \{\omega_{min}, \omega_{max}\}\right) \tag{15}$$

By formulas (14) and (15), when the predicted value is equal to the actual value, i.e. the error is zero, the weight will remain unchanged, thereby ensuring the compression stability.

After prediction, the low-bit data of the image data are similar to noises. Direct encoding of these data will cause code expansion. The solution is to divide the encoder into two parts: K value selection and encoding. Any bit smaller or equal to K bit is not coded, while any bit greater than K bit is subject to Rice coding.

3. SYSTEM DESIGN

Under the scan mode of camera imaging, the data in all spectra were scanned line by line. To match the data output method of the camera, the BIL mode was adopted to process the image data. For the camera, the valid bit width, maximum width, and maximum number of spectrum bands are 12bit, 1280, and 160, respectively.

According to the CCSDS algorithm, the data of the previous rows and the previous P bands are required to process the data in the current row. The relevant data were cached in the BIL mode. If the height of each data entry is 32, then at least 2.46M of space is needed. The space demand exceeds the capacity of on-chip storage resources, which cannot be satisfied by general FPGA chips. To solve the problem, an off-chip memory DDR3 was selected for data caching.

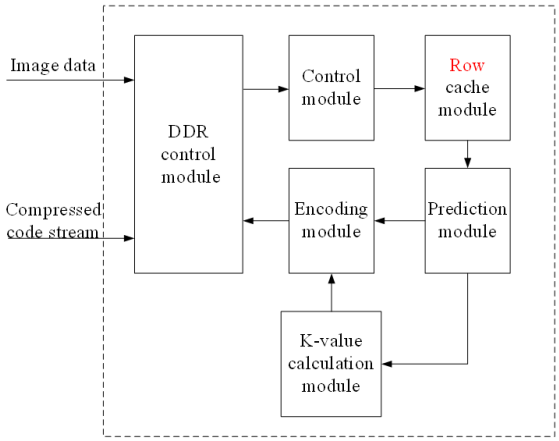

Figure 4. The architecture of the system

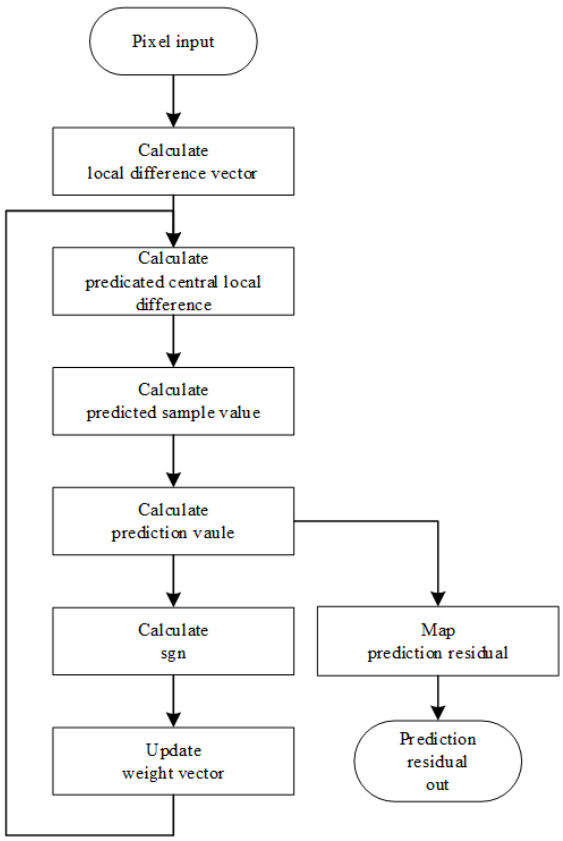

Figure 5. The workflow of the prediction module

As shown in Figure 4, the system caches the image data via the DDR control module, which dynamically configures the parameters and controls the working state of the compressor. The row cache module caches the input data, calculates the pixel values needed to predict the current pixel, and outputs the results to the prediction module. The prediction module predicts the pixel value, and calculates the prediction error. The k-value calculation module computes the parameter k required for encoding the current predicted value. Finally, the encoding module relies on the error and k-value to realize entropy encoding of the error. In the entire system, prediction module is the only module to have a feedback structure, that is, a full-pipeline construction can be directly implemented for all the other modules.

4. DESIGN AND IMPLEMENTATION OF FPGA

4.1 Limitations of traditional prediction modules

The prediction module calculates the difference vector $U_z(t)$ based on the values of the current pixel and its peripheral pixels, derives the predicted value $\hat{d}_z(t)$ of the center local difference from the weight vector $W_z(t)$, and computes predicted value $\hat{S}(t)$ and thus the error $e_z(t)$. After mapping, $e_z(t)$ is outputted to the encoding module, and used to calculate the value of sgn, for the update of the weight vector $W_z(t)$. The updated weight vector will be used in the next operation [19].

Figure 5 explains the workflow of the prediction module. In the same spectrum band, the prediction of the next pixel value depends on the updated value of the current pixel. Thus, there is a long feedback loop from the prediction $\hat{d}_z(t)$ of center local difference to the update of weight vector $W_z(t+1)$.

In BIL and BSQ modes, the data must be processed continuously in the same spectrum. To ensure full-pipeline construction, the calculation delay of the feedback loop was limited to one cycle. For the 12-bit data input from the camera, when $\rho(t)$=12, the following operations are required at the most to update the weight vector in the next cycle: one 24-bit multiplication, one 12-bit addition, two 24-bit additions, one 25-bit addition, and two 26-bit additions. Too many calculations in one cycle slows down the running speed of the hardware. What is worse, the computing speed of the traditional structure can only reach 66M samples per second. Although the running speed could be accelerated by adding registers to the feedback loop, the corresponding data throughput will drop exponentially due to the lack of full pipeline construction, failing to meet the actual requirements.

4.2 Implementation of the new prediction module

Under the improved CCSDS algorithm, there are only three cases for each update of weight vector $W_z(t)$:

$$W_z(t+1)=\begin{cases} clip\left(W_z(t)+\left\lfloor\frac{1}{2}\left(2^{-\rho(t)}*U_z(t)\right)\right\rfloor,\{\omega_{min},\omega_{max}\}\right), sgn=1 \\ clip\left(W_z(t)+\left\lfloor\frac{1}{2}\left(-2^{-\rho(t)}*U_z(t)\right)\right\rfloor,\{\omega_{min},\omega_{max}\}\right), sgn=-1 \\ W_z(t), \qquad sgn=0 \end{cases} \tag{16}$$

In formula (16), the calculation of $\left\lfloor\frac{1}{2}\left(\pm 2^{-\rho(t)}*U_z(t)\right)\right\rfloor$ is independent of the feedback loop. On this basis, this paper designs a forward prediction structure.

If the update of $W_z(t)$ is not completed in the current cycle, three possible results $W_{zp1}(t+1)$, $W_{zp2}(t+1)$, and $W_{zp3}(t+1)$ derived from $W_z(t+1)$ can be used simultaneously to predict the center local difference. In this way, $\hat{d}_{zp1}(t+1)$, $\hat{d}_{zp2}(t+1)$, and $\hat{d}_{zp3}(t+1)$ are obtained. Once $W_z(t)$ is updated in the next cycle, the correct $W_z(t+1)$ and $\hat{d}_z(t+1)$ can be selected after the corresponding sgn is obtained, ensuring the continuity of the calculation.

Under the new structure, the maximum number of cycles to limit the update of $W_z(t)$ is increased from one to two. Meanwhile, the new structure meets the condition of full-pipeline construction. Hence, an additional register can be added to the feedback loop. The workflow of the forward prediction of weight vector is illustrated in Figure 6.

Moreover, the prediction of the center local difference is split into two parts to balance the delay at both ends of the register induced by multiplications. First, the weight vector $W_z(t+1)$ is multiplied by the corresponding element in the difference vector $U_z(t+1)$. Then, the addition is performed to obtain $\hat{d}_z(t+1)$ after the data pass through the register.

To further reduce the computing load of the critical path in the feedback loop, the following variables $M_z(t)$, $A_z(t)$, $B_z(t)$ and $C_z(t)$ can be defined:

$$M_z(t)=(\hat{d}_z(t)-\left\lfloor\frac{\hat{d}_z(t)}{2^{\rho(t)+2}}\right\rfloor*4)+(\sigma_{zyx}-\left\lfloor\frac{\sigma_{zyx}}{4}\right\rfloor*4) \tag{17}$$

$$A_z(t)=\begin{cases} 4*S_z(t)-\left\lfloor\frac{\sigma_{zyx}}{4}\right\rfloor*4, M_z(t)<4 \\ 4*S_z(t)-\left\lfloor\frac{\sigma_{zyx}}{4}\right\rfloor*4-1, M_z(t)\geq 4 \end{cases} \tag{18}$$

$$B_z(t)=2*S_{max}+1-\sigma_{zyx} \tag{19}$$

$$C_z(t)=2*S_{min}-\sigma_{zyx} \tag{20}$$

In addition, sgn_1, sgn_2, and sgn_3 can be respectively defined as:

$$sgn_1=\begin{cases} 1, & S_z(t)>\left\lfloor\frac{S_{max}}{2}\right\rfloor \\ 0, & S_z(t)=\left\lfloor\frac{S_{max}}{2}\right\rfloor \\ -1, & S_z(t)<\left\lfloor\frac{S_{max}}{2}\right\rfloor \end{cases} \tag{21}$$

$$sgn_2=\begin{cases} 1, & \left\lfloor\frac{\hat{d}_z(t)}{2^{\rho(t)+2}}\right\rfloor*4<A_z(t) \\ 0, & \left\lfloor\frac{\hat{d}_z(t)}{2^{\rho(t)+2}}\right\rfloor*4=A_z(t) \\ -1, & \left\lfloor\frac{\hat{d}_z(t)}{2^{\rho(t)+2}}\right\rfloor*4>A_z(t) \end{cases} \tag{22}$$

$$sgn_3=\begin{cases} 1, S_z(t)>0 \\ 0, S_z(t)=0 \\ -1, S_z(t)<0 \end{cases} \tag{23}$$

Hence, sgn can be expressed as:

$$sgn = \begin{cases} sgn_1, & \left|\frac{\hat{d}_z(t)}{2^{\rho(t)}}\right| > B_z(t) \\ sgn_2, B_z(t) \le \left|\frac{\hat{d}_z(t)}{2^{\rho(t)}}\right| \le C_z(t) \\ sgn_3, & \left|\frac{\hat{d}_z(t)}{2^{\rho(t)}}\right| < C_z(t) \end{cases} \quad (24)$$

The calculations of $4 * S_z(t) - \left\lfloor\frac{\sigma_{zyx}}{4}\right\rfloor * 4$, $B_z(t)$, $C_z(t)$, sgn_1, and sgn_3, which are not related to $\hat{d}_z(t)$, can be performed outside the feedback loop through simple shifting, addition, and subtraction.

Through the above optimization, for input data with 12bit width, the 26bit and 12bit additions required to compute the sgn value in the critical path of the feedback loop are simplified to one 2bit addition during the prediction of $M_z(t)$. More importantly, the bid width required to compute $M_z(t)$ does not change, even if the bit width of the input data increases. In other words, the delay will not increase with the bit width of the input data. Therefore, the optimization greatly shortens the delay in the feedback loop. The sgn calculation structure is displayed in Figure 7, where delta, pcld, and ohm represent σ_{zyx}, $\hat{d}_z(t)$, and $\rho(t)$, respectively; the bit width is R+1.

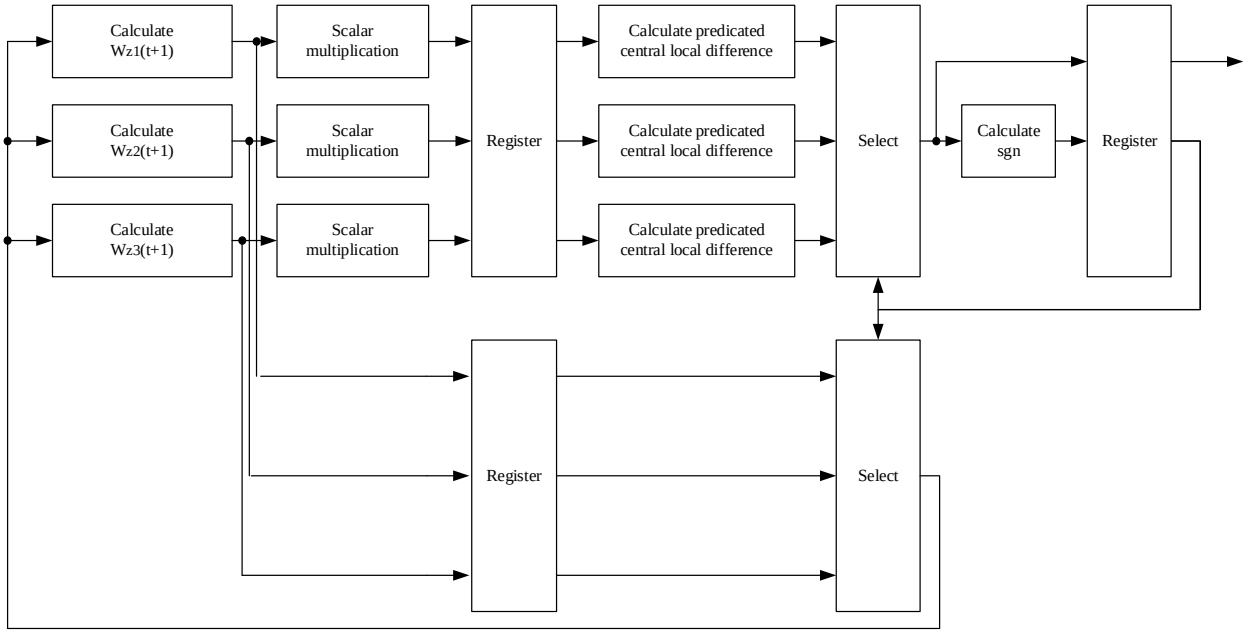

Figure 6. The workflow of forward prediction of weight vector

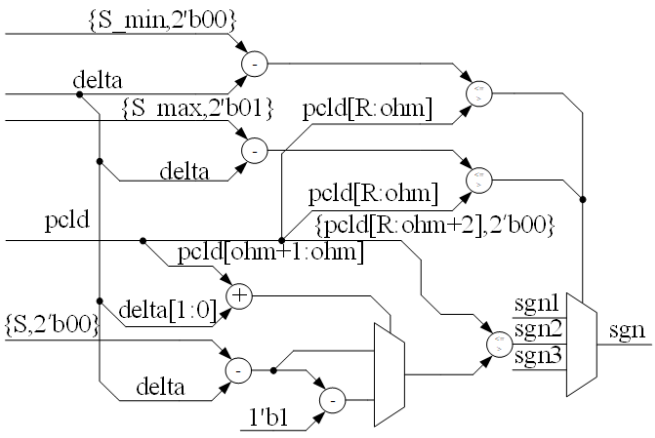

Figure 7. The structure of sgn calculation

4.3 Design of encoding module

The encoding module encompasses two sub-modules: K-value calculation module, and code stream splice module. The former mainly determines the k-value, while the latter generates a variable-length code, and splices the code to output a code stream of fixed 16-bit width.

4.3.1 K-value calculation module

The calculation of $k_z(t)$ value depends on the values of the counter $\Gamma(t)$ and the error accumulator $\sum_z(t)$. If $2 * \Gamma(t) > \sum_z(t) + \left\lfloor\frac{49}{2^7}\Gamma(t)\right\rfloor$, then $k_z(t)$=0; otherwise, $k_z(t)$ is the largest positive integer that satisfies:

$$k_z(t) \le D - 2 \quad (25)$$

$$2^{k_z(t)} * \Gamma(t) \le \sum_z(t) + \left\lfloor\frac{49}{2^7}\Gamma(t)\right\rfloor \quad (26)$$

where, D is the bit width of the input data.

If the bit width D equals 16, the determination of $k_z(t)$ requires up to 14 comparisons. Suppose only one comparison is completed per cycle. It takes 14 cycles to obtain the final result. The long delay wastes a lot of time. If all data are compared in one cycle, the relatively high bit width of $\sum_z(t)$ and the fan-out effect will make it impossible to finish the comparisons in time.

After comprehensive consideration, dichotomy was adopted to optimize the calculation of $k_z(t)$. Let (n, m] be the value range of each $k_z(t)$ value. Then, $k_z(t) = \left\lfloor\frac{n+m}{2}\right\rfloor$ was substituted into formula (26). If the condition is satisfied, then

$m = \left\lfloor \frac{n+m}{2} \right\rfloor$; otherwise, $n = \left\lfloor \frac{n+m}{2} \right\rfloor$. This process is repeated iteratively until $k_z(t)$ is determined.

After the optimization, when the bit width D equals 16, the value of $k_z(t)$ can be identified within 4 judgements. Each time, $\sum_z(t)$ will only fan out to two areas. This means the optimization manages to shorten the delay, and ease hardware implementation.

To further save resources, the step size was set to 49, and the counter $\Gamma'(t)$ was made equivalent to $\Gamma(t)$. Then, formula (26) can be rewritten as:

$$2^{k_z(t)} * \Gamma(t) \le \sum_z(t) + \left\lfloor \frac{\Gamma'(t)}{2^7} \right\rfloor \tag{27}$$

With the help of a counter $\Gamma'(t)$, the multiplication between $\Gamma(t)$ and 49 in formula (17) was converted into the addition of 49 at each accumulation of $\Gamma'(t)$. In this way, all the calculations in formula (18) can be realized through simple addition and shifting.

The hardware structure of single dichotomy judgment at $k_z(t)=k$ is explained in Figure 8, where acc, cnt49, and pcnt represent $\sum_z(t)$, $\Gamma(t)$, and $\Gamma'(t)$, respectively.

Next, the single dichotomy judgements with different k values were cascaded into the partial block diagram for $k_z(t)$ calculation (Figure 9).

4.3.2 Code stream splice module

Figure 10 presents the block diagram of the code stream splice module. If the coding is performed right after the K-value calculation, many consecutive zeros will appear in the code stream, dragging down the compression efficiency. To solve the problem, a comparator was added to the module. The comparator compares the size between the highest bit to the K bit of the mapping values and the threshold of 64 bits. Then, only the mapping values equal to or below the threshold were encoded. Those above the threshold were not encoded, but written as raw data.

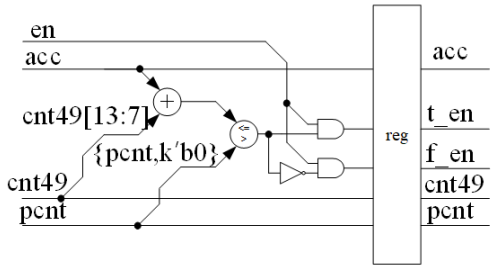

Figure 8. The hardware structure of a single dichotomy judgment

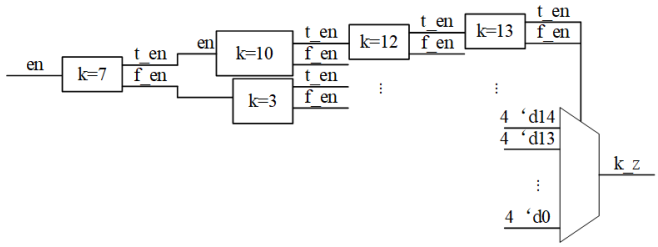

Figure 9. The partial block diagram for $k_z(t)$ calculation

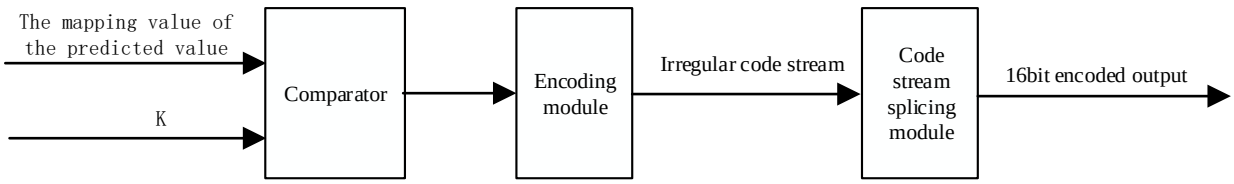

Figure 10. The block diagram of code stream splice module

5. PERFORMANCE VERIFICATION

The proposed hyperspectral image compression system was verified through functional simulation and off-board test. The environment and excitation files of the functional simulation were built on ModelSim. The off-board test was conducted on the xc7k325tffg900 chip of Xilinx, Inc., using Verilog language and Vivado 2020.1.

In the off-board test, the correct program verified by functional simulation was written to the FPGA board, and hyperspectral image data were imported to the board via the host computer. Since the transmission speed of the serial port is below 103MHz, the data received by the serial port were stored in a buffer zone. Every 32 rows of the received data were outputted to the compressor module at a working frequency of 103MHz. After compression, the code stream of the compressed image data was fed back to the host computer via the serial port. Then, the host computer compares the feedback with the compression result of the software. If the two are consistent, then the compressor must have been running normally on the hardware.

The test data were selected from the actual images shot by the camera. Dozens of G data were compiled into continuous data of different widths and spectral bands. In addition, different test data were employed to test the boundary conditions, such that the data do not overflow. In addition to comparing against the software compression result, the hardware compression result was decompressed and contrasted with the original data to verify the correctness of the system.

The test data are of various data formats with different number of columns. To ensure the reliability of the test, the number of rows was fixed at 32. The test results show that the compression ratio is about 1:2.

Table 1. The comparison of hardware compression results

Data mode (32 rows)	Compression ratio
79 bands 320 columns	1:2.2
79 bands 1,280 columns	1:2.16
10 bands 80 columns	1:1.85
10 bands 1,280 columns	1:1.84
105 bands 1,280 columns	1:2.09
126 bands 640 columns	1:2.25

Through the off-board test, it was learned that our system supports the maximum valid bit width of image data of 12bit, the maximum width of 1,280, and the maximum number of spectral bands of 160. The highest operating frequency was achieved at 103MHz by reducing the computing load in the critical path. In addition, the full-pipeline construction guaranteed a throughput rate of 103M samples per second. From the input of image data to the output of encoded data, the system only took 33 cycles, i.e. 320ns at 103MHz. Vivado analysis shows that the power consumption of the system stood at a low level of 0.465W.

Table 2 shows the resource occupancy rate of the compressor. Using double data rate (DDR) 3 to cache image data, the compressor occupied less than 3% of on-chip random-access memory (RAM). The falling computing load on the critical path pushes up the loads on the other paths. Compared with that (12) of other structures, our system occupied 63 digital signal processors (DSPs). Nevertheless, the occupancy rate of on-chip resources remained low at only 7.5%. Besides, our system improved the processing speed by over 56% with a small resource cost.

Table 2. The resource occupancy rate of compressor

Resource type	Occupied number/total	Resource occupancy rate/%
LUT	6,228/203,800	3.06
LUTRAM	7/64,000	0.01
FF	7,252/407,600	1.78
BRAM	10/445	2.25
DSP	63/840	7.50

Note: LUT, LUTRAM, FF, and DSP stand for lookup table, flip-flop, lookup table random-access memory, and block random-access memory, respectively.

6. CONCLUSIONS

Based on the BIL mode of CCSDS algorithm, this paper designs a prediction module and an encoding module, and implements them on FPGA board. The main innovations of the proposed system are as follows:

(1) To enhance system resilience, the block compression was adopted to process the data source. Through the processing, the compression error of a single pixel will not propagate into compression errors across the system.

(2) In the prediction module, forward prediction and the full-pipeline construction were introduced, such that the algorithm speed is no longer limited by the delay of the feedback loop in weight update.

(3) The optimized feedback loop can adapt to the BSQ mode, making it possible to compress BIL and BSQ data in real time. The proposed system is faster, more flexible, and more resource-efficient than the existing technology. In addition, the system parameters can be configured dynamically at a low resource cost, meeting the needs of satellite-borne missions.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (Grant No.: 61671310), and Program Foundation by Liaoning Province Education Administration (Grant No.: L2017044).

REFERENCES

[1] Wang, Y., Xing, L.N. (2015). Remote sensing satellite networking technology and remote sensing system: A survey. In 2015 12th IEEE International Conference on Electronic Measurement & Instruments (ICEMI), Qingdao, China, pp. 1251-1256. https://doi.org/10.1109/ICEMI.2015.7494508

[2] Thakar, L., Dutta, C., Sura, P.S., Udupa, S. (2016). Design & realization of multi mission data handling system for remote sensing satellite. In 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Jaipur, India, pp. 967-972. https://doi.org/10.1109/ICACCI.2016.7732170

[3] Valsesia, D., Magli, E. (2017). Fast and lightweight rate control for onboard predictive coding of hyperspectral images. IEEE Geoscience and Remote Sensing Letters, 14(3): 394-398. https://doi.org/10.1109/LGRS.2016.2644726

[4] Multispectral Hyperspectral Data Compression Working Group. (2011). Lossless Multispectral & Hyperspectral Image Compression CCSDS 123.0-R-1, ser. Red Book (draft). CCSDS, May.

[5] Santos, L., Gómez, A., Sarmiento, R. (2019). Implementation of CCSDS standards for lossless multispectral and hyperspectral satellite image compression. IEEE Transactions on Aerospace and Electronic Systems, 56(2): 1120-1138. https://doi.org/10.1109/TAES.2019.2929971

[6] Santos, L., Berrojo, L., Moreno, J., López, J.F., Sarmiento, R. (2015). Multispectral and hyperspectral lossless compressor for space applications (HyLoC): A low-complexity FPGA implementation of the CCSDS 123 standard. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 9(2): 757-770. https://doi.org/10.1109/JSTARS.2015.2497163

[7] Mamun, M., Jia, X., Ryan, M.J. (2013). Nonlinear elastic model for flexible prediction of remotely sensed multitemporal images. IEEE Geoscience and Remote Sensing Letters, 11(5): 1005-1009. https://doi.org/10.1109/LGRS.2013.2284358

[8] Conoscenti, M., Coppola, R., Magli, E. (2016). Constant SNR, rate control, and entropy coding for predictive lossy hyperspectral image compression. IEEE Transactions on Geoscience and Remote Sensing, 54(12): 7431-7441. https://doi.org/10.1109/TGRS.2016.2603998

[9] Huo, C., Zhang, R., Peng, T. (2009). Lossless compression of hyperspectral images based on searching optimal multibands for prediction. IEEE Geoscience and Remote Sensing Letters, 6(2): 339-343. https://doi.org/10.1109/LGRS.2008.2012135

[10] Wu, X., Memon, N. (2000). Context-based lossless interband compression-extending CALIC. IEEE Transactions on Image Processing, 9(6): 994-1001. https://doi.org/10.1109/83.846242

[11] Song, J., Zhang, Z., Chen, X. (2013). Lossless compression of hyperspectral imagery via RLS filter. Electronics Letters, 49(16): 992-994. https://doi.org/10.1049/el.2013.1315

[12] Aiazzi, B., Baronti, S., Alparone, L. (2009). Lossless compression of hyperspectral images using multiband

lookup tables. IEEE Signal Processing Letters, 16(6): 481-484. https://doi.org/10.1109/LSP.2009.2016834

[13] Lin, C.C., Hwang, Y.T. (2010). An efficient lossless compression scheme for hyperspectral images using two-stage prediction. IEEE Geoscience and Remote Sensing Letters, 7(3): 558-562. https://doi.org/10.1109/LGRS.2010.2041630

[14] Mielikainen, J., Huang, B. (2012). Lossless compression of hyperspectral images using clustered linear prediction with adaptive prediction length. IEEE Geoscience and Remote Sensing Letters, 9(6): 1118-1121. https://doi.org/10.1109/LGRS.2012.2191531

[15] Fjeldtvedt, J., Orlandić, M., Johansen, T.A. (2018). An efficient real-time FPGA implementation of the CCSDS-123 compression standard for hyperspectral images. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 11(10): 3841-3852. https://doi.org/10.1109/JSTARS.2018.2869697

[16] Gong, Z.M., Yang, J.X., Li, S.W. (2015). FPGA implementation of parallel frame synchronization system based on CCSDS. Microelectronics & Computer, 32(7): 82-85.

[17] Báscones, D., González, C., Mozos, D. (2017). Parallel implementation of the CCSDS 1.2.3 standard for hyperspectral lossless compression. Remote Sensing, 9(10): 973-990. Doi:10.3390/rs9100973

[18] Sun, J.W., Xue, C.B., Zhen, T., Zhang, Z.W. (2019). Sequential imagery lossless compression algorithm for space astronomical observation. Chinese Journal of Space Science, 39(6): 847-852. https://doi.org/10.11728/cjss2019.06.847

[19] Báscones, D., González, C., Mozos, D. (2020). An FPGA accelerator for real-time lossy compression of hyperspectral images. Remote Sensing, 12(16): 2563. https://doi.org/10.3390/RS12162563