


## Article

# FPGA-Based Implementation of Multidimensional Reconciliation Encoding in Quantum Key Distribution

Qing Lu <sup>1,2,†</sup>, Zhenguo Lu <sup>1,2,†</sup>, Hongzhao Yang <sup>1,2</sup>, Shenshen Yang <sup>3,4,\*</sup> and Yongmin Li <sup>1,2,\*</sup> 

- <sup>1</sup> State Key Laboratory of Quantum Optics and Quantum Optics Devices, Institute of Opto-Electronics, Shanxi University, Taiyuan 030006, China
- <sup>2</sup> Collaborative Innovation Center of Extreme Optics, Shanxi University, Taiyuan 030006, China
- <sup>3</sup> College of Physics and Information Engineering, Shanxi Normal University, Taiyuan 030031, China
- <sup>4</sup> Key Laboratory of Spectral Measurement and Analysis of Shanxi Province, Shanxi Normal University, Taiyuan 030031, China
- \* Correspondence: yangss@sxnu.edu.cn (S.Y.); yongmin@sxu.edu.cn (Y.L.)
- † These authors contributed equally to this work.

**Abstract:** We propose a multidimensional reconciliation encoding algorithm based on a field-programmable gate array (FPGA) with variable data throughput that enables quantum key distribution (QKD) systems to be adapted to different throughput requirements. Using the circulatory structure, data flow in the most complex pipeline operation in the same time interval, which enables the structural multiplexing of the algorithm. We handle the calculation and storage of eight-dimensional matrices cleverly to conserve resources and increase data processing speed. In order to obtain the syndrome more efficiently, we designed a simplified algorithm according to the characteristics of the FPGA and parity-check matrix, which omits the unnecessary operation of matrix multiplication. The simplified algorithm could adapt to different rates. We validated the feasibility and high speed of the algorithm by implementing the multidimensional reconciliation encoding algorithm on a Xilinx Virtex-7 FPGA. Our simulation results show that the maximum throughput could reach 4.88 M symbols/s.

**Keywords:** continuous variable quantum key distribution; multidimensional reconciliation; field-programmable gate array; encoding; variable throughput



**Citation:** Lu, Q.; Lu, Z.; Yang, H.; Yang, S.; Li, Y. FPGA-Based Implementation of Multidimensional Reconciliation Encoding in Quantum Key Distribution. *Entropy* **2023**, *25*, 80. <https://doi.org/10.3390/e25010080>

Academic Editor: Rosario Lo Franco

Received: 30 November 2022

Revised: 21 December 2022

Accepted: 29 December 2022

Published: 31 December 2022



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Quantum cryptography can provide information-theoretic security by combining one-time pad with quantum key distribution (QKD) [1,2] and has become an important branch and hotspot in the field of modern cryptography. QKD allows legitimate parties, Alice and Bob, to share secure keys even if the quantum channel is controlled by eavesdropper Eve. The fundamental theorems of quantum physics (no-cloning theorem, etc.) guarantee that quantum states transmitted through a quantum channel cannot be replicated accurately. Any eavesdropping behavior inevitably disturbs the quantum states on which the key information is encoded, which results in the increase in channel noises. Such noises can be monitored by the legitimate parties; therefore, any eavesdropping behavior can be discovered.

According to different carriers of the key, QKD can be divided into discrete-variable QKD (DVQKD) and continuous-variable QKD (CVQKD) [3–6]. DVQKD uses the polarization or phase degree of single photons to encode key information, which can realize long-distance key distribution using single-photon detection technology and post-selection. CVQKD employs the quadrature components of quantum states to encode key information; it is compatible with existing optical communication technology, and the key rate is high for short and medium distances. Important progress has been made recently [7–14]. However,

the data post-processing of CVQKD is relatively complex because it usually works at a low-SNR regime.

A typical CVQKD system usually consists of three parts: (1) preparation, distribution, and measurement of quantum states; (2) data sifting and parameter estimation; (3) data post-processing process [15]. The last part can be divided into two stages, information reconciliation and privacy amplification. In the information reconciliation stage, Alice and Bob obtain the same binary keys by correcting the data errors between their raw keys. After information reconciliation, Alice and Bob extract the final secret key using privacy amplification techniques.

At present, there are two schemes for data reconciliation in the CVQKD system, slice reconciliation [16] and multidimensional reconciliation [17], which are suitable for different SNR ranges. Slice reconciliation is suitable for relatively high SNRs, i.e., larger than 1 (short transmission distance) [18,19], and multidimensional reconciliation is suitable for low SNRs, from 0.01 to 1 (long transmission distance) [13,20]. In the case of Gaussian-modulated CVQKD, the multidimensional reconciliation algorithm provides a powerful encoding scheme for low-SNR scenarios and thus effectively extends the key distribution distance. In this way, the channel between Alice and Bob is converted into a virtual binary input additive white Gaussian noise (AWGN) channel; therefore, efficient binary codes can be employed.

In quantum information processing, error-correcting codes play a very critical role in protecting information from noise interference [21]. In CVQKD, an error correction code (ECC) with a large block is required to obtain high reconciliation efficiency, which is crucial for system performance. Because multiple iterations and low-density parity-check (LDPC) codes [22] are required in data reconciliation, the required computation complexity is high. On the other hand, the repetition rate of the CVQKD system increases rapidly. In this case, the speed of the corresponding data post-processing should match it; otherwise, the actual key rate would be reduced. The throughput of error correction algorithms based on general central processing units (CPUs) is very limited. A feasible solution to increase throughput is to utilize hardware-based acceleration, such as Graphics Processing Units (GPUs) [23–27] and field-programmable gate arrays (FPGAs) [15,28], which dramatically improve the computation speed. FPGAs are very attractive when designing prototypes and are applicable to small-scale production. They have high processing speed, parallelism, re-programmability, and low power consumption. The last one provides them with good integration abilities.

Multidimensional reconciliation algorithms have not been realized on an FPGA platform. In this paper, we achieve the encoding algorithm of multidimensional reconciliation with variable data throughput on an FPGA. To this end, we use flexible division of pipeline operation combined with an equal-interval circulatory structure, which can achieve different throughput. The calculation and storage of eight-dimensional matrices is cleverly designed to conserve resources and increase data processing speed. The high data processing speed of encoding can increase the real-time secret key rate of the CV-QKD system. Furthermore, a simplified algorithm without operation of matrix multiplication is exploited according to the characteristics of the FPGA and parity-check matrix to construct the syndrome efficiently.

The rest of this paper is organized as follows: In Section 2, we present the principles and steps of multidimensional reconciliation. In Section 3, we present the design and the detailed implementation of the multidimensional algorithm on an FPGA chip. The performance of the algorithm is analyzed in Section 4. In Section 5, we give a summary.

## 2. The Principle of Multidimensional Reconciliation

For Gaussian modulation CVQKD protocols, the shared raw keys of Alice and Bob are correlated Gaussian variables. For long transmission distances, the SNR is very low. In this case, the Gaussian variables have a small absolute value and are distributed around 0; thus, it is difficult to discriminate the sign and realize the encoding. The basic ideal of multidimensional

mensional reconciliation [17] is that one can perform a proper rotation before encoding the key in the sign of the coordinates; in this way, the small-absolute-value coordinates can be eliminated. The schematic diagram of multidimensional reconciliation is shown in Figure 1. The encoding ideal is as follows: Given  $y \in S^{n-1}$  ( $S^{n-1}$  represents a sphere), define a cube  $Q_y$  centered on 0 and containing  $y$  such that the prior distribution of  $y$  in  $Q_y$  is uniform. The description of  $Q_y$  can be achieved by describing the orthogonal transformation of the transformed canonical cube. Bob randomly selects vertices  $U$  ( $U \in S^{n-1}$ ) of the canonical cube and then provides Alice with an orthogonal transformation  $M(y, U)$  that maps  $y$  to  $U$ , satisfying  $M(y, U)y = U$ ; this transformation defines cube  $Q_y$ . Finally, Alice can recover  $U$  according to  $x$  and  $M(y, U)$ . The specific steps of multidimensional reconciliation are reported below.

Step 1. For security, Alice and Bob use the method of sequential combination to form a  $d$ -dimensional vector of  $d$  continuous variables in multidimensional reconciliation and obtain  $X = (x_1, x_2, \dots, x_d)$  and  $Y = (y_1, y_2, \dots, y_d)$ , where  $d$  denotes the dimension of multidimensional reconciliation. Here, we choose  $d = 8$  because its performance with a low SNR is better than that of other dimensions ( $d = 1, 2, 4$ ) [29]. Assuming that the quantum channel is linear, the relationship between Alice and Bob can be expressed as

$$y = lx + z, \quad (1)$$

and

$$x \sim \mathcal{N}(0, \sigma_x^2), \quad (2)$$

$$z \sim \mathcal{N}(0, \sigma_z^2). \quad (3)$$

where  $x$  and  $y$  represent the  $d$ -dimensional vectors of Alice and Bob, respectively; and  $z$  represents the noise of the quantum channel.

Step 2. In order to change the variable space from a non-uniform Gaussian distribution to a uniform Gaussian distribution, the  $d$ -dimensional state points need to be mapped from Euclidean space  $D^d$  to unit spherical space  $S^{d-1}$ . To this end, Alice and Bob normalize their Gaussian variables  $x$  and  $y$ .

$$x = \frac{x}{\|x\|}, \quad (4)$$

and

$$y = \frac{y}{\|y\|}, \quad (5)$$

where

$$\|x\| = \sqrt{\langle x, x \rangle}, \quad (6)$$

and

$$\|y\| = \sqrt{\langle y, y \rangle}. \quad (7)$$

Step 3. Bob generates a random binary sequence  $(b_1, b_2, \dots, b_d)$ . Then, a  $d$ -dimensional random vector  $u = (u_1, u_2, \dots, u_d)$  can be generated as

$$u = \left[ \frac{(-1)^{b_1}}{\sqrt{d}}, \frac{(-1)^{b_2}}{\sqrt{d}}, \dots, \frac{(-1)^{b_d}}{\sqrt{d}} \right]. \quad (8)$$

Step 4. Bob calculates rotation matrix  $M(y, u)$ . When  $d = 8$ , there is a (non-unique) family of 8 eight-dimensional orthogonal matrices  $(A_1, A_2, \dots, A_d)$  [17], where  $A_1$  is an  $8 \times 8$  identity matrix. Moreover, for  $i, j > 1$ , we have

$$\{A_i, A_j\} = -2\delta_{ij}A_1, \tag{9}$$

where  $\{A, B\} = AB + BA$ ,  $\delta_{ij}$  is the unit impulse function. Rotation matrix  $M(y, u)$  can be calculated from a family of  $d$ -dimensional orthogonal matrices  $(A_1, A_2, \dots, A_d)$  as

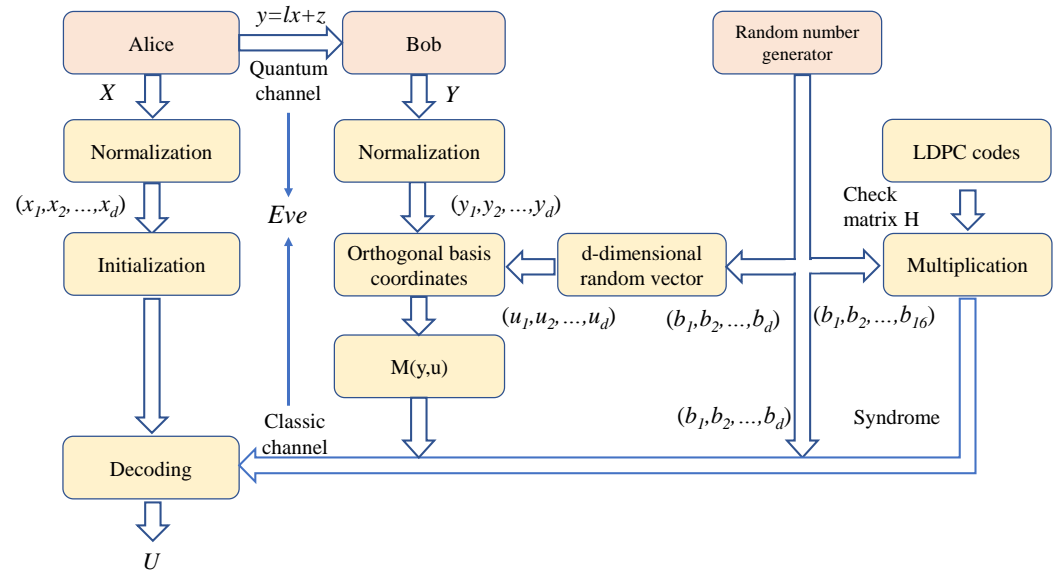
$$M(y, u) = \sum_{i=1}^d \alpha_i(y, u)A_i, \tag{10}$$

where,  $\alpha_i(y, u) = (A_i y \mid u)$  are the coordinates of  $u$  in basis  $(A_1 y, A_2 y, \dots, A_d y)$  and  $(A_1 y, A_2 y, \dots, A_d y)$  is a set of standard orthonormal bases for multidimensional space. The above proves that  $M(y, u)y = u$ .

Step 5. Bob calculates the syndrome using parity-check matrix  $H$  and random binary sequence  $(b_1, b_2, \dots, b_{16})$ .

$$I_{syn} = H \cdot (b_1, b_2, \dots, b_{16}), \tag{11}$$

where  $I_{syn}$  is the syndrome and  $(b_1, b_2, \dots, b_{16})$  is a 16-bit binary string. The width of the binary string is determined by the structure of LDPC codes.



**Figure 1.** Diagram of the multidimensional reconciliation scheme. Here,  $d$ -dimensional random vector  $u$  is generated from a random binary sequence  $(b_1, b_2, \dots, b_8)$ . Random binary sequence  $(b_1, b_2, \dots, b_{16})$  and parity-check matrix  $H$  are multiplied to obtain the syndrome.

### 3. FPGA Logic Design and Implementation of Multidimensional Reconciliation Encoding

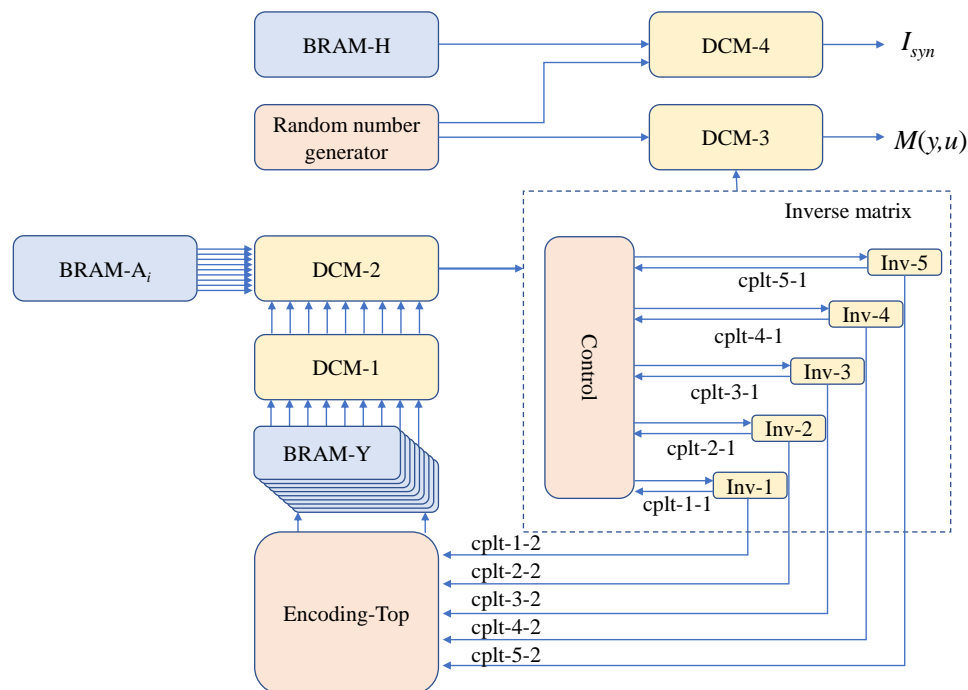
In the FPGA design and implementation, we used the pipeline operation to achieve high throughput with minimal FPGA resources and theoretically analyzed the throughput of our scheme. Then, we designed the storage mode of an eight-dimensional matrix according to the addressing mode of the FPGA and simplified the calculations for the orthogonal basis coordinates and matrix inversion of the eight-dimensional matrix. Finally, the syndrome was obtained through addressing and shifting.

### 3.1. Hardware Implementation Scheme

In order to implement the multidimensional reconciliation algorithm on an FPGA chip, we used VC709 Evaluation Kit manufactured by Xilinx, which included a Virtex-7 VX690T FPGA [30] and a user-programmable differential oscillator (range: 10 MHz–810 MHz). The Virtex-7 FPGA had a total of 433,200 LUTs, 866,400 Flip-Flops, and 3600 DSPs, as well as 52,920 Kb BRAMs.

To efficiently implement the multidimensional reconciliation algorithm, the relationship between hardware resources and throughput needs to be carefully investigated. In view of the available storage resources and editable logical resources, we adopted two approaches in different data processing parts, including the combination of data parallel and time division multiplexing, and the equal-interval circulatory structure-based pipeline operation. These methods can realize the exchange between area and speed. Because the encoding algorithm contains a large number of eight-dimensional matrix operations and the data are 32-bit single-precision floating-point numbers, this consumes a lot of editable logical resources. To overcome these issues, some modules were designed to be implemented with time division multiplexing and the circulatory structure with the same time interval. Due to the storage mode of the FPGA being a one-dimensional array, we converted the matrix form to fit the storage of the FPGA.

The logic design structure diagram of the multidimensional reconciliation encoding algorithm is shown in Figure 2. *Encoding-Top* is the top-level module of the encoding algorithm. *BRAM-Y* are eight block memory units that are configured as a true dual-port RAM with a width of 32 and depth of 8. *BRAM-A<sub>i</sub>* is a block memory unit that is configured as a true dual-port RAM with a width of 64 and depth of 8 and is used to store eight orthogonal matrices ( $A_1, A_2, \dots, A_8$ ). *BRAM-H* is a block memory unit that is configured as a single-port RAM with a width of 18 and depth of 500,000 and is employed to store the check matrix. *DCM* are four different data computing modules. Each *cplt* signal marks the end of the corresponding module operation. *Inv* are five matrix inversion modules with the circulatory structure operating in the same time interval.



**Figure 2.** FPGA design for multidimensional reconciliation encoding. The *Control* module switches between different matrix inversion modules according to signal  $cplt-j-1$  ( $j \in \{1, 2, \dots, 5\}$ ) and sends a startup flag to module *Inv-j*. Module *Encoding-Top* starts processing new data according to signal  $cplt-j-2$ .

### 3.1.1. Analysis of Data Throughput

The total throughput,  $T_h$ , is a key parameter to evaluate multidimensional reconciliation encoding and can be expressed as

$$T_h = \frac{d}{\frac{1}{f_s} T_n}, \tag{12}$$

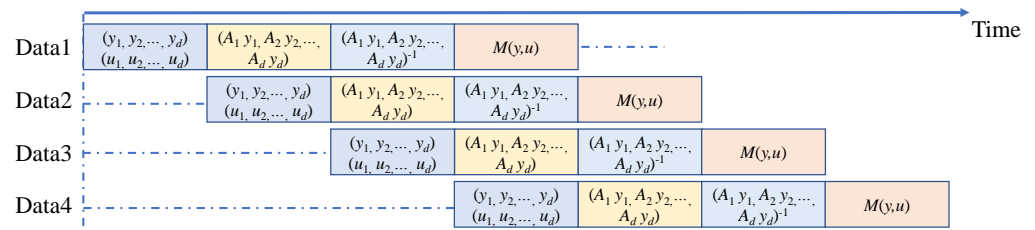
where  $f_s$  is the system clock frequency,  $T_n$  is the number of clock cycles required by the longest pipeline operation step, and  $d$  is the dimension of multidimensional reconciliation. It can be seen that the total throughput is closely related to the above three parameters. At a given clock frequency, the encoding performance is optimal when  $d = 8$ , so the throughput is inversely proportional to  $T_n$ . Therefore, with limited hardware resources, the pipeline task block loop should be optimized to achieve high throughput  $T_h$ .

The pipeline operation divides the combinatorial logic into a series of task modules and requires the addition of a first-level register before and after each task module. If the task module is too small, it consumes a large number of registers; each level of registers requires one clock cycle, so it also causes the operation time of the task module to increase. Therefore, we should reasonably divide the pipeline structure according to the specific throughput requirements.

### 3.1.2. Flexible Division of Pipeline Operation

The goal of the algorithm optimization is to achieve high throughput with minimal FPGA resources. As mentioned above, given the clock frequency and  $d = 8$ , the number of clock cycles,  $T_n$ , that the longest pipeline step requires is the key factor that affects throughput. We mainly used pipeline technology and a series-parallel structure to balance throughput and resource utilization of the FPGA. Pipeline technology cannot shorten the processing time of a single datum, but it can effectively shorten the processing time of the overall data. In contrast, the parallel structure can shorten the processing time of a single datum.

After carefully analyzing the calculation process of the encoding algorithm, we divided the pipeline operation of the encoding operation into four operation steps. Pipelining was used in the normalization of encoding and random number mapping operations, as well as in the syndrome generation module. The module with the longest running time is the calculation of  $(A_1 y_1, A_2 y_2, \dots, A_d y_d)^{-1}$ . Figure 3 shows the designed pipeline structure, in which Data2 starts to run the first step when Data1 has finished the first step. The circularity structure with the same time interval is employed for the module that computes  $(A_1 y_1, A_2 y_2, \dots, A_d y_d)^{-1}$  to suppress  $T_n$  according to the required throughput.



**Figure 3.** The pipeline structure of the encoding algorithm. The first step is to calculate  $(y_1, y_2, \dots, y_d)$  and  $(u_1, u_2, \dots, u_d)$  using  $Y$  and  $(b_1, b_2, \dots, b_d)$ . The second step is to build standard orthonormal basis  $(A_1 y_1, A_2 y_2, \dots, A_d y_d)$  for the multidimensional space. The third step is matrix inversion. The fourth step is to map  $u$  to standard orthonormal basis  $(A_1 y_1, A_2 y_2, \dots, A_d y_d)$  and obtain  $M(y, u)$ .

### 3.2. Eight-Dimensional Matrix Operation

Because the performance of the multidimensional reconciliation algorithm is optimal when  $d = 8$ , we only studied the eight-dimensional matrix operation. During the operation

process of the eight-dimensional matrix, such as addition, subtraction, multiplication, division, root, square, and inversion [31], the bit growth of data may occur. Although not every level of operation encounters bit growth, each bit growth instance leads to the doubling of the maximum value of the data. This may cause data overflow if fixed-point numbers are employed, making the results of the operation incorrect because the data are not accurate enough. In contrast, floating-point arithmetic has a large dynamic range, and because of its automatic scaling, the possibility of data overflow and mantissa loss can be eliminated. We used 32-bit single-precision floating-point numbers for the calculation of all basic units.

### 3.2.1. Storage of 8-Dimensional Matrices

According to the addressing mode of the FPGA, the  $8 \times 8$  matrix is extended to a one-dimensional array of length 64. In order to read and store quickly, we need to use 8 memory units, each with a depth of 8 and a width of 32. The first address of each memory unit stores the first row of the 8-dimensional matrix, and so on. Therefore, each write-and-read operation only takes 8 clock cycles.

### 3.2.2. Construction of Orthogonal Basis Coordinates

The orthogonal basis coordinates of  $u$  is  $(A_1y, A_2y, \dots, A_d y)$ , where  $(A_1, A_2, \dots, A_d)$  is a set of orthogonal matrices that can be computed by four  $2 \times 2$  matrices [17]. The calculation process is reported below.

The four basis matrices are given by

$$K_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, K_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

$$K_2 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, K_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$

and the orthogonal matrices can be calculated as

$$(A_1, A_2, \dots, A_8) = \{K_{000}, K_{332}, K_{320}, K_{312}, K_{200}, K_{102}, K_{123}, K_{121}\}, \tag{13}$$

where  $K_{i_1, \dots, i_l} = K_{i_1} \otimes \dots \otimes K_{i_l}$  represents the tensor product.

According to the characteristic of orthogonal matrix  $A_i$ , each row of it has only one non-zero element, and the absolute value of this element is 1. Therefore, the multiplication of a one-dimensional vector  $y$  with a matrix  $A_i$  can be converted to finding a non-zero element of an orthogonal matrix  $A_i$  and rearranging the  $y$  vector. In this way, we can eliminate many multiplication and addition operations and save a lot of LUT resources.

For example,

$$A_3 = \begin{pmatrix} 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \end{pmatrix},$$

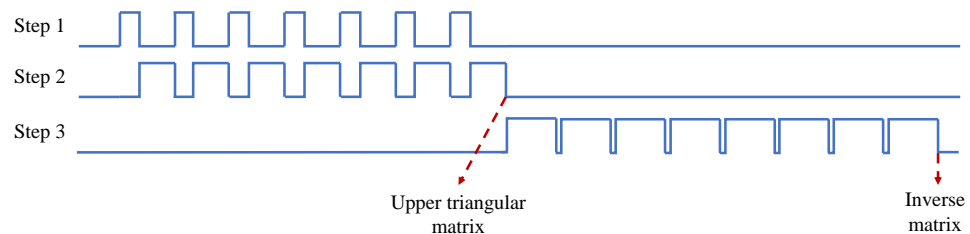
$$A_3 \times y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{pmatrix} \times \begin{pmatrix} 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} -y_3 \\ -y_4 \\ y_1 \\ y_2 \\ y_7 \\ y_8 \\ -y_5 \\ -y_6 \end{pmatrix}.$$

### 3.2.3. Inverse Matrix of the Eight-Dimensional Matrix

Because  $(A_1, A_2, \dots, A_d)$  is a group of eight orthogonal matrices,  $(A_1y_1, A_2y_2, \dots, A_dy_d)$  is invertible. The calculation of  $M(y, u)$  can be written as

$$M(y, u) = \sum_{i=0}^d \alpha_i(y, u)A_i = (A_1y_1, A_1y_1, \dots, A_dy_d)^{-1}u. \tag{14}$$

For the calculation of  $(A_1y_1, A_2y_2, \dots, A_dy_d)^{-1}$ , we used the Gaussian elimination method [32] to obtain the inverse matrix of the eight-dimensional matrix, which can be divided into three steps: (1) finding the maximum value of each row; (2) obtaining an upper triangular matrix using elementary row operations; (3) data normalization. Due to the calculation of the inverse matrix consuming a lot of LUTs and DSPs, we used time division multiplexing to balance the speed and area of the FPGA, as shown in Figure 4. Step 1 and Step 2 perform loop execution seven times to obtain an upper triangular matrix. Then, Step 3 performs loop execution eight times to obtain an inverse matrix. Step 1 sends the position of the maximum value for each row to Step 2, and Step 2 sends the matrix to Step 1 after elementary row operations.



**Figure 4.** Time division multiplexing of inverse matrix. Step 1 is finding the maximum value of each row. Step 2 is elementary row operations. Step 3 is data normalization.

In order to search for the maximum value of each row, eight comparators and an AND door are exploited. During the calculation process, the eight data in each row are simultaneously read from the same address. Because the comparator consumes only 87 LUTs, seven comparators are run concurrently and then the AND gate is used to decide the maximum value in each row. In this way, the logic delay of this combination circuit is minimized, which consists of the delay of the comparator of single-precision floating-point numbers and the delay of an AND gate.

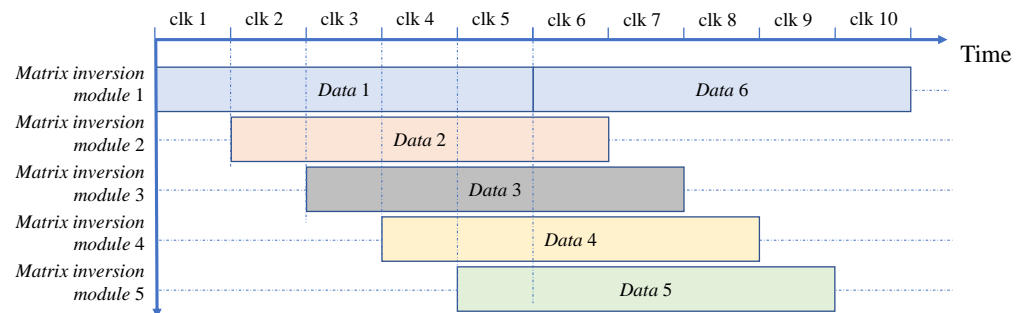
### 3.2.4. Multiplexing Structure of Matrix Inversion Module

The matrix inversion module cannot be subdivided into multi-level pipelines with the pipeline operation due to module multiplexing and nested loops. To improve the computation speed, we designed a circulatory structure with the same time interval, as shown in Figure 5.

Firstly,  $T_n$  is calculated based on the target throughput. The input data control module then assigns the input data stream to different matrix inversion modules simultaneously. Then, Data 2 performs the matrix inversion operation after  $T_n$  clock cycles relative to Data 1, and so on. Therefore, Data 1 and Data 6 are both calculated by Matrix inversion module 1.



Each data stream has a corresponding matrix inversion module with a time interval of  $T_n$ . The number of reused matrix inversion modules is  $\frac{T_{inv}}{T_n}$ , where  $T_{inv}$  is the time required for the inversion of an eight-dimensional matrix in serial operation.



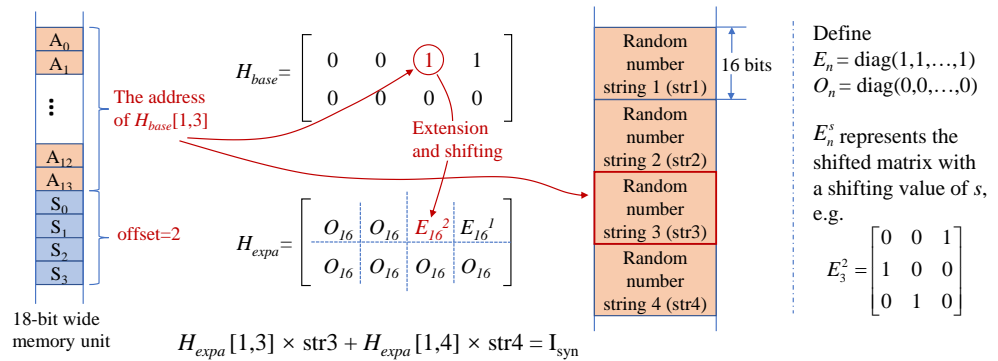
**Figure 5.** Circulatory structure with the same time interval.  $clk$  has the clock cycle of  $T_n$ . Each *Matrix inversion module*  $j$  is multiplexed.

### 3.3. Construction of the Syndrome

Due to the existence of channel loss and excess noise in quantum key distribution, there is inevitably some discrepancy when the two groups of Gaussian numbers of Alice and Bob are transformed into binary bits in the spherical space. To correct the bit errors, Bob can send a syndrome of his bit string to Alice. Here, the syndrome can be obtained by multiplying the check matrix with Bob's random bit string. The check matrix of an LDPC code [22] has very strong sparsity, that is, the "0" element accounts for most of the check matrix and the "1" element is very sparsely distributed. Considering the sparsity characteristic of an LDPC code, the shifting method is used to calculate the syndrome.

The storage of the entire check matrix requires a large amount of resources. In order to save memory resources, we store the check matrix by storing the positions of non-zero elements in each row of the matrix. The check matrix is extended in a quasi-cyclic manner [33]. Notice that the shifting operation is implemented after quasi-cyclic expansion, in which the non-zero element is extended to a  $16 \times 16$  identity matrix. So, we use an 18-bit-wide memory unit to save the basis matrix and the extended matrix. The first 14 bits ( $A_0$ – $A_{13}$ ) describe the positions of non-zero elements in the basis matrix, and the last four bits ( $S_0$ – $S_3$ ) describe the shifting of the extended matrix.

For the multiplication of the basis matrix and the input random bit string, the position of the non-zero elements of the basis matrix is used as the address to read the bit string, and the drifting value of the non-zero element is used to shift the bit string. The multiple shifted bit strings are accumulated to obtain the product of a row of the check matrix and the random bit string, that is, the syndrome, as shown in Figure 6. If the address of the  $N$ th non-zero element is smaller than the address of the  $(N-1)$ th non-zero element, that means that we start to compute the next syndrome, which is the product of the next row of the check matrix and the random bit string. This saves a lot of DSPs and LUTs and increases the running rate, because a lot of multiplication and addition operations are omitted.



**Figure 6.** Calculation procedure of the syndrome for a  $2 \times 4$  base matrix.  $H_{base}$  and  $H_{expa}$  denote the base matrix and the matrix extended in a quasi-cyclic manner.  $str1$ ,  $str2$ ,  $str3$ , and  $str4$  denote independent random bit strings.

#### 4. Results and Discussions

In this section, we show the implementation results of multidimensional reconciliation encoding on an FPGA chip (Xilinx Virtex-7 FPGA). The feasibility of the encoding algorithm was validated. When the pipeline step takes longer than  $T_n$ , we can use a modified ping-pong operation to multiplex the step. In this case, throughput  $T_h$  (as shown in (11)) becomes

$$T_h = \frac{nd}{\frac{1}{f_s} T_n}, \tag{15}$$

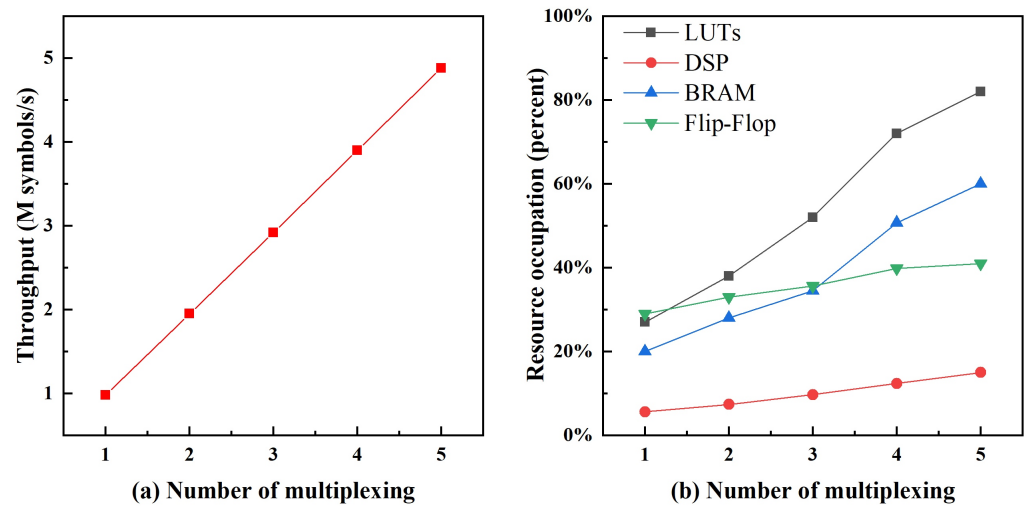
where  $n$  is the number for multiplexing. According to all the relevant parameters [15], the practical secret key rate of a CV-QKD system is given by

$$K_{prac} = \alpha(1 - FER) \left(\frac{n}{N}\right) (\beta I_{AB} - \chi_{BE} - \Delta(n)), \tag{16}$$

where  $\alpha = PP_{out}/PP_{in}$ ;  $PP_{out}$  and  $PP_{in}$  represent the post-processing output and input rates, respectively;  $FER$  is the frame error rate;  $n$  is the number of data used to distill the secret key;  $N$  is the number of sifted data after quantum transmission and measurement;  $I_{AB}$  is the mutual information between Alice and Bob;  $\chi_{BE}$  is the Holevo bound; and  $\Delta(n)$  is the finite-size offset factor. Notice that  $PP_{out}$  depends on  $T_h$  and satisfies  $PP_{out} \leq T_h$ ; therefore, coding throughput  $T_h$  affects the secret key rate of the CV-QKD system through  $\alpha$ . The consumed resources and throughput for different multiplexing structures are shown in Figure 7.

We can see that the data throughput and hard resources consumed increased with the number of multiplexing. A throughput of 0.98 Msymbols/s was achieved without multiplexing. The throughput increased to 4.88 M symbols/s when the number of multiplexing was five. The resource consumption of LUTs was the highest, whereas the resource consumption of DSPs was the lowest.

The throughput results of the multidimensional reconciliation encoding algorithm on an FPGA and a CPU are shown in Table 1. The model of the CPU was MXC-6301D(EA), running with the C programming language. It is evident that the FPGA-based encoding algorithm dramatically improved the data processing speed in comparison to the CPU-based encoding algorithm, which had a throughput of only 0.063 M per second.



**Figure 7.** Resource consumption and data throughput for different multiplexing structures.

**Table 1.** Comparison of the throughput of the multidimensional reconciliation encoding algorithm between an FPGA and a CPU.

Computing Platform	Throughput (M symbols/s)
FPGA (Xilinx Virtex-7)	4.88
CPU (MXC-6301D(EA))	0.063

Our proposed algorithm for obtaining the syndrome can be adapted to systems with different parity-check matrices. The simulated results on an FPGA are shown in Table 2. Three check matrices with two different rates (0.02 and 0.1) and two different code lengths (160,000 and 500,000) were used to calculate the syndrome. For a fixed code rate, the time taken was directly proportional to the code length. For a fixed code length, the time taken was slightly longer when the code rate was higher.

**Table 2.** Time taken for constructing the syndrome using different code rates and code lengths.

Rate	Length	Time (s)
0.1	160,000	0.00452
0.1	500,000	0.00904
0.02	500,000	0.00801

## 5. Conclusions

In this paper, we propose and demonstrate an FPGA-based multidimensional reconciliation encoding algorithm with variable data throughput that is suitable for applications in the real-time data post-processing of CVQKD systems. The syndrome construction algorithm consumes very little BRAMs and LUTs of the FPGA and can adapt to parity-check matrices with different code rates and code lengths. It is noted that the achieved data throughput is still limited. In our current algorithm, we find that matrix inversion consumes lots of hardware resources, which hinders the further improvement of throughput. In the future, we will improve the matrix inversion algorithm to make it compatible with pipeline operations; combined with higher system clock speed of the FPGA, a multidimensional reconciliation encoding algorithm with much higher throughput is possible.

**Author Contributions:** Q.L. conceived the study, performed the design of the algorithm and hardware implementations, and drafted the article; Z.L. and H.Y. assisted in the implementation of the hardware; S.Y. participated in the design of the algorithm, discussed the results, checked the draft, and critically reviewed the manuscript; Y.L. proposed and supervised the project, checked the draft,

and provided a critical revision of the manuscript. All authors have read and approved the final manuscript.

**Funding:** This work was supported by National Natural Science Foundation of China (NSFC) (grant No. 62175138), Key Research and Development Program of Guangdong Province (2020B0303040002), Aeronautical Science Foundation of China (20200020115001), Shanxi 1331KSC, and Scientific and Technological Innovation Programs of Higher Education Institutions in Shanxi Province (grant No.2021L258).

**Institutional Review Board Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

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

## References

- Xu, F.; Ma, X.; Zhang, Q.; Lo, H.K.; Pan, J.W. Secure quantum key distribution with realistic devices. *Rev. Mod. Phys.* **2020**, *92*, 025002.
- Pirandola, S.; Andersen, U.L.; Banchi, L.; Berta, M.; Bunandar, D.; Colbeck, R.; Englund, D.; Gehring, T.; Lupo, C.; Ottaviani, C.; et al. Advances in quantum cryptography. *Adv. Opt. Photonics* **2020**, *12*, 1012–1236.
- Ralph, T.C. Continuous variable quantum cryptography. *Phys. Rev. A* **1999**, *61*, 010303.
- Diamanti, E.; Leverrier, A. Distributing secret keys with quantum continuous variables: principle, security and implementations. *Entropy* **2015**, *17*, 6072–6092.
- Li, Y.M.; Wang, X.Y.; Bai, Z.L.; Liu, W.Y.; Yang, S.S.; Peng, K.C. Continuous variable quantum key distribution. *Chin. Phys. B.* **2017**, *26*, 040303.
- Laudenbach, F.; Pacher, C.; Fung, C.H.F.; Poppe, A.; Peev, M.; Schrenk, B.; Hentschel, M.; Walther, P.; Hübel, H. Continuous-variable quantum key distribution with Gaussian modulation—The theory of practical implementations. *Adv. Quantum Technol.* **2018**, *1*, 1800011.
- Walk, N.; Hosseini, S.; Geng, J.; Thearle, O.; Haw, J.Y.; Armstrong, S.; Assad, S.M.; Janousek, J.; Ralph, T.C.; Symul, T.; et al. Experimental demonstration of Gaussian protocols for one-sided device-independent quantum key distribution. *Optica* **2016**, *3*, 634–642.
- Wang, T.; Huang, P.; Zhou, Y.; Liu, W.; Ma, H.; Wang, S.; Zeng, G. High key rate continuous-variable quantum key distribution with a real local oscillator. *Opt. Express* **2018**, *26*, 2794–2806.
- Zhang, Y.; Chen, Z.; Pirandola, S.; Wang, X.; Zhou, C.; Chu, B.; Zhao, Y.; Xu, B.; Yu, S.; Guo, H. Long-distance continuous-variable quantum key distribution over 202.81 km of fiber. *Phys. Rev. Lett.* **2020**, *125*, 010502.
- Qi, B.; Gunther, H.; Evans, P.G.; Williams, B.P.; Camacho, R.M.; Peters, N.A. Experimental passive state preparation for continuous variable quantum communications. *Phys. Rev. Appl.* **2020**, *13*, 054065.
- Dequal, D.; Trigo Vidarte, L.; Roman Rodriguez, V.; Vallone, G.; Villoresi, P.; Leverrier, A.; Diamanti, E. Feasibility of satellite-to-ground continuous-variable quantum key distribution. *NPJ Quantum Inform.* **2021**, *7*, 3.
- Tian, Y.; Wang, P.; Liu, J.; Du, S.; Liu, W.; Lu, Z.; Wang, X.; Li, Y. Experimental demonstration of continuous-variable measurement-device-independent quantum key distribution over optical fiber. *Optica* **2022**, *9*, 492–500.
- Jain, N.; Chin, H.M.; Mani, H.; Lupo, C.; Nikolic, D.S.; Kordts, A.; Pirandola, S.; Pedersen, T.B.; Kolb, M.; Ömer, B.; et al. Practical continuous-variable quantum key distribution with composable security. *Nat. Commun.* **2022**, *13*, 4740.
- Pan, Y.; Wang, H.; Shao, Y.; Pi, Y.; Li, Y.; Liu, B.; Huang, W.; Xu, B. Experimental demonstration of high-rate discrete-modulated continuous-variable quantum key distribution system. *Opt. Lett.* **2022**, *47*, 3307.
- Yang, S.S.; Lu, Z.G.; Li, Y.M. High-speed post-processing in continuous-variable quantum key distribution based on FPGA implementation. *J. Lightw. Technol.* **2020**, *38*, 3935–3941.
- Van Assche, G.; Cardinal, J.; Cerf, N.J. Reconciliation of a quantum-distributed Gaussian key. *IEEE Trans. Inform. Theory* **2004**, *50*, 394–400.
- Leverrier, A.; Alléaume, R.; Boutros, J.; Zémor, G.; Grangier, P. Multidimensional reconciliation for a continuous-variable quantum key distribution. *Phys. Rev. A* **2008**, *77*, 042325.
- Bai, Z.; Yang, S.; Li, Y. High-efficiency reconciliation for continuous variable quantum key distribution. *Jpn. J. Appl. Phys.* **2017**, *56*, 044401.
- Wang, X.; Wang, H.; Zhou, C.; Chen, Z.; Yu, S.; Guo, H. Continuous-variable quantum key distribution with low-complexity information reconciliation. *Opt. Express* **2022**, *30*, 30455–30465.
- Jeong, S.; Jung, H.; Ha, J. Rate-compatible multi-edge type low-density parity-check code ensembles for continuous-variable quantum key distribution systems. *NPJ Quantum Inform.* **2022**, *8*, 6.
- Wang, H.; Xue, Y.; Qu, Y.; Mu, X.; Ma, H. Multidimensional Bose quantum error correction based on neural network decoder. *NPJ Quantum Inform.* **2022**, *8*, 134.
- Richardson, T.; Urbanke, R. *Modern Coding Theory*; Cambridge University Press: Cambridge, UK, 2008.

23. Milicevic, M.; Feng, C.; Zhang, L.M.; Gulak, P.G. Quasi-cyclic multi-edge LDPC codes for long-distance quantum cryptography. *NPJ Quantum Inform.* **2018**, *4*, 21.
24. Wang, X.; Zhang, Y.; Yu, S.; Guo, H. High speed error correction for continuous-variable quantum key distribution with multi-edge type LDPC code. *Sci. Rep.* **2018**, *8*, 10543.
25. Li, Y.; Zhang, X.; Li, Y.; Xu, B.; Ma, L.; Yang, J.; Huang, W. High-throughput GPU layered decoder of quasi-cyclic multi-edge type low density parity check codes in continuous-variable quantum key distribution systems. *Sci. Rep.* **2020**, *10*, 14561.
26. Wang, H.; Li, Y.; Pi, Y.; Pan, Y.; Shao, Y.; Ma, L.; Zhang, Y.; Yang, J.; Zhang, T.; Huang, W.; et al. Sub-Gbps key rate four-state continuous-variable quantum key distribution within metropolitan area. *Commun. Phys.* **2022**, *5*, 162.
27. Ai, X.; Malaney, R. Optimised multithreaded CV-QKD reconciliation for global quantum networks. *IEEE Trans. Commun.* **2022**, *70*, 6122–6132.
28. Li, H.; Pang, Y. FPGA-accelerated quantum computing emulation and quantum key distillation. *IEEE Micro* **2021**, *41*, 49–57.
29. Jouguet, P.; Kunz-Jacques, S.; Leverrier, A. Long-distance continuous-variable quantum key distribution with a Gaussian modulation. *Phys. Rev. A* **2011**, *84*, 062317.
30. Jose, S. 7 Series FPGAs Overview. 2018. Available online: [https://docs.xilinx.com/v/u/en-US/ds180\\_7Series\\_Overview](https://docs.xilinx.com/v/u/en-US/ds180_7Series_Overview) (accessed on 28 December 2022).
31. Vázquez-Castillo, J.; Castillo-Atoche, A.; Carrasco-Alvarez, R.; Longoria-Gandara, O.; Ortegón-Aguilar, J. FPGA-based hardware matrix inversion architecture using hybrid piecewise polynomial approximation systolic cells. *Electronics* **2020**, *9*, 182.
32. Golub, G.H.; Van Loan, C.F. *Matrix Computations*; JHU Press: Baltimore, MD, USA, 2013.
33. Fossorier, M.P. Quasicyclic low-density parity-check codes from circulant permutation matrices. *IEEE Trans. Inf. Theory* **2004**, *50*, 1788–1793.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.