LOW-POWER STAIRCASE ENCODER IMPLEMENTATION FOR HIGH-THROUGHPUT FIBER-OPTICAL COMMUNICATIONS

by

Shizhong Li

Submitted in partial fulfillment of the requirements
for the degree of Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
August 2019

Table of Contents

List of Tables

List of Figures

Abstract

This thesis presents a very large scale integration (VLSI) architecture of a high-throughput, low latency and power staircase forward error correction (FEC) encoder. The designed encoder achieves low latency and memory overhead by splitting the parity generation matrix and pre-computing partial parity bits for the next staircase block while generating the current staircase block. The proposed encoder is designed with multistage pipelined architecture that enables high efficiency in terms of throughput and area. The proposed staircase encoder was synthesized using 65nm CMOS technology. The throughput of the encoder achieves 432Gbps when operating at 909MHz, with the power consumption of 323mW.

List of Abbreviations Used

BCH	Bose-Hocquenghem-Chaudhuri
BER	Bit Error Rate
DCI	Data Center Interconnect
DWDM	Dense Wavelength Division Multiplexing
FEC	Forward Error Correction
FPGA	Field-programmable Gate Array
GF	Galois Field
HDL	Hardware Description Language
ITU	International Telecommunication Union
LCM	Least Common Multiple
LSB	Least Significant Bit
MSB	Most Significant Bit
MUX	Multiplexer
OIF	Optical Internetworking Forum
RAM	Random-access Memory
ROM	Read-only Memory
RTL	Register-transfer Level
VLSI	Very-large-scale Integration

Acknowledgements

I would like to express my sincerest gratitude to my supervisor Dr. Kamal El-Sankary and Co-supervisor Dr. Dmitry Trukhachev for their continuous support, patience and encouragement during my graduate study. I could not complete this thesis work without all the valuable suggestions from them.

I would like to thank Dr. Jason Gu and Dr. William J. Phillips for reviewing my thesis and being a part of my supervisory committee.

Many thanks to Mark Leblanc for providing technical supports. I am also grateful to our department secretaries Nicole Smith and Rebecca Baccardax for their support and paperwork preparations. I wish to thank my research group mates Abolfazl Zokaei, Alireza Karami and Heba Sultana for their encouragement and suggestion.

I would also like to express my gratitude to Precise-ITC and Mitacs Accelerate Program for providing financial support for this research work.

Special thanks to my girlfriend Bixin Chen for accompanying and supporting me throughout my graduate study.

Finally, I would like to express my deepest gratitude to my parents: my mother Dajun Nie and my father Qing Li, for their unconditional support and encouragement.

Chapter 1

Introduction

1.1 Motivations

Driven by the increasing demands in higher internet bandwidth, the development of high throughput optical-fiber communication system advanced rapidly. The 400G long-reach optical network standard [1] has been established by International Telecommunication Union (ITU) and ready to be adopted. Besides that, a 400ZR implementation agreement is developed by Optical Internetworking Forum (OIF) for designing pluggable optical modules for the data center interconnect (DCI). Due to the rapid growth of cloud-based computations, the traffic between data centers increased dramatically each year [2]. The pluggable optical modules are implemented in the DCI to increase the system bandwidth. In both the 400G and 400ZR optical network systems, the high-performance error-correction staircase codes are the key component to ensure high performance [1, 3].

Staircase codes, presented in [4], emerged as a competitor to a group of codes proposed for ITU-T G.975.1 standard dedicated to high-speed fiber-optical communication over submarine links. While the best G.975.1 code can guarantee ultra-reliable communication 0.96dB away from the Shannon capacity limit, staircase code tightened this gap and offered an improvement of 0.42dB over the best G.975.1 code. Staircase codes quickly gained a reputation of a low-complexity high-performance FEC for optical links and were generalized into a class of spatially-coupled split-component codes [5].

From the structural perspective staircase codes can be seen as a convolutional version of product codes [6]. The encoding of information is best represented by a staircase-shaped semi-infinite array (discussed in more detail in Chapter 3) where vertical and horizontal component codes cross-check each other. Such component code interconnection allows for use of low-complexity iterative decoding where hard bit-flipping decoding of component codes is performed in alternating manner. At the same time, the convolutional staircase-like structure creates a spatial graph coupling effect in a semi-infinite Tanner graph [7] of a staircase code. Spatial graph coupling allows the staircase decoder to benefit from the threshold-saturation phenomenon exhibited by spatially-coupled codes [8,9] and deliver near-capacity performance using iterative decoding [5].

The performance of the FEC codes and their capabilities to solve application-specific problems are often limited by their hardware implementation. Therefore, it is crucial to develop the optimal VLSI design for an FEC code. Unfortunately, very few studies have been published in terms of VLSI implementation of staircase codes with [10] addressing staircase encoding, and [11, 12] dedicated to the staircase decoder. FPGA emulations were performed on the concatenated staircase and Hamming code in [13]. The work [10] developed a parallel staircase encoder to achieve the throughput of 100Gbps. However, the design presented in [10] is not focused on energy-efficiency and the implementation presented there simply follows the mathematical steps of the encoding algorithm. As staircase codes make their way into present and future communication systems [14, 15] and standards the interest in efficient hardware implantation of these codes is rapidly increasing.

1.2 Contributions

1. This thesis discusses the staircase encoding process for the 400ZR optical network 400ZR optical network recommendation ITU-T 709.2/Y.1331.2 [1, 3] for 400Gpbs ethernet standard.

2. This thesis proposes a high-throughput and low-power VLSI design of a staircase encoder which fits the 400ZR optical network recommendation ITU-T 709.2/Y.1331.2 [1, 3] for 400Gpbs ethernet standard.

3. The proposed staircase encoder design achieves low latency and memory overhead by splitting the parity generation matrix into two parts that allow to pre-compute partial parity bits for the next staircase block while the current staircase block is in process of its generation.

4. Enabling high throughput and area efficiency by implementing multistage pipelined architecture on the staircase encoder.

1.3 Thesis Organization

The thesis is organized as follows:

1. Chapter 2 reviews different types of error-correcting codes that are related to the staircase codes. The 400G and 400ZR optical network systems are also discussed.

2. Chapter 3 presents the detailed encoding process of BCH and staircase codes based on the ITU-T recommendations.

3. Chapter 4 shows the proposed staircase encoder design and its VLSI implementation.

4. Chapter 5 presents the simulation and synthesis results of the proposed staircase encoder.

5. Chapter 6 concludes the thesis and discusses future work.

Chapter 2

Research Background

2.1 Error-Correcting Codes

With the advancement in the processing speed of electronics devices, the demands for high throughput and bandwidth of digital communication systems are also increasing. However, data can be corrupted when transferring in the channel of communication systems due to channel noises. The corrupted data lowers the reliability of the communication system. Therefore, it is crucial to implement error-correcting techniques in communication systems to flip the erroneous bits in the data [16].

Forward error correction (FEC) code is a commonly applied error correction method for digital communication systems. FEC codes assist the system to detect and flip numbers of corrupted in the received data without the need for data retransmission [16]. The technique of FEC coding is to first add redundant bits (parity bits) to the information data through encoding to form a codeword at the transmitter side. Then the codeword gets transmitted, and the receiver decodes the received codeword for detecting errors and correcting the corrupted bits in the information bits.

There are two main types of FEC codes, block codes and convolutional codes. The difference in the block and convolutional codes are in terms of the encoding process. The block codes encoding is an information bits block with fixed length followed by its parity bits block. The encoding of convolutional codes not only involves current information bits but also a certain number of previous information bits [17].

The metrics to evaluate the FEC codes, including code rate, coding gain, complexity and bit error rate (BER) requirement. The code rate is the ratio of information bits to codeword bits, which can provide information about the number of useful bits in the transmitted bits. The coding gain is the reduction of required signal to noise ratio $\frac{E_b}{N_o}$ (E_b is energy per bit, N_o is spectral noise density) between the FEC coded system and the uncoded system for achieving the same BER [16]. The powerful FEC codes also use the difference in $\frac{E_b}{N_o}$ between the Shannon limit and the coded system at a certain BER for performance measurement [18]. The Shannon limit defines the maximum data rate in a communication channel. The required pre-FEC BER for achieving the same post-FEC BER often used to compare different FEC codes.

2.2 Fiber-Optical Communication System

The fiber-optical communication system is a digital communication system, but unlike the traditional system where data is transmitted through copper, the fiber-optical communication system uses light to transmit data from transmitter to receiver through the optical fiber channel. There are several advantages of fiber-optical sys-

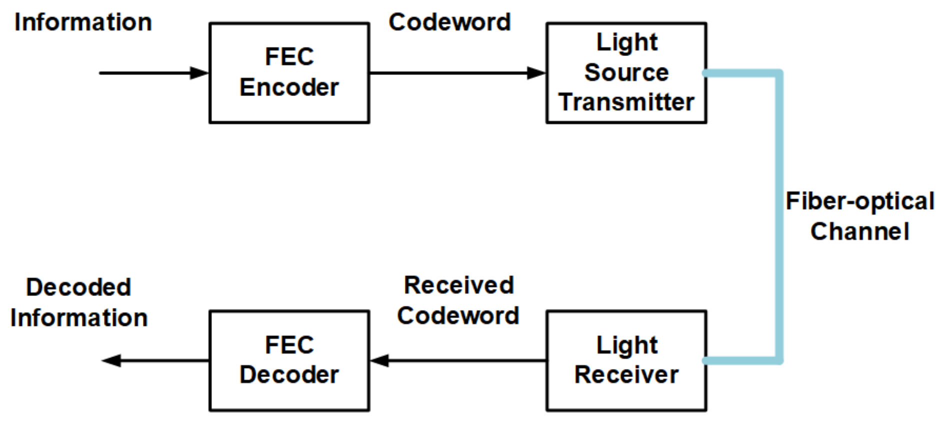

Figure 2.1: Fiber-optical communication system.

tem compares to the traditional digital communication system, including much higher bandwidth, faster transmission speed and lower cost. The fiber-optical channel is also immune to the electromagnetic interference which is a common noise factor for the copper system [19].

Figure 2.1 shows a simplified model of the fiber-optical communication system which is similar to the traditional digital communication system in terms of FEC encoder and decoder. The inputted information bits are encoded through FEC encoder to produce the corresponding codeword. Then the codeword is processed in the light source transmitter and subsequently transmitted through a fiber-optical channel. The light receiver detects the received light sources which then are transformed to codeword bits. The FEC decoder determines the correctness of received codeword, and the decoder flips the certain number of erroneous bits in the codeword if there are errors.

2.3 Finite Field

A finite field or Galois field is defined as a field that has a finite number of elements. The field is an algebraic structure which follows certain properties of addition, subtraction, multiplication and division. The finite field can be found in many applications for error control coding [20].

If we use p to represent the prime number, then a Galois field can be defined and denoted as $GF(p^m)$ for any m. The polynomial $b_{m-1}x^{m-1} + \ldots + b_1x^1 + b_0$ is used to represent the field $GF(p^m)$ where the coefficient b_k is in range of $\{0, 1, \ldots, p-1\}$ [16].

In terms of the implementation of Galois field on the error-correction codes, the binary field $GF(2^m)$ is commonly applied. The multiplication and addition are two frequently used arithmetic operations over $GF(2^m)$. Table 2.1 shows examples of these operations.

Table 2.1: Multiplication and Addition over $GF(2^m)$

$\times$	0	1
0	0	0
1	0	1

$+$	0	1
0	0	1
1	1	0

We can see that the multiplication over $GF(2^m)$ is the same as modulo-2 multiplication and XOR operation. The addition over $GF(2^m)$ is AND operation, or equivalently, a modulo-2 addition.

Let α be the root of the primitive polynomial, the nonzero elements of $GF(2^m)$ can then be represented by field elements $\{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^m-2}\}$. The primitive polynomials are defined as minimal polynomials for primitive elements in a finite field [17].

2.4 BCH Codes

The Bose, Chaudhuri, and Hocquenghem (BCH) codes are powerful multiple error-correcting cyclic codes over the finite field (Galois field) [21]. The BCH codes quickly become the common error-correction codes for digital communication systems since the first introduction due to their robust and efficient coding mechanism. Besides the excellent performance, the hardware-implementation-friendly algorithm also contributes to the popularity of the BCH codes. BCH codes are implemented as component codes in the staircase codes in the 400G recommendations [1,3].

The basics of the binary BCH codes are briefly discussed by following [20]. Let a be the primitive elements in GF(2^m) where m is any positive integer with value m $\geq$ 3. The t-error-correcting BCH codes can be constructed using generator polynomial $g(x)$ which is the lowest degree polynomial over GF(2). Then, the roots of the generator polynomial are $a, a^2, a^3, , \ldots, a^{2t}$. Let $p_i(x)$ be the minimal polynomial of α^i where i is any positive integer. The $g(x)$ can be found through $g(x) = LCM\{p_1(x), p_2(x), \ldots, p_{2t}(x)\}$.

If we consider a message vector m with length k, the vector can be presented as a polynomial $m(x) = m_0 + m_1x + m_2x^2 + \cdots + m_{k-1}x^{k-1}$. Then the codeword can be determined through

$$c(x) = x^{n-k}m(x) + x^{n-k}m(x) \pmod{g(x)} \tag{2.1}$$

where n is the codeword length.

2.5 Product Codes

The idea of product codes is to obtain powerful error-correcting codes through multiple low error-correction-capability codes. The encoding of product codes usually involves two component codes, one code is in charge of the horizontal encoding process and the other code is responsible for the vertical encoding process. Each encoding process generates one block of parity bits. After completion of vertical and horizontal processes, a block which filled with parity of parity bits can be generated. This block contains the most reliable parity bits [20].

We use $C_h(n_1, k_1)$ and $C_v(n_2, k_2)$ to represent two component codes used in the product codes. The generator matrix for $C_h(n_1, k_1)$ is $\mathbf{G}_h$ and $\mathbf{G}_v$ is for $C_v(n_2, k_2)$. The information block is represented as $\mathbf{A}$. The product codes performs horizontal encoding on $\mathbf{A}$ first by multiplying $\mathbf{G}_h$ with $\mathbf{A}$ to produce a block of row parity bits $\mathbf{P}_h$. Then the information block $\mathbf{A}$ is concatenated with $\mathbf{P}_h$ to form a block $[\mathbf{A}\ \mathbf{P}_h]$. The vertical encoding process then takes place by multiplying the $[\mathbf{A}\ \mathbf{P}_h]$ with $\mathbf{G}_v$ to get parity blocks $\mathbf{P}_h$ and $\mathbf{P}_{hv}$. The block $\mathbf{P}_{hv}$ is the parity of parity bits. The generated product code structure is illustrated in Figure 2.2. From this encoding process, we can see that each bit in any codeword row of product code is covered by both the horizontal and vertical parity bits. If the minimum distance of code C_h is d_h

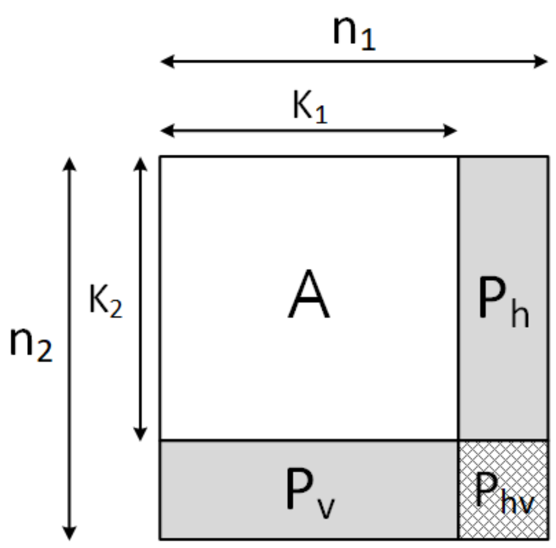

Figure 2.2: Structure of the product code.

and the minimum distance of C_v is d_v, then the minimum distance of their product codes is the $d_h d_v$ [20].

The advantage of product codes is their high error-correcting ability on burst and random errors [22]. Product codes also inspired the development of staircase codes [4].

2.6 Convolutional Codes

The operation of convolutional codes is different compares to block codes. In block codes, the parity bits are directly appended to information bits and the parity bits only relate to the present information bits. Convolutional codes apply a sliding-window technique which features memory elements in the encoding process of the information bit stream. The parity bits of convolutional codes not only depend on the current information bit but also depend on the previous information bits [16].

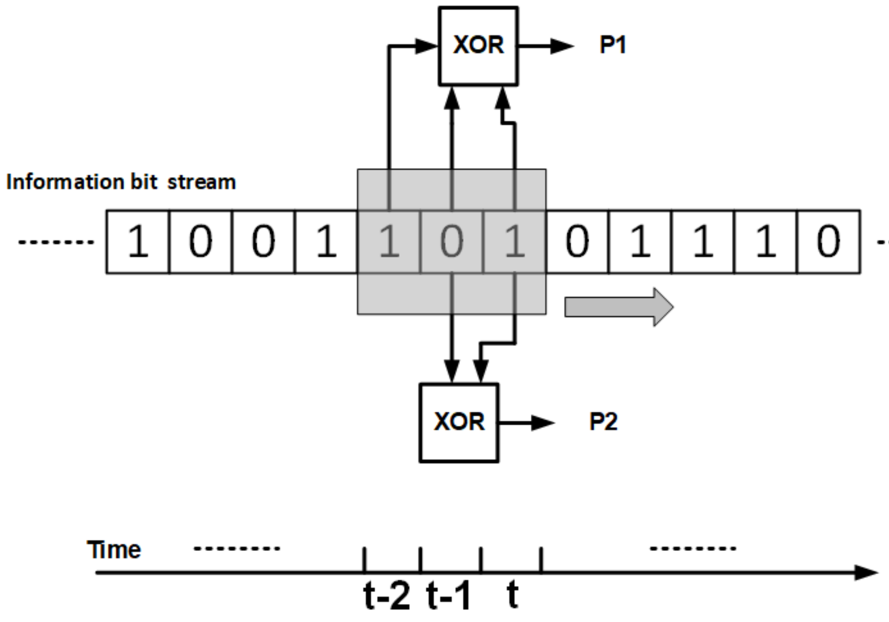

Figure 2.3: Convolutional codes sliding window encoding.

One example of the convolutional sliding window encoding of the information bit stream is illustrated in Figure 2.3. The gray rectangle box is the window which slides from left to right one bit at a time on the information bit stream. If we use m to represent the bits in the bit stream, then bits in the window are $m(t)$, $m(t-1)$ and $m(t-2)$ where $m(t)$ is the current bit under encoding. Then two parity bits $P1$ and $P2$ are computed through XOR-operation to encode the current bit $m(t)$. We can see that the previous bits are involved in the encoding of the current bit in the convolutional codes. To implement the sliding window encoding of convolutional codes, the memory elements of the codes are commonly implemented using linear shift registers.

The staircase codes apply the similar concepts of convolutional coding by replacing the bit in convolutional codes with the block of product codes [4]. The advantage of convolutional codes is that they are easier to implement in hardware compares to some linear block codes [23].

2.7 400G, 400ZR and FEC

400G (400Gbps) represents the technology of the next generation optical transport network. 400G refers to the throughput of the data in the optical network between hosts. The common interpretation of 400G is the single-carrier 400G [24].

400ZR or 400G ZR is a subset of single-carrier 400G technology and the implementation agreement of 400ZR is developed by OIF. The purpose of OIF 400ZR is to layout design standards for pluggable optical transceivers in the short-range (less than 120km) DWDM applications. The DWDM (Dense Wavelength Division Multiplexing) applications are commonly found in the data center for increasing the optical bandwidth through an adjustable number of optical transceivers. The design requirements for the optical transceiver based on 400ZR include high throughput (single-carrier 400G) and low power consumption (less than 15W when implementing in 7nm CMOS technology). The key component for achieving single-carrier 400G is to implement an efficient FEC to meet the performance requirements [3]. The algorithm of FEC also needs to be hardware-friendly and the hardware implementation of the FEC needs to be able to reach high throughput (400G) with low power consumptions.

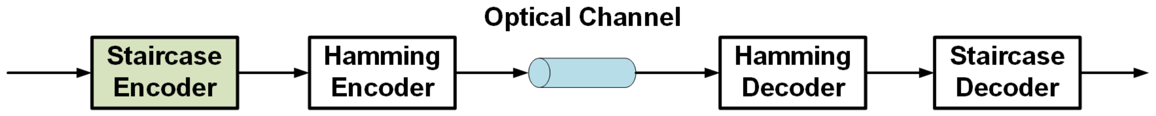

Figure 2.4: The 400ZR concatenated FEC.

The FEC of 400ZR is concatenated FEC as illustrated in Figure 2.4 with hard-decision staircase codes as outer codes and soft-decision hamming codes as inner codes. With the concatenated FEC implementation, the BER error floor can reach 10^{-15} and the net coding gain is 10.8dB [25].

Chapter 3

Staircase Codes

This chapter describes the encoding process of the BCH code which used as component code of staircase code based on the 400ZR optical network recommendation. In addition, the encoding process of the staircase code is discussed. Prior to the hardware implementation of staircase encoder, we need to build the staircase encoder in software that met the specifications. Therefore, the staircase encoder is implemented in Matlab before the Verilog implementation. The codewords outputs from the Matlab version of the encoder can be used to verify the outputs of the Verilog version of encoder. The Matlab implementation also assists with the hardware implementation by finding the optimal implementation method.

3.1 BCH Encoder

The BCH code is used as the component code in staircase coding. The main task in BCH encoding is to compute a generator matrix. Based on the 400ZR recommendation [3], the generator and parity-check matrices of BCH (1022,990) are obtained through the following steps. The equation (3.1) represents the primitive polynomial for the finite field $GF(2^{10})$.

$$p(x) = 1 + x^3 + x^{10} \tag{3.1}$$

If α is the root of the primitive polynomial, then nonzero elements of $GF(2^{10})$ can be represented as

$$GF(2^{10}) = \{\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{1022}\} \tag{3.2}$$

where $\alpha^0 = \alpha^{1023} = 1$. The binary representation of finite field elements is

$$\alpha^i = b_9\alpha^9 + b_8\alpha^8 + \ldots + b_0, 0 \leq i \leq 1022 \tag{3.3}$$

Since $b_9, b_8, \ldots, b_0$ are binary digits, the integer z can be applied to represent the decimal value of these digits, where $z = b_9 2^9 + b_8 2^8 + \ldots + b_0$. Then the function *log* and its inverse *exp* function are defined to associate the integer z with i which is the power of the primitive root.

$$log(z) = i, \quad exp(i) = z \tag{3.4}$$

Once the binary representations of finite field elements are generated, the column vector $\boldsymbol{f}(k)$ of the parity-check matrix $\mathbf{H}$ can be constructed through

$$\boldsymbol{f}(k) = \begin{bmatrix} \beta_k \\ \beta_k^3 \\ \beta_k^5 \\ D(\beta_k) \\ \overline{D(\beta_k)} \end{bmatrix}, \quad 1 \leq k \leq 1023 \tag{3.5}$$

where

$$\beta_k = \alpha^{log(k)} \tag{3.6}$$

and

$$D(\beta_k) = b_2\bar{b_1}\bar{b_0} \vee \bar{b_2}b_1 \vee \bar{b_2}\bar{b_1}b_0 \tag{3.7}$$

β_k is a 10-bit binary representation of primitive root α with power $log(k)$, and b is the bit in the binary representation. The bits sequence in β_k is the least significant bit (LSB) b_0 occupies the first row of the vector and the most significant bit (MSB) b_9 occupies the tenth row. $\bar{b}$ is the complement of b. The size of $\boldsymbol{f}(k)$ is 32 bits. The parity-check matrix $\mathbf{H}$ with size 32×1022 is defined as

$$\mathbf{H} = [\boldsymbol{f}(1021)\ \boldsymbol{f}(1022)\ \boldsymbol{f}(1)\ldots\boldsymbol{f}(510)\ \boldsymbol{f}(511+\pi^{-1}(0))\ldots\boldsymbol{f}(511+\pi^{-1}(509))] \quad (3.8)$$

where $\pi^{-1}(\cdot)$ is the inverse of the permutation function $\pi(\cdot)$ as shown below:

$$\begin{aligned}
&\pi(0:7) = 478:485 \quad \pi(8) = 0 \quad \pi(9:11) = 486:488 \quad \pi(12) = 1\\
&\pi(13) = 489 \quad \pi(14:16) = 2:4 \quad \pi(17:19) = 490:492 \quad \pi(20) = 5\\
&\pi(21) = 493 \quad \pi(22:24) = 6:8 \quad \pi(25) = 494 \quad \pi(26:32) = 9:15\\
&\pi(33:35) = 495:497 \quad \pi(36) = 16 \quad \pi(37) = 498 \quad \pi(38:40) = 17:19\\
&\pi(41) = 499 \quad \pi(42:48) = 20:26 \quad \pi(49) = 500 \quad \pi(50:64) = 27:41\\
&\pi(65:67) = 501:503 \quad \pi(68) = 42 \quad \pi(69) = 504 \quad \pi(70:72) = 43:45\\
&\pi(73) = 505 \quad \pi(74:80) = 46:52 \quad \pi(81) = 506 \quad \pi(82:128) = 53:99\\
&\pi(129) = 507 \quad \pi(130) = 100 \quad \pi(131) = 508 \quad \pi(132:256) = 101:225\\
&\pi(257) = 509 \quad \pi(258:509) = 226:477 \qquad (3.9)
\end{aligned}$$

The permutation function expression $\pi(X : X+N) = Y : Y+N$ represents the following functions, $\pi(X) = Y$, $\pi(X+1) = Y+1$,...,$\pi(X+N) = Y+N$. The matrix $\mathbf{C}$ consists the last 32 columns of $\mathbf{H}$ and the inverse of this matrix is $\mathbf{C^{-1}}$. Then,

$$\mathbf{C^{-1}H} = [\mathbf{P}^T |\ \mathbf{I}] \quad (3.10)$$

The BCH code is a systematic linear code, thus the generator matrix can always be written as

$$\mathbf{G} = [\mathbf{I} \mid \mathbf{P}] \tag{3.11}$$

Finally, parity-generation matrix $\mathbf{P}$ with size 990×32 is obtained. The parity bits of BCH codeword can then be computed by multiplying $\mathbf{P}$ matrix to the information bits.

In terms of the Matlab implementation of BCH encoder which based on the procedures above, the first step is to generate 10-bit binary representations for all the primitive roots ($0 \leq i \leq 1022$). The build-in Matlab function *gftuple* is applied here to generate binary representations of field elements based on the primitive polynomial $p(x) = 1 + x^3 + x^{10}$. The 1023 binary numbers are then converted to their corresponding decimal values. A look-up table for log / exp functions which relates primitive root power i to integer z is prepared. Table 3.1 shows a few results from the look-up table. When computing the β_k in the column vector, decimal value z in the look-up table

Table 3.1: Parts of the look-up table for binary representations of primitive root, power of primitive root and their decimal values

Binary Representations	**Power of primitive root (i)**	**Decimal values (z)**
0000000001	0	1
0000001001	10	9
1000010001	100	529
1111101010	500	1002
1000000100	1022	516

is replaced by k. Using column vector $\boldsymbol{f}(511 + \pi^{-1}(0))$ as an example. Based on the permutation function, $\pi^{-1}(0) = 8$. So the integer $k = 511 + 8 = 519$. We can then obtain $\log(519) = 955$ based on the look-up table. Therefore, the β_k elements of the column vector are $\beta_k = \alpha^{955} = [1110000001]^T$(LSB to MSB), $\beta_k^3 = \alpha^{2865} = \alpha^{819} =$

$[0111000110]^T$ and $\beta_k^5 = \alpha^{4775} = \alpha^{683} = [0001101110]^T$. With $b_2 = b_1 = b_0 = 1$ (bits of β_k) and (3.7), we can get $F(\beta_k) = 0$ and $\overline{F(\beta_k)} = 1$. Thus, the final result of column vector $\boldsymbol{f}(511 + \pi^{-1}(0))$ is $[111000000101110001100001101111001]^T$. The computation of parity-generation matrix can then be completed after the parity-check matrix is obtained.

3.2 Staircase Codes

The staircase encoding process is algorithmically represented using a semi-infinite two-dimensional array given in Fig. 3.1. The array consists of blocks $\mathbf{B}_k, k = 0, 1, 2, \ldots$ that form a staircase pattern and are filled with information and parity-check symbols throughout the encoding process.

The initial block $\mathbf{B}_0$ is filled with known symbols (zeros in our case) to initiate the process of spatial graph coupling which is utilized at the receiver during the data decoding process. The first group of information bits to be encoded is filled into the left part of block $\mathbf{B}_1$. The corresponding positions are shaded in gray in Figure 3.1. Once the information bits are filled in the rows of the two adjacent blocks $(\mathbf{B}_0, \mathbf{B}_1)$, they are encoded with component code encoders (the Bose-Chaudhuri-Hocquenghem (BCH) codes commonly serve as component codes) and the resulting parity-check bits are filled into the white area representing the right part of block $\mathbf{B}_1$. This is regarded as a *horizontal* encoding operation. The next component code encoding operation is *vertical.* The second group of the information bits is filled onto the gray area at the top of block $\mathbf{B}_2^T$. Then the columns of the matrix $(\mathbf{B}_1, \mathbf{B}_2^T)^T$ are encoded vertically by the component BCH codes and the resulting parity bits are filled into the white area at the bottom of the block $\mathbf{B}_2^T$. The process then continues with the third block $\mathbf{B}_3$ filled with new information bits and the parity bits resulting from horizontal encoding and so on. The *horizontal* and *vertical* encoding steps are executed in alternating

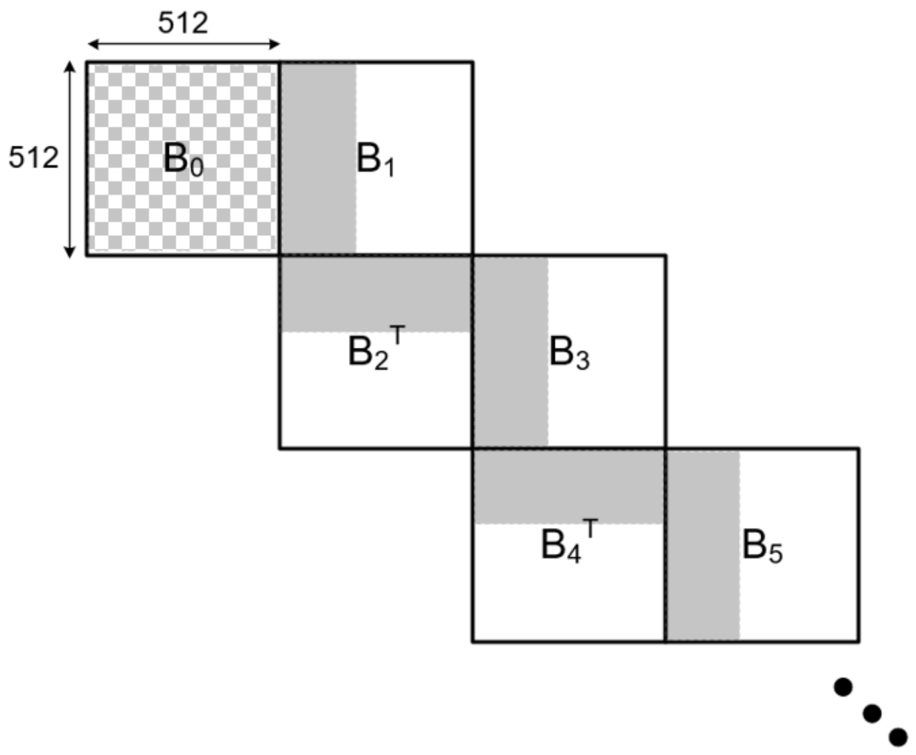

Figure 3.1: Visualization of a staircase code array representation.

manner. Figure 3.2 shows two adjacent staircase blocks $\mathbf{B}_{k-1}^T$ (already encoded) and $\mathbf{B}_k$. By $\mathbf{A}_k$ we denote the new block of information filled in at stage k and $\mathbf{C}_k$ is the parity block resulting from the component code encoding. A row in $\mathbf{B}_{k-1}^T$ block concatenated with the corresponding row in $\mathbf{B}_k$ block forms a valid component code codeword.

The construction of the staircase array specified in the recommendation [1] has two minor differences with the classic staircase encoding. The shortened BCH $(1022, 990)$ codes are considered and, therefore, to form squared 512×512 staircase blocks two rows of zeros are added at every step as described below. In addition, the encoding is not precisely carried out row-wise and column-wise, but, instead, jth row in $\mathbf{B}_{k-1}^T$ (see Figure 3.2) is encoded together with ith row of $\mathbf{B}_k$. The index j is generated from i using a permutation matrix given in [1]. The precise encoding algorithm is described mathematically by the following steps:

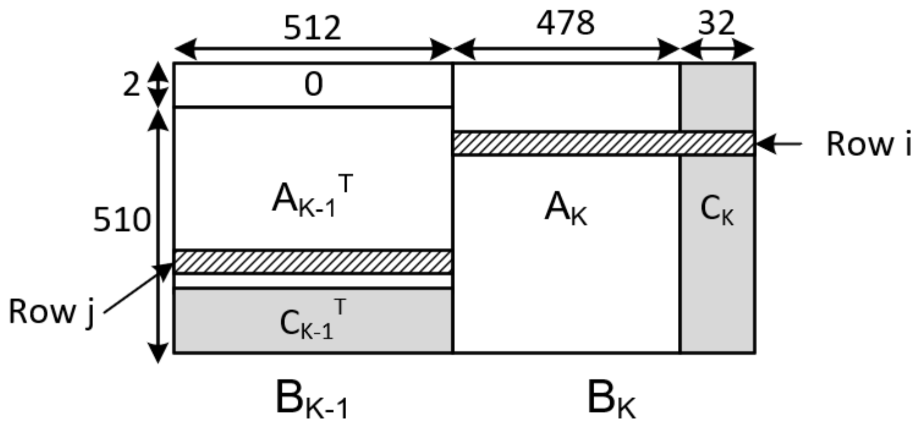

Figure 3.2: Two adjacent staircase blocks.

1. Select a 512×512 all-zeros matrix as reference staircase block $\mathbf{B}_0$;
2. Compute the parity block $\mathbf{C}_1$ by using $\mathbf{B}_0$ and information matrix $\mathbf{A}_1$. The staircase block $\mathbf{B}_1$ is formed by appending $\mathbf{C}_1$ to $\mathbf{A}_1$;
3. Insert two all-zeros rows to transposed $\mathbf{B}_1$ to change the block size to 512×512;
4. Apply the same process to compute $\mathbf{C}_2$ and $\mathbf{B}_2$ for information matrix $\mathbf{A}_2$. This iterative process continues until all the information matrix $\mathbf{A}_k$ are encoded.

Figure 3.3 illustrates the steps of staircase encoding process. The parity blocks $\mathbf{C}_k$ are generated through BCH encoding. A parity-generation matrix $\mathbf{P}$ (size 990×32) is pre-computed based on the BCH parity check matrix given in section 3.1. The following steps are used to compute row i of $\mathbf{C}_k$:

$$\mathbf{C}_k(i) = [\mathbf{B}_{k-1}^T(i)\ \mathbf{A}_k(i)] \times \mathbf{P} \qquad \text{for } 1 \leq i \leq 2 \tag{3.12}$$

$$\mathbf{C}_k(i) = [\mathbf{B}_{k-1}^T(\pi(i-3))\ \mathbf{A}_k(i)] \times \mathbf{P} \qquad \text{for } 3 \leq i \leq 512 \tag{3.13}$$

Figure 3.2 also shows a valid 1022-bit BCH codeword is formed by taking the row j in $\mathbf{B}_{k-1}^T$ concatenated with row i in $\mathbf{B}_k$, where $j = \pi(i-3)$. The bits in row j are the leftside of BCH codeword and bits in row i corresponding to rightside of the

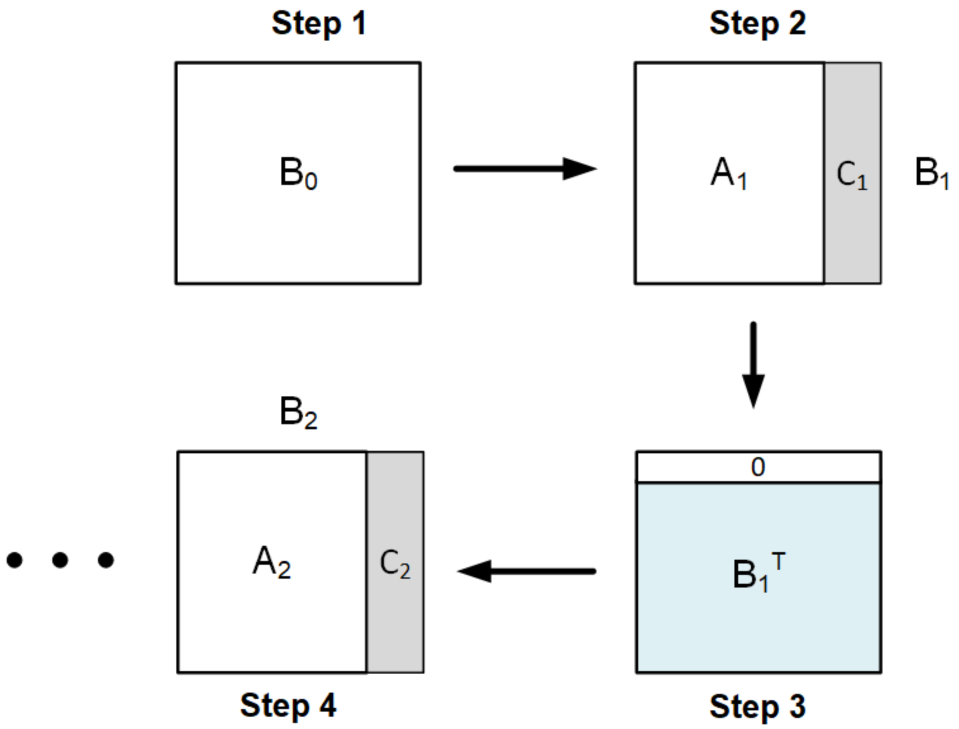

Figure 3.3: Staircase encoding process.

codeword. For example, if $i = 50$, then $j = \pi(47) = 25$. Therefore, a valid BCH code is formed by concatenating row 25 from $\mathbf{B}_{k-1}^T$ and row 50 from $\mathbf{B}_k$.

For the Matlab implementation of staircase encoder, random binary blocks with size 512×478 are generated to be applied as information blocks $\mathbf{A}_k$ to the staircase encoder. Figure 3.4 shows the Matlab model of the staircase encoding process. When an information block is transmitted to staircase encoder, the encoder will first determine the index (k) of this block. If the block is the first block $(k = 1)$, then the encoder selects the reference staircase block B_0 as the previous staircase block when computing parity bits for the inputted information block. Otherwise, the encoder uses previous staircase block $B_{k-1}(k \geq 2)$ to generate the parity bits. The permutation $\pi(\cdot)$ look-up table is embedded in the staircase encoder which gives the index of the corresponding row from the previous staircase block for the index of the row of current staircase block. With equations 3.12 and 3.13, the parity bits matrix (size

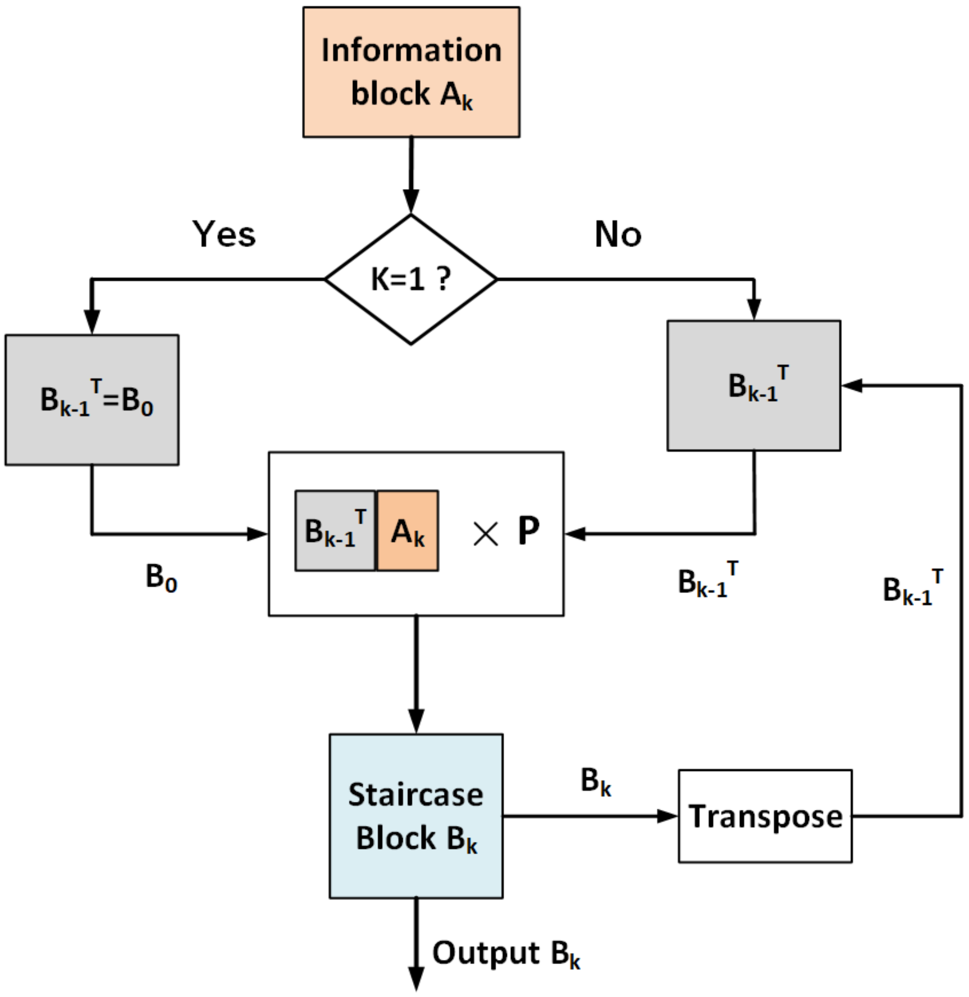

Figure 3.4: The Matlab model of the staircase encoding process.

512 × 32) for the information block can be computed. The staircase block is then a concatenation between information block and parity bits matrix. Once a staircase block $\mathbf{B}_k$ (size 512 × 510) is ready, it gets transmitted immediately. After that, one copy of the block $\mathbf{B}_k$ is transposed and stored in the memory of the staircase encoder for computing the parity bits matrix for the next information block.

To verify the correctness of the generated staircase blocks, these blocks are tested by re-computing the parity bits for the information vectors through BCH encoder. By taking information vectors from current staircase block and their corresponding rows of the previous block, the 990-bit information vectors (concatenated based on permutation function) are formed and multiplied with the parity generation P matrix to produce 32-bit parity bits. These bits are then being compared with the parity bits of generated staircase blocks. If two results are equivalent, the generated staircase blocks are valid.

Chapter 4

Staircase Encoder Implementation

In this chapter, the VLSI implementation of staircase encoder is presented. The hardware design of staircase encoder is focused on low-power, high throughput and low latency aspects. This chapter also discusses the adaptation of staircase encoding process for the optimal hardware implementation. The design of the encoder is implemented using Verilog hardware description language. With the pre-computed parity generation matrix, the complexity of staircase encoder design can be reduced.

4.1 Staircase Encoding process for Hardware Implementation

To adapt the staircase encoding process towards hardware implementation, the parity generation matrix $\mathbf{P}$ is split into two sub-matrices: $\mathbf{P}_a$ of size 512×32 and $\mathbf{P}_b$ of size 478×32 to allow pipelined computation with reduced memory size. The size of $\mathbf{P}_a$ and $\mathbf{P}_b$ matrices are determined based on the size of the information block and transposed previous staircase block.

$$\mathbf{P} = \left[\begin{array}{ccccccc} 0 & 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1 \end{array}\right] \begin{array}{l} \left.\vphantom{\begin{array}{c}0\\0\\\vdots\\1\end{array}}\right\} \mathbf{P}_a \\ \left.\vphantom{\begin{array}{c}0\\\vdots\\0\end{array}}\right\} \mathbf{P}_b \end{array}$$

$\mathbf{P}_b$ is used to compute parity bits of current staircase block $\mathbf{B}_k$ and $\mathbf{P}_a$ is involved in computing parity bits of previous staircase block $\mathbf{B}_{k-1}^T$. Hence (4.1) and (4.2) change to

$$\mathbf{C}_k(i) = \mathbf{B}_{k-1}^T(i) \times \mathbf{P}_a + \mathbf{A}_k(i) \times \mathbf{P}_b \qquad \text{for } 1 \leq i \leq 2 \tag{4.1}$$

$$\mathbf{C}_k(i) = \overbrace{\mathbf{B}_{k-1}^T(\pi(i-3)) \times \mathbf{P}_a}^{\mathbf{C}_{p1,k}} + \overbrace{\mathbf{A}_k(i) \times \mathbf{P}_b}^{\mathbf{C}_{p2,k}} \qquad \text{for } 3 \leq i \leq 512 \tag{4.2}$$

4.2 Staircase Encoder Design

To achieve high throughput and low latency, we have developed a parallel staircase encoder design. The proposed architecture is shown in Figure 4.1, which includes a Bit-matrix Multiplier, Bit-Matrix Adder, Parity Generation Units, Parity-Bit Registers, ROMs, Multiplexer, Controller, Information Buffer and Data Formatter. The staircase encoder processes a 478-bit input data (one row of information block) at each clock cycle and appends 32 parity bits to the input sequence in order to output a 510-bit codeword. This is equivalent to the computation of one row of the staircase block. The parity bits are computed in parallel and pipelined to achieve high data throughput. The pre-computed $\mathbf{P}_a$ and $\mathbf{P}_b^T$ matrices are stored in ROM of the staircase encoder. Furthermore, $\mathbf{P}_a$ and $\mathbf{P}_b^T$ matrices are involved in the parallel computation of the parity bits in the current and previous staircase blocks. Instead of storing the entire previous staircase block in order to compute parity bits for the current staircase block, only 32 parity bits of the transposed previous staircase block are computed and stored. As a result, the required memory size is reduced to the minimum of information required to continue encoding. The 478-bit input data of

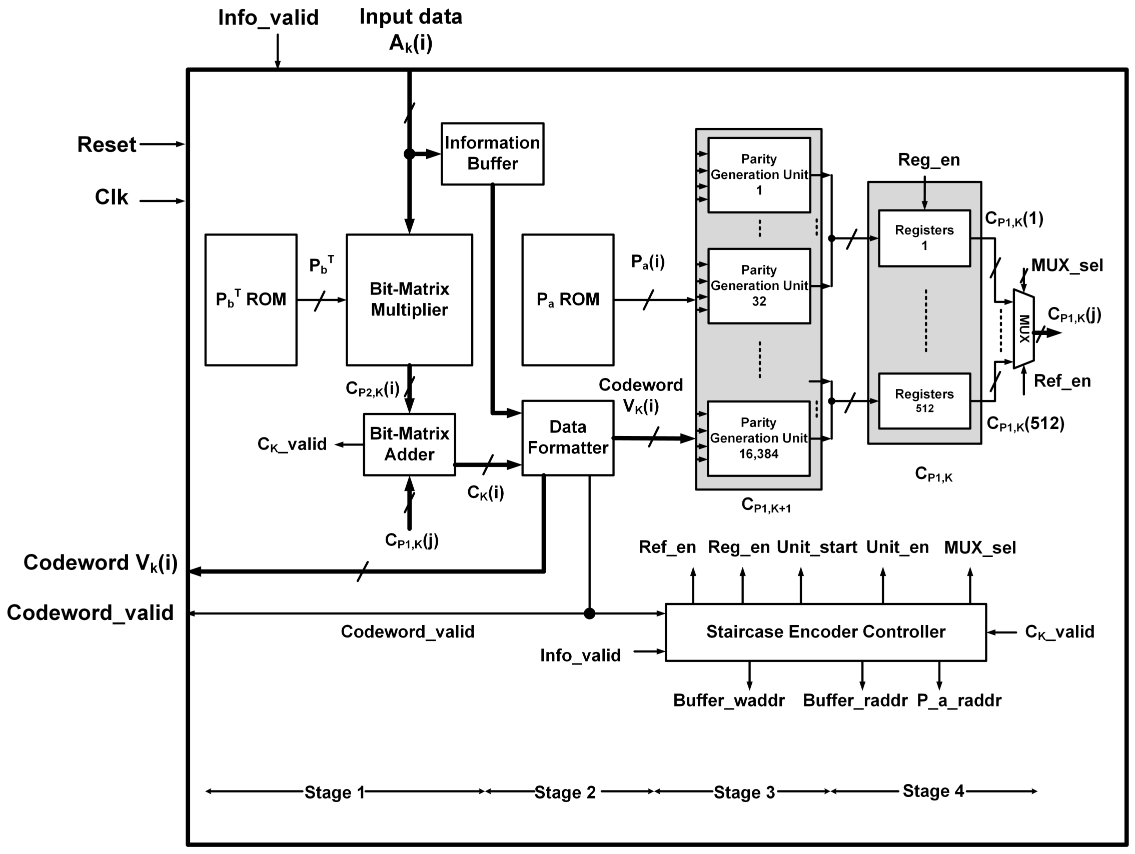

Figure 4.1: Architecture of the proposed staircase encoder.

the staircase encoder is denoted as $\mathbf{A}_k(i)$ in Figure 4.1, where k is the block index, indicating the index of the staircase block. Furthermore, i is the row index of the input data, e.g., input data $\mathbf{A}_1(1)$ is located in the first row of staircase block $\mathbf{B}_1$. The input data $\mathbf{A}_k(i)$ is passed to three parts of the encoder: the Bit Matrix Multiplier the Information Buffer. Figure 4.2 presents a timing diagram showing the pipelined operation of the staircase encoder. There are four pipelined stages in the hardware implementation of the staircase encoder. The input data $\mathbf{A}_k(i)$ is first being processed by Multiplication and Addition operations which take place in the Bit-Matrix Multiplier and Bit-Matrix Adder to compute parity bits $\mathbf{C}_k(i)$ for the input data. The next operation is to combine the $\mathbf{A}_k(i)$ with $\mathbf{C}_k(i)$ in the Data Formatter to output codeword $\mathbf{V}_k(i)$. After that, the Parity Generation processes the codeword in Parity Generation Units to compute partial parity bits $\mathbf{C}_{p1,k+1}$. When all the codewords in one staircase block are processed, $\mathbf{C}_{p1,k+1}$ becomes $\mathbf{C}_{p1,k}$ (complete parity bits) which are used in the incoming information block. The staircase encoder contains a control unit which is the Controller shown in Figure 4.1. The Controller assists the encoder to schedule the encoding process by changing values of control signals. The following subsections discuss each module of staircase encoder in more details.

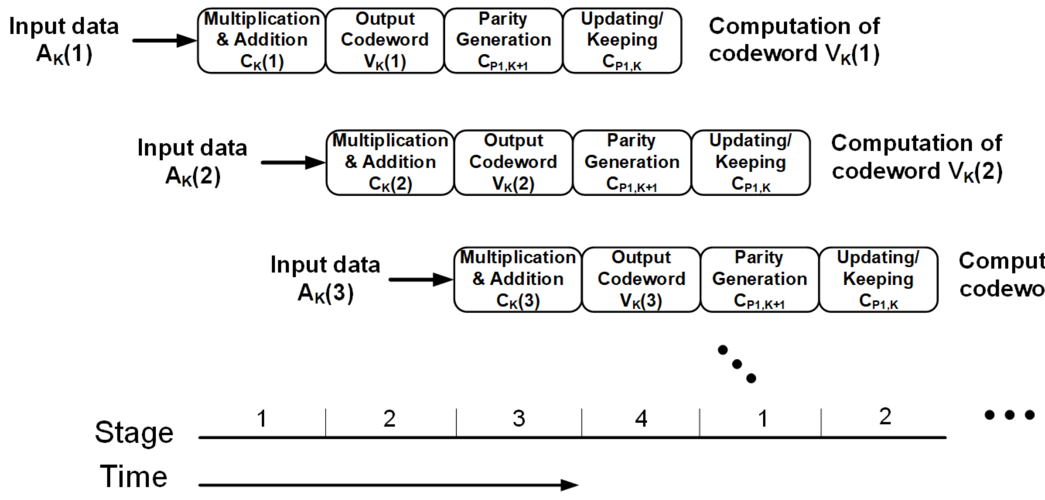
Input data $A_K(1)$
Multiplication & Addition $C_K(1)$
Output Codeword $V_K(1)$
Parity Generation $C_{P1,K+1}$
Updating/ Keeping $C_{P1,K}$
Computation of codeword $V_K(1)$
Input data $A_K(2)$
Multiplication & Addition $C_K(2)$
Output Codeword $V_K(2)$
Parity Generation $C_{P1,K+1}$
Updating/ Keeping $C_{P1,K}$
Computation of codeword $V_K(2)$
Input data $A_K(3)$
Multiplication & Addition $C_K(3)$
Output Codeword $V_K(3)$
Parity Generation $C_{P1,K+1}$
Updating/ Keeping $C_{P1,K}$
Computation of codeword $V_K(3)$
Stage
1
2
3
4
1
2
Time

Figure 4.2: The timing diagram for the pipelined operation of the staircase encoder.

4.2.1 Bit-Matrix Multiplier

In the Bit-Matrix Multiplier shown in Figure 4.3, the modulo-2 multiplication is applied to $\mathbf{A}_k(i)$ and $\mathbf{P}_B^T$ in parallel by ANDing and XORing of bits in $\mathbf{A}_k(i)$ with rows in $\mathbf{P}_b^T$ to produce 32 partial parity bits $\mathbf{C}_{p2,k}(i)$. The Bit-Matrix Multiplier consists of 32 Parity Multipliers that compute the 32 parity bits of $\mathbf{C}_{p2,k}(i)$ at the same time.

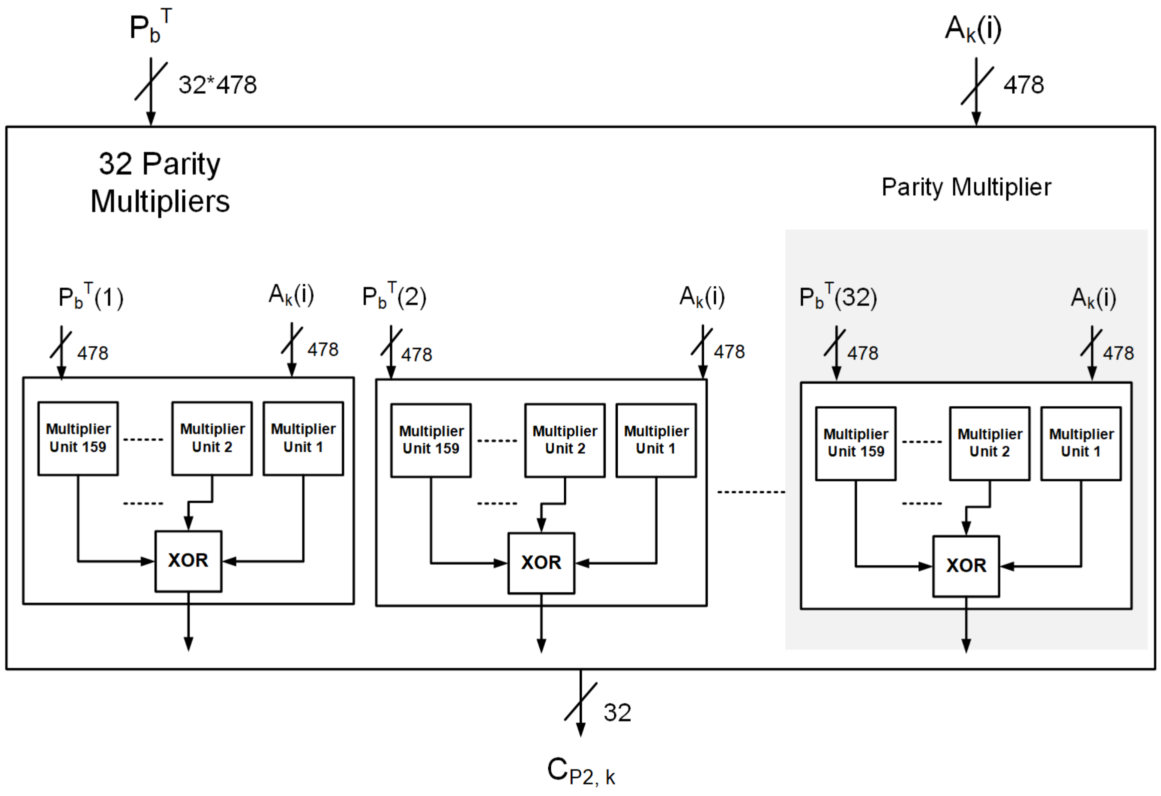

Figure 4.3: The architecture of the Bit-Matrix Multiplier.

Each Parity Multiplier computes one bit of $\mathbf{C}_{p2,k}(i)$. The parallel computation also happens inside the Parity Multiplier by using 159 Multiplier Units. Each Multiplier Unit shown in Figure 4.4 process 3 bits from a row of $\mathbf{P}_b^T$ and $\mathbf{A}_k(i)$ respectively. The number of input bits of Multiplier Units is selected to achieve high parallelization level. The bits generated from Multiplier Units are XORed with the bits generated by ANDing the last bit of $\mathbf{A}_k(i)$ with the last bit of row of $\mathbf{P}_b^T$. The final computed value from a Parity Multiplier forms one of the bits in $\mathbf{C}_{p2,k}(i)$. The Bit-Matrix Multiplier implemented purely combinational logic and there is no clock input to the Multiplier, hence the power consumption is reduced.

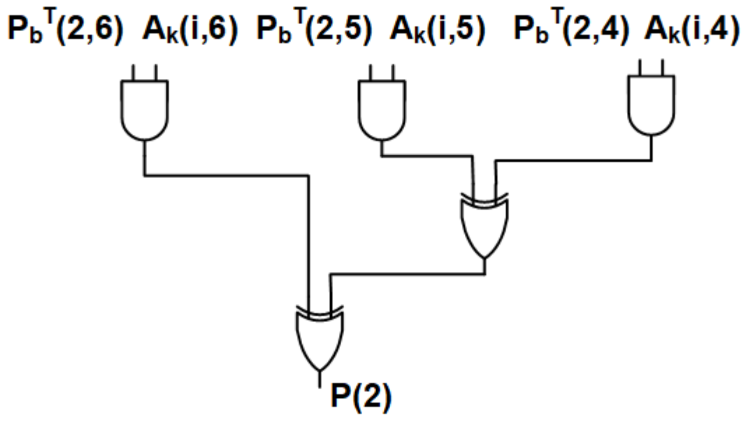

Figure 4.4: Architecture of the Multiplier Unit 2 in the Parity Multiplier 2.

4.2.2 Bit-Matrix Adder

At the next step, the $\mathbf{C}_{p2,k}(i)$ adds to the 32 partial parity bits (pre-computed) $\mathbf{C}_{p1,k}(j)$ through modulo-2 addition to generate 32 complete parity bits $\mathbf{C}_k(i)$ that are concatenated to the input data $\mathbf{A}_k(i)$. The permutation function π determines the row index j in $\mathbf{C}_{p1,k}(j)$ matrix corresponding to the row index i in $\mathbf{C}_{p2,k}(i)$ matrix. When the parity bits $\mathbf{C}_k(i)$ for the current input data $\mathbf{A}_k(i)$ is computed, the Bit-Matrix Adder outputs parity bits valid signal (C_k_valid) to the staircase encoder Controller.

4.2.3 Data Formatter and Information Buffer

The parity bits are generated three clock cycles after the corresponding information vector are inputted to the encoder. Therefore, the information vectors inputted during this period need to be stored temporarily so the computed parity bits can be concatenated with their matched information vectors to form codeword output. The reading and writing orders of the Information Buffer are controlled by the read address (Buffer_raddr) and write address (Buffer_waddr) signals provided by the Controller. The controlling mechanism of the Buffer is discussed in details in Section 4.2.7.

The 510-bit codeword $\mathbf{V}_k(i)$ is formed by concatenating the input data $\mathbf{A}_k(i)$ with the corresponding parity bits $\mathbf{C}_k(i)$ in the Data Formatter. The codeword $\mathbf{V}_k(i)$ is then immediately transmitted. Two additional 0s are inserted before the MSB of the current codeword $\mathbf{V}_k(i)$ and the length of the codeword is now 512 bits. The extended codeword is inputted to the Parity Generation Units. Besides outputting codeword, the Data Formatter also outputs codeword valid signal (Codeword_valid) to the Controller as well as the output port of the encoder to acknowledge the peripheral components a valid codeword is presented.

4.2.4 Parity Generation Units

The Parity Generation Units compute the 32 partial parity bits $\mathbf{C}_{p1,k+1}$, which are used to compute the next staircase block. There are 512 32-bit $\mathbf{C}_{p1,k+1}$ to be parallelly computed; thus, the system requires 16,384 total Parity Generation Units. Figure 4.5 shows the Parity Generation Unit.

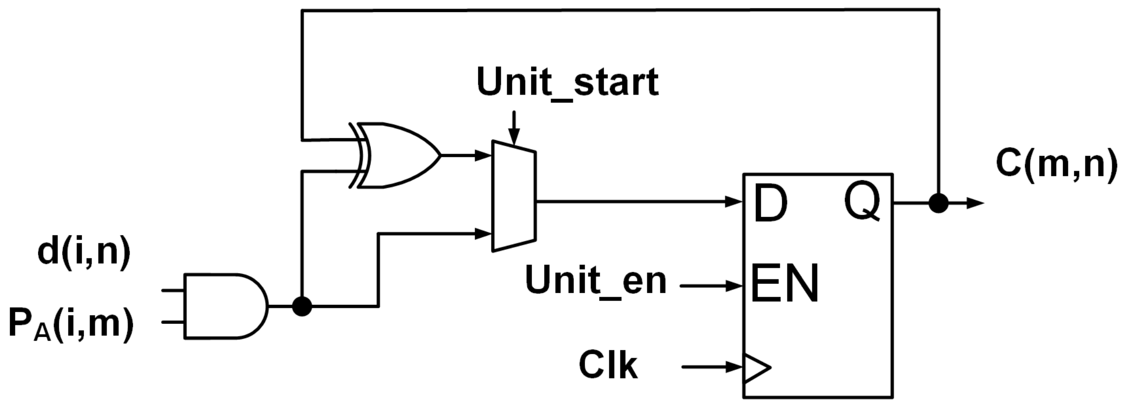

Figure 4.5: The parity generation unit.

Each Unit generates one parity bit in $\mathbf{C}_{p1,k+1}$ once the system processes all 512 rows in one staircase block. The Units generate parity bits in parallel so that once a staircase block is processed, the units complete the computation of all 512 $\mathbf{C}_{p1,k+1}$ and these are prepared for use in the encoding of next staircase block. We denote a bit in transposed $\mathbf{C}_{p1,k+1}$ matrix by $\boldsymbol{c}(m,n)$ and the bit in the previous staircase block $\mathbf{B}_{k-1}$ with two additional all-zeros rows inserted by $\boldsymbol{d}(q,n)$ where q and m are

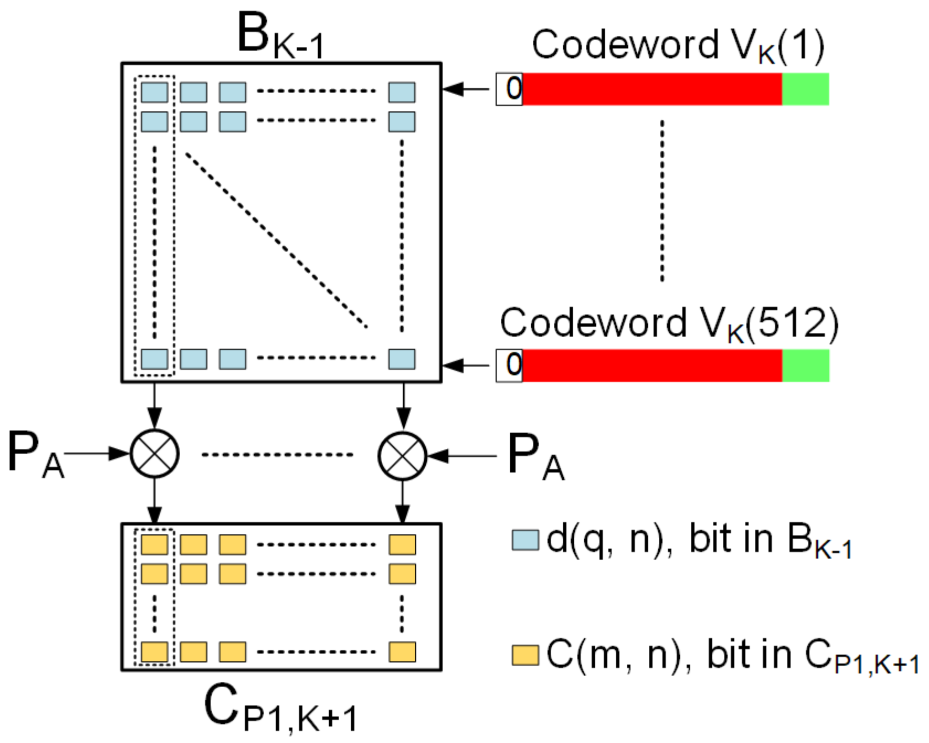
B_{K-1}
Codeword V_K(1)
0
Codeword V_K(512)
0
P_A
P_A
d(q, n), bit in B_{K-1}
C(m, n), bit in C_{P1,K+1}
C_{P1,K+1}

Figure 4.6: The parity generation of $C_{p1,k+1}$.

row indices and n is column index. Then $\boldsymbol{c}(m, n)$ can be computed as follows,

$$\boldsymbol{c}(m, n) = \sum_{q=1}^{512} \boldsymbol{d}(q, n) \times \mathbf{P}_a(q, m) \tag{4.3}$$

The parity generation of $C_{p1,k+1}$ is presented in Figure 4.6. For every new row of data inputted into the encoder, $\boldsymbol{c}(m, n)$ is partially updated with the relevant codeword bit and its corresponding parity from $\mathbf{P}_a$ matrix. For example, when the codeword $\mathbf{V}_k(1)$ is ready, the first row in $\mathbf{B}_{k-1}$ block is also known and bits in the row are denoted as $\boldsymbol{d}(1, 1), \boldsymbol{d}(1, 2), \ldots, \boldsymbol{d}(1, 512)$. We can then simultaneously compute the first component $\boldsymbol{d}(1, n) \times \mathbf{P}_a(1, m)$ for every bit in $\mathbf{C}_{p1,k+1}$ through parity generation units. When the codeword $\mathbf{V}_k(2)$ is computed, the parity generation units compute the second component $\boldsymbol{d}(2, n) \times \mathbf{P}_a(2, m)$ and the results are added to the first component. Once the codeword $\mathbf{V}_k(512)$ is processed, all the bits are computed and ready to be used for the next staircase block input. Equation (3.6) represents computation of parity bit $\boldsymbol{c}$ throughout the clock cycles t where $1 \leq t \leq 512$. Figure 4.7 depicts the timing diagram of the parity generation based on (3.6). In this diagram, parity bits $c^t([1:32], q)$, $1 \leq q \leq 512$ are computed in parallel and are only partially computed during $1 \leq t \leq 511$.

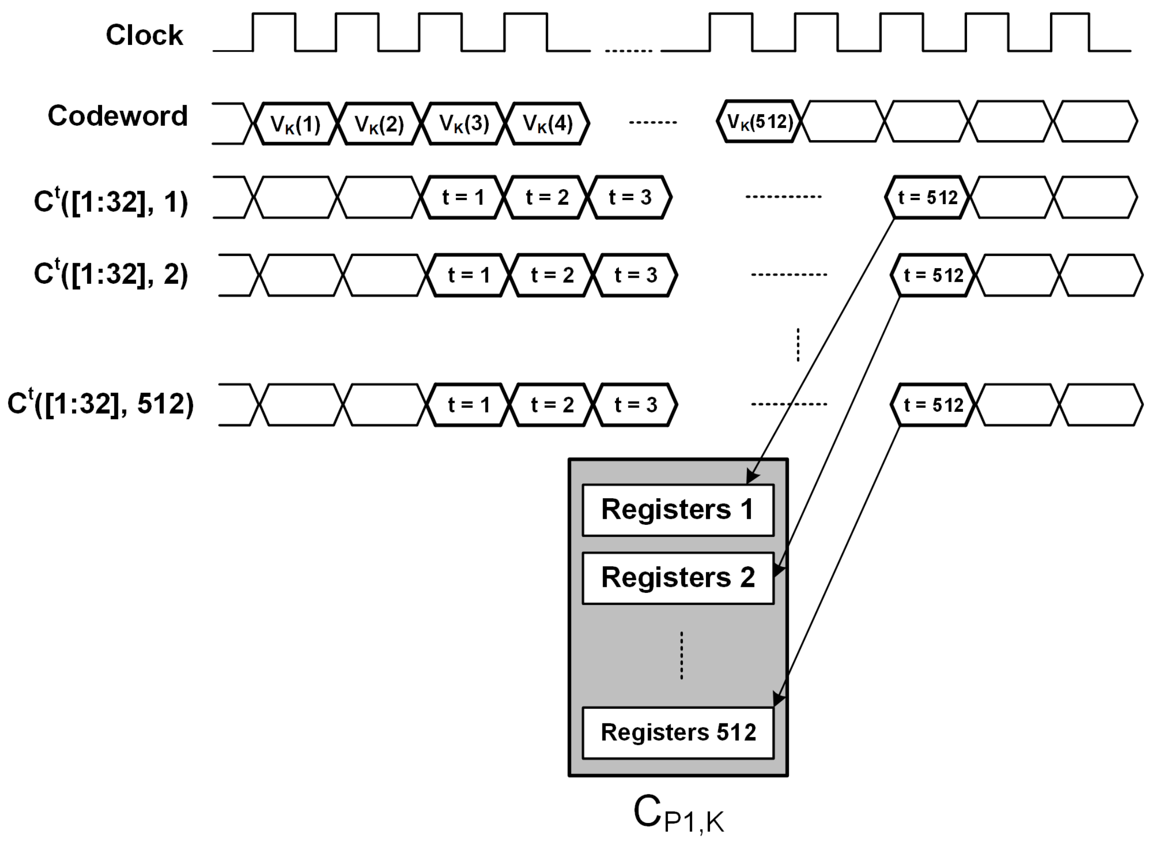

Figure 4.7: Timing diagram of the parity generation.

$$\begin{aligned}\boldsymbol{c}^t([1:32],q) &= \sum_{i=1}^{t} \boldsymbol{d}(1,q) \times \mathbf{P}_a(i,[1:32]) \\ &= \boldsymbol{c}^{t-1}([1:32],q) + \boldsymbol{d}(1,q) \times \mathbf{P}_a(i,[1:32])\end{aligned} \tag{4.4}$$

At $t = 512$, the parity bits $\boldsymbol{c}([1:32],[1:512])$ are completely computed. Then, $\mathbf{C}_{p1,k+1}$ become $\mathbf{C}_{p1,k}$ which are stored in 512 32-bit registers as illustrated in Figure 4.7. The value of $\mathbf{C}_{p1,k}$ registers will update again when the next staircase block is processed. The staircase encoder processes one row in the staircase block at each clock cycle. This requires 512 clock cycles to complete encoding of one staircase block.

4.2.5 ROMs

P_a and P_b^T matrices are stored in the ROMs, which are hard-coded in the Verilog implementation of the staircase encoder instead of using peripheral look-up tables. The reason for this is to make the ROMs compatible with the Verilog synthesizer that we used to obtain more accurate synthesis results. The memory of P_b^T shown in Figure 4.8 consists of 32 ROMs with each contains 478 bits and connected directly to the fixed location on the Bit-Matrix Multiplier. For example, the content of ROM $P_b^T(1)$ is connected to the first Parity Multiplier, therefore, the bit-matrix multiplication can be applied directly to the inputted information vector with the $P_b^T(1)$. P_a is stored in a 16,384 bits ROM with 512 read addresses and 32 bits per line. The read address is provided by the staircase encoder Controller based on the index of the row of the current staircase block which is processed in the Parity Generation Units. The ROMs for P_a and P_b^T matrices are combinational logic circuits, and the power consumption is minimal.

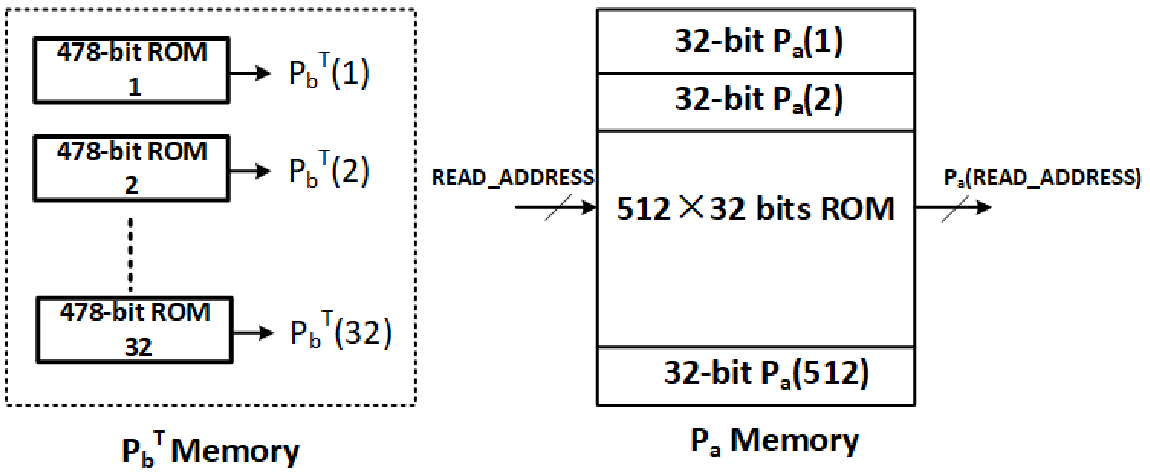

Figure 4.8: The P_a and P_b^T ROMs.

4.2.6 $\mathbf{C}_{p1,k}$ Registers and Multiplexer

The computed parity bits matrix $\mathbf{C}_{p1,k}$ is stored in 512 32-bit Registers. Each 32-bit Register stores one column of the partial parity bits $\mathbf{C}_{p1,k}$ matrix. Although registers occupy the larger area in the physical layout of VLSI circuits compare to RAM, registers store and access data faster. Registers can parallelly access data stored in various locations in the registers. Since the bits of $\mathbf{C}_{p1,k}$ are transmitted from Parity Generation Units in parallel, 512 32-bit data need to be stored in parallel in the memory at the same time. In this case, the registers are applied for the purpose of reducing process time and allowing the parallel access to the partial parity bits $\mathbf{C}_{p1,k}$.

The staircase encoder Controller sends enable signal to the $\mathbf{C}_{p1,k}$ Registers to control when to update the content of the registers. The outputs of $\mathbf{C}_{p1,k}$ registers are connected to a Multiplexer (MUX) and the connection orders are based on the permutation function. For example, the output of Register 3 is connected to the input position 10 of the MUX. Therefore, the value of selection signal (MUX_sel) of MUX can be determined by up counter (value of selection signal is from 0 to 511) for the corresponding row index of the previous staircase block. The connection

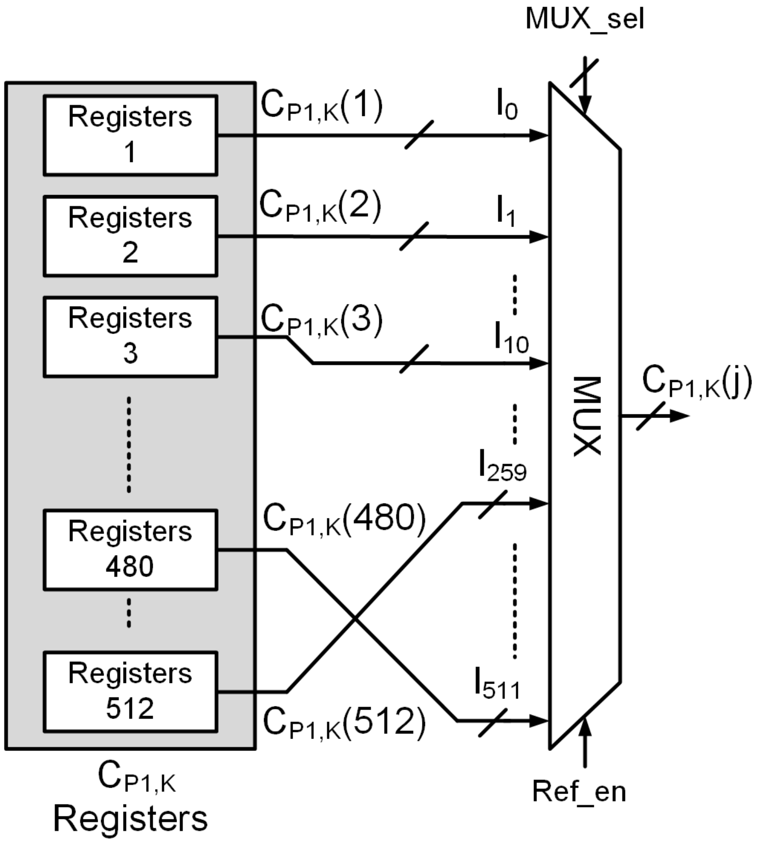

Figure 4.9: The connection between $\mathbf{C}_{p1,k}$ Registers and MUX.

between $\mathbf{C}_{p1,k}$ Registers and the MUX is illustrated in Figure 4.9. When the first information block is being processed, the reference enable signal (Ref_en) of the MUX selects reference parity bits (all-zeros) as $\mathbf{C}_{p1,k}$. The output $\mathbf{C}_{p1,k}(j)$ from the MUX is transmitted to the Bit-Matrix Adder which is added with $\mathbf{C}_{p2,k}(i)$ to compute parity bits $\mathbf{C}_K(i)$ for the information vector $\mathbf{A}_K(i)$.

4.2.7 Staircase Encoder Controller

The staircase encoder Controller schedules the encoding process by controlling the processing sequences of the modules of the encoder. The Controller is designed to have minimal instructions to save power. Figure 4.10 depicts the architecture of the staircase encoder Controller. The Controller contains Block Counter, Codeword

Counter, Information Buffer Controller, P_a ROM Controller, MUX Controller and a Finite State Machine.

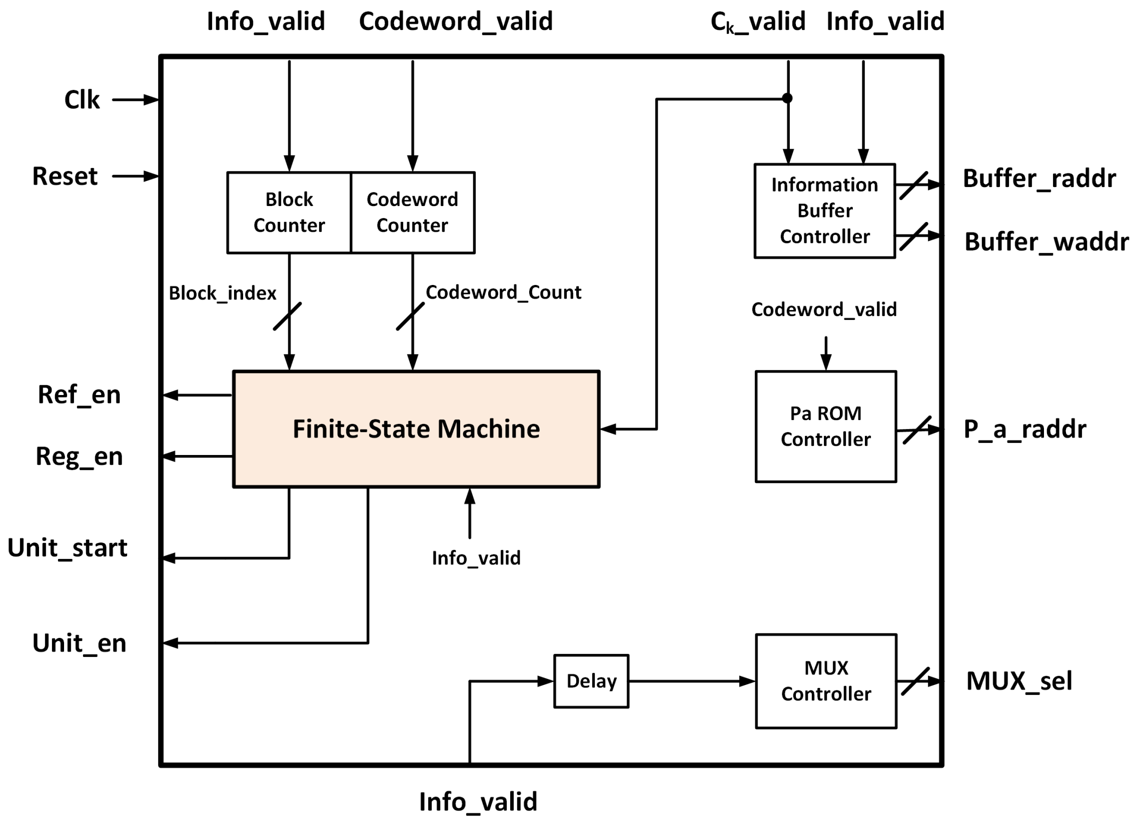

Figure 4.10: The staircase encoder Controller.

The Block Counter keeps track of the staircase block index by counting the inputted information rows through info_valid signal. When the counted info_valid signal presence equals to 512, the value of block count signal block_index gets incremented by one.

The Codeword Counter keeps the count (Codeword_count) of the codewords being processed in the staircase block through Codeword_valid. The two signals Block_index and Codeword_Count are inputted to the Finite State Machine, alongside the signals Info_valid and $\mathbf{C}_k$_valid.

The Information Buffer Controller outputs read address (Buffer_waddr) and write address (Buffer_raddr) for the Buffer based on the Info_valid and $\mathbf{C}_k$_valid signals.

When there are information vectors present (Info_valid=1), Buffer_waddr starts to increment to let the Buffer store the vectors. After the computation of parity bits ($\mathbf{C}_k$_valid=1) for the first information vector, the Buffer_raddr starts to increment to signal the Buffer to output the stored information vectors to the Data Formatter. The depth of the Buffer is 3 which is selected to be the same as the latency between the information vector and its corresponding parity bits.

The P_a ROM Controller determines read address (P_a_raddr) for accessing the contents of P_a ROM. When the codeword is computed (Codeword_valid=1), the Parity Generation process starts. Therefore P_a ROM Controller begins to increment P_a_raddr which value is the same as the index of the current codeword being processed and obtained by Codeword_valid signal. The value of the read address of P_a ROM is reset to 0 once all the codewords (512) in one block are processed.

The MUX Controller outputs the inputs selection signal (MUX_sel) for the Multiplexer. The value of MUX_sel is determined by signal Info_valid (delayed by two clock cycles). The Info_valid is delayed in order to be aligned with the beginning of parity bits $\mathbf{C}_k$ computation. The MUX controller uses the number of occurrences of Info_valid as the value of MUX_sel. The MUX_sel is transmitted to the Multiplexer to select the corresponding outputs from $\mathbf{C}_{p1,k}$ Registers. Since the $\mathbf{C}_{p1,k}$ are already connected to the MUX in a permutated order, the partial parity bits $\mathbf{C}_{p1,k}(j)$ can be selected directly for the current $\mathbf{C}_{p2,k}(i)$. The value of MUX_sel is also being reset to 0 when the information vectors in one block are processed.

The last critical component of the staircase encoder Controller is the Finite State Machine which generates control signals for different states. The Finite State Machine of the Controller has four states which are Start, Parity Bits Computation, Codeword Computation and Parity Generation. Figure 4.11 shows the states and their transitions of the Finite State Machine. Figure 4.11 also illustrated the value of the control signals of each state for the state input. The descriptions of control

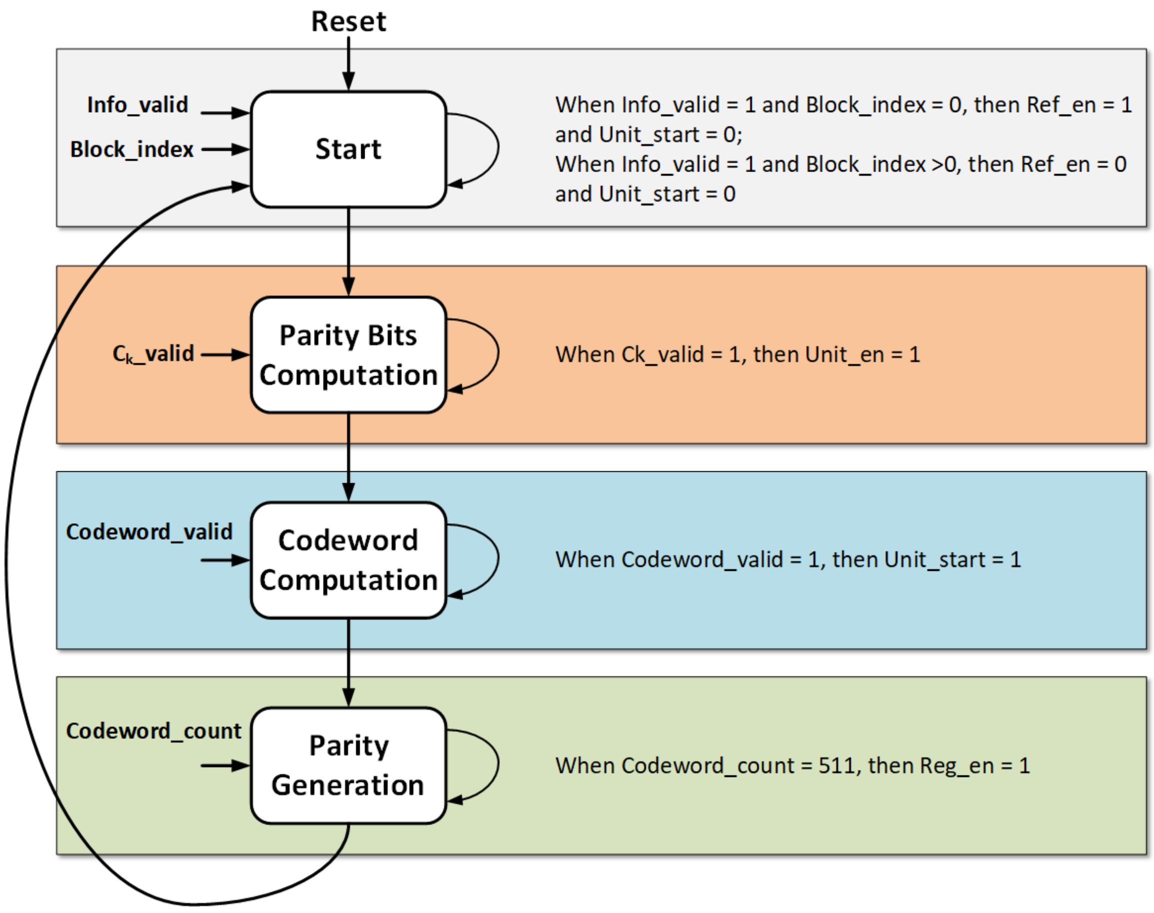
Reset
Info_valid
Block_index
Start
When Info_valid = 1 and Block_index = 0, then Ref_en = 1 and Unit_start = 0;
When Info_valid = 1 and Block_index >0, then Ref_en = 0 and Unit_start = 0
Ck_valid
Parity Bits Computation
When Ck_valid = 1, then Unit_en = 1
Codeword_valid
Codeword Computation
When Codeword_valid = 1, then Unit_start = 1
Codeword_count
Parity Generation
When Codeword_count = 511, then Reg_en = 1

Figure 4.11: The Finite State Machine of the Controller.

signal outputs of the Finite State Machine are listed in Table 4.1. The Start state is executed when the encoder starts the encoding process of a new information block. Therefore, when the Information vector is inputted (Info_valid=1), the next state is Parity Bits Computation, otherwise, the next state is still the Start. The output control signals of this state are Ref_en and Unit_start. The block_index determines the value of Ref_en, and Info_valid determines the results of Unit_start

At the beginning of parity bits $\mathbf{C}_k$ computation, the state Parity Bits Computation is executed. When the $\mathbf{C}_k$ is computed ($\mathbf{C}_k$_valid=1), the next state is Codeword Computation. If there are no valid parity bits, the state stays at Parity Bits Computation until valid parity bits are computed. This state has an output control signal

Table 4.1: The output control signals of the Finite State Machine

Control Signal	**Value**	**Description**
Ref_en	0	The partial parity bits are selected from $\mathbf{C}_{p1,k}$ Registers.
	1	All zero parity bits are selected as partial parity bits.
Unit_start	0	The Units process the first codeword row of a block.
	1	The Units process the rest codeword rows of a block.
Unit_en	0	Keeps value of the registers of the Units.
	1	Updates value of the registers of the Units.
Reg_en	0	Keeps value of the $\mathbf{C}_{p1,k}$ Registers.
	1	Updates value of the $\mathbf{C}_{p1,k}$ Registers.

Unit_en which value is based on the $\mathbf{C}_k$_valid.

The state Codeword Computation is executed at the beginning of codeword concatenation (information vector and parity bits). When the first codeword of the staircase block is ready (Codeword_valid=1), the next state is the Parity Generation. If the valid codeword is not present, the next state is still Codeword Computation. Unit_start is the output control signal of this state, the value of Unit_start gets updated for the value of Codeword_valid.

The last state is the Parity Generation which involves $\mathbf{C}_{p1,k+1}$ parity bits computation and updating values of the $\mathbf{C}_{p1,k}$ Registers. This state gets executed when the first codeword is computed. Codeword_count is used to keep track of the number of processed codeword rows of one staircase block. When the codeword count reaches 512, the next state is the first state Start. Otherwise, the state stays at the Parity Generation. The control signal output of this state is Reg_en which value is updated based on the value of Codeword_count.

The Finite State Machine runs all four states to complete the encoding process of one staircase block. When the next information block is inputted, the finite state machine will execute from the state Start to the state Parity Generation again. The main functionality of the finite state machine is to control the parts of Parity Generation Units and $\mathbf{C}_{p2,k}(i)$ Registers.

4.3 Power Reduction of the Staircase Encoder

Low power is one of the main design considerations for the staircase encoder VLSI implementation, and it is crucial to find a suitable power reduction method for the encoder. There are two types of power consumption in VLSI circuit, dynamic (switching) and static (leakage) power. The main part of dynamic power is dissipated through charging and discharging the load capacitors of the circuits. Therefore, the dynamic power consumption is proportional to the switching activities in VLSI circuits [26]. The static power is dissipated through leakage current which is continuous regardless of the switching activities of the circuits. In terms of circuits system-level design, the feasible power reduction method is to reduce the dynamic power consumption. Since it is more effective to find ways to reduce the power of VLSI circuits in the RTL design level instead of the implementation/physical design level. The most common techniques to reduce the dynamic power are clock gating and reducing clock frequency. Since the high throughput aspect of the staircase encoder is critical, the clock frequency reduction of the encoder is limited. The more pertinent approach to reduce the power consumption is through clock gating.

The clock gating technique reduces the switching activity of the circuits by stopping clock signals in certain regions of the design. The clock gating is a common method to reduce the power in sequential circuits [27], because the dynamic power is only consumed when switching activity occurs. By stopping the clock of the flip-flops prohibited them to switch values, the dynamic power consumption of the flip-flops goes to zero. The staircase encoder has a large number of registers located in the Parity Generation Units and the $\mathbf{C}_{p1,k}$ Registers. Since the registers in the Parity

Generation Units are updated constantly during staircase encoding, the power reduction through clock gating on these registers is minimal. However, the values of the $\mathbf{C}_{p1,k}$ registers do not require to update constantly. The valid value of $\mathbf{C}_{p1,k+1}$ is computed once all 512 rows of the current staircase block are processed, therefore the $\mathbf{C}_{p1,k}$ registers need to update their value only once at the end of the encoding process. The clock signals of $\mathbf{C}_{p1,k}$ registers are enabled accordingly, thus reducing the switching activity of the staircase encoder.

The Efficient front-end HDL coding style also contributes to the power reduction of the staircase encoder. The Verilog codes of the encoder are designed to avoid unnecessary data transitions by having the default value of the signals. The Counters in the staircase encoder Controller are designed to have start and stop counting commands to avoid the Counters keep counting after certain conditions are met.

Chapter 5

Simulation and Synthesis Results

In this chapter, the simulation and synthesis results of the proposed staircase encoder design are presented. The testing setup to verify the outputs of staircase encoder hardware implementation is discussed. The simulation waveforms of the encoder are presented. The power consumption, area, throughput and operating frequency of the staircase encoder are listed in the synthesis results.

5.1 Testbench Setup

The testbench is applied to verify the outputs from the staircase encoder Verilog implementation. The codeword outputs generated from the Matlab version of the staircase encoder are applied as reference output for the Verilog version of the encoder. The information vectors generated from the Matlab version are used as the stimulus to the testbench. There are 50 information blocks which equivalent to 25,600 478-bit information rows to input to the encoder. The testbench reads from the information blocks one row per clock cycle. Information valid signal (info_valid) is inputted along with the information vector to the encoder by the testbench. The testbench implements a self-checking technique which auto compares the Verilog generated codewords with the pre-computed reference codewords (codeword_expected), if there are differences occur, the signal codeword_errors will increment.

5.2 Simulation Waveforms

The Verilog implementation of the staircase encoder is simulated using Modelsim SE. Figure 5.1 shows the waveform of the encoder at the beginning of the first block encoding process. Figure 5.2 presents the transitions of the signals of the staircase encoder during the encoding of the first and second information blocks. Figure 5.3 illustrates the encoding of the first 3 blocks. Finally, the encoding of all the 50 blocks is presented in Figure 5.4. The simulation output results (50 staircase blocks) of the staircase encoder agree with the reference results.

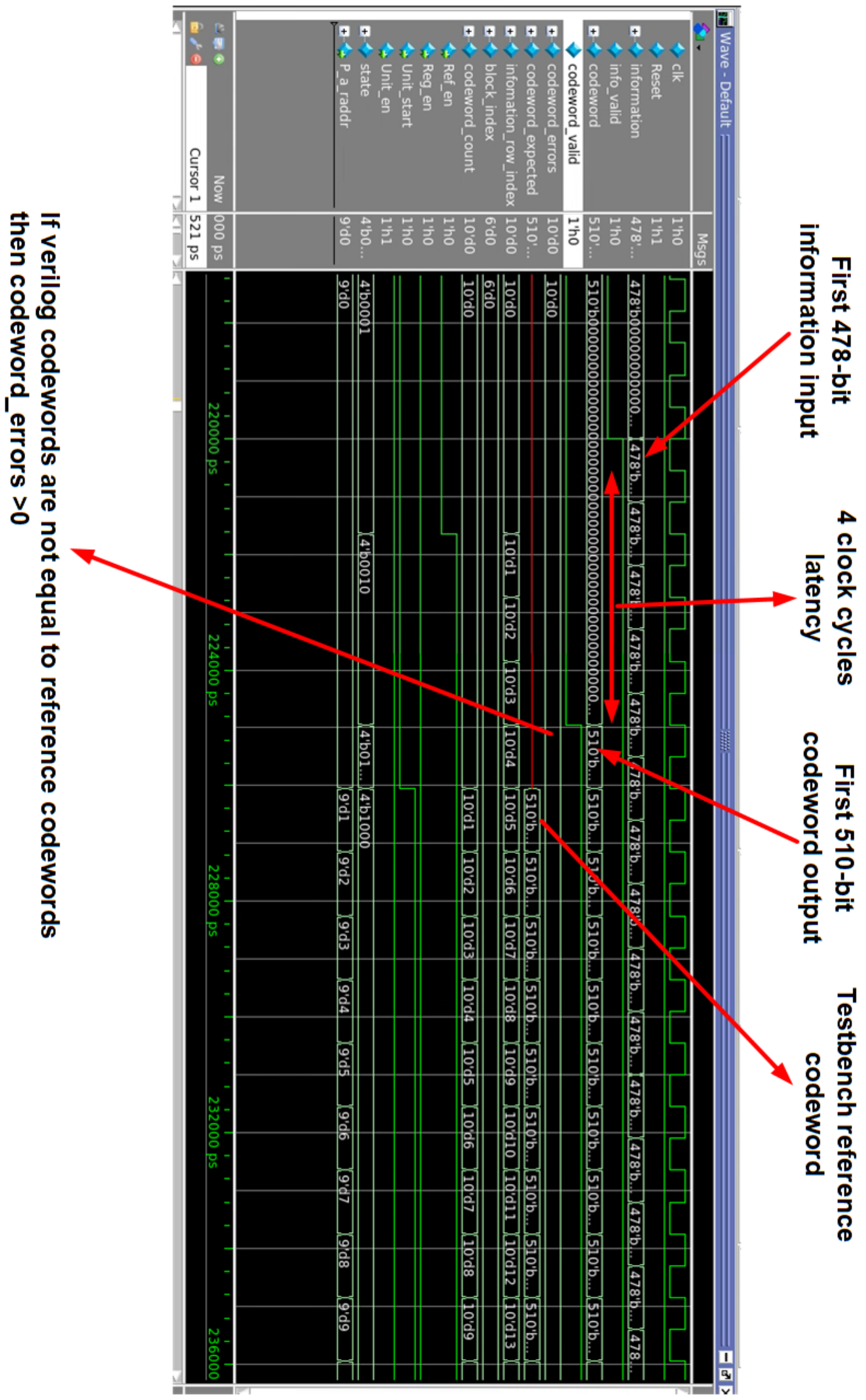

Figure 5.1: At the beginning of the first block encoding.

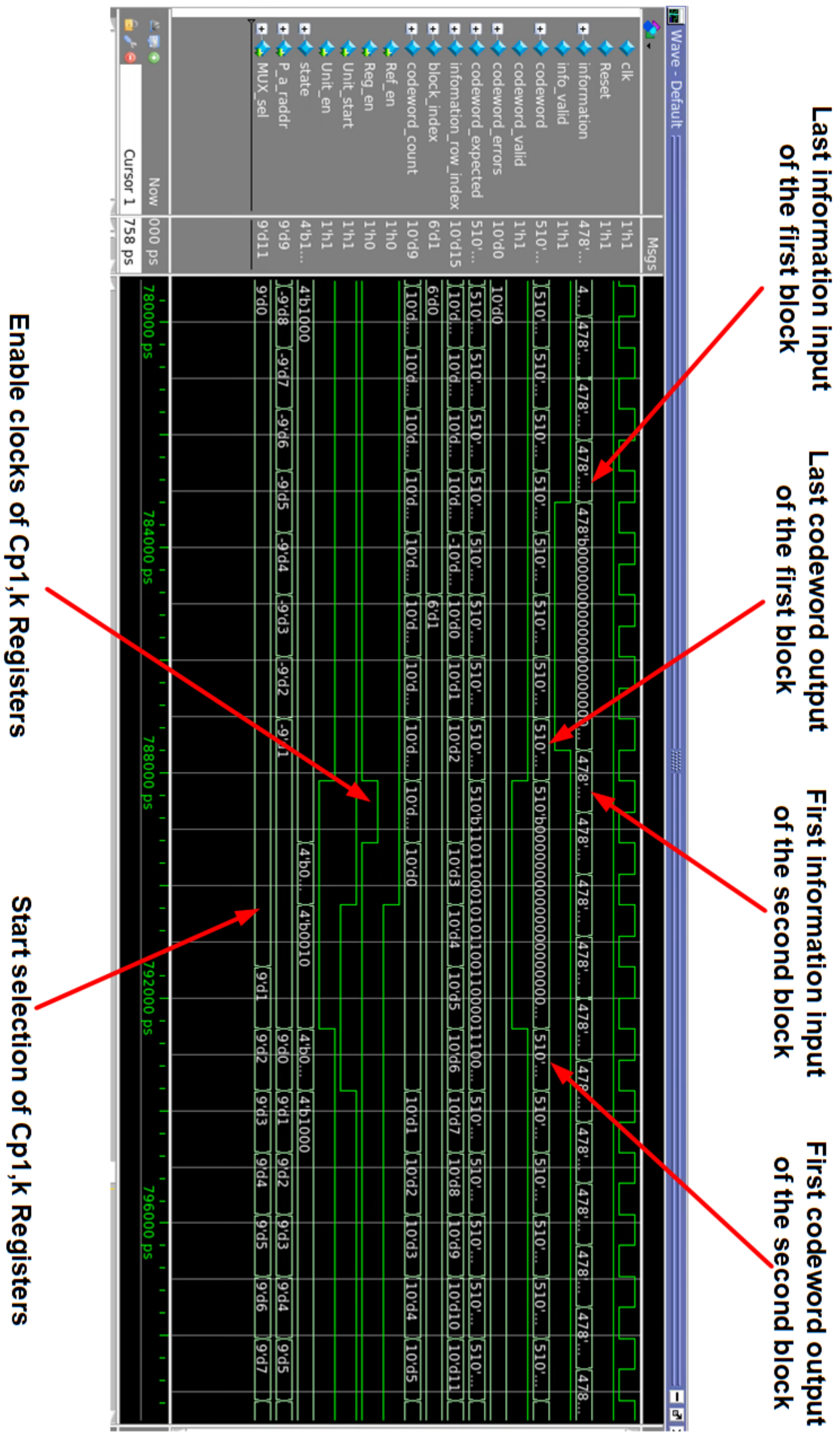

Figure 5.2: Between the first and the second block.

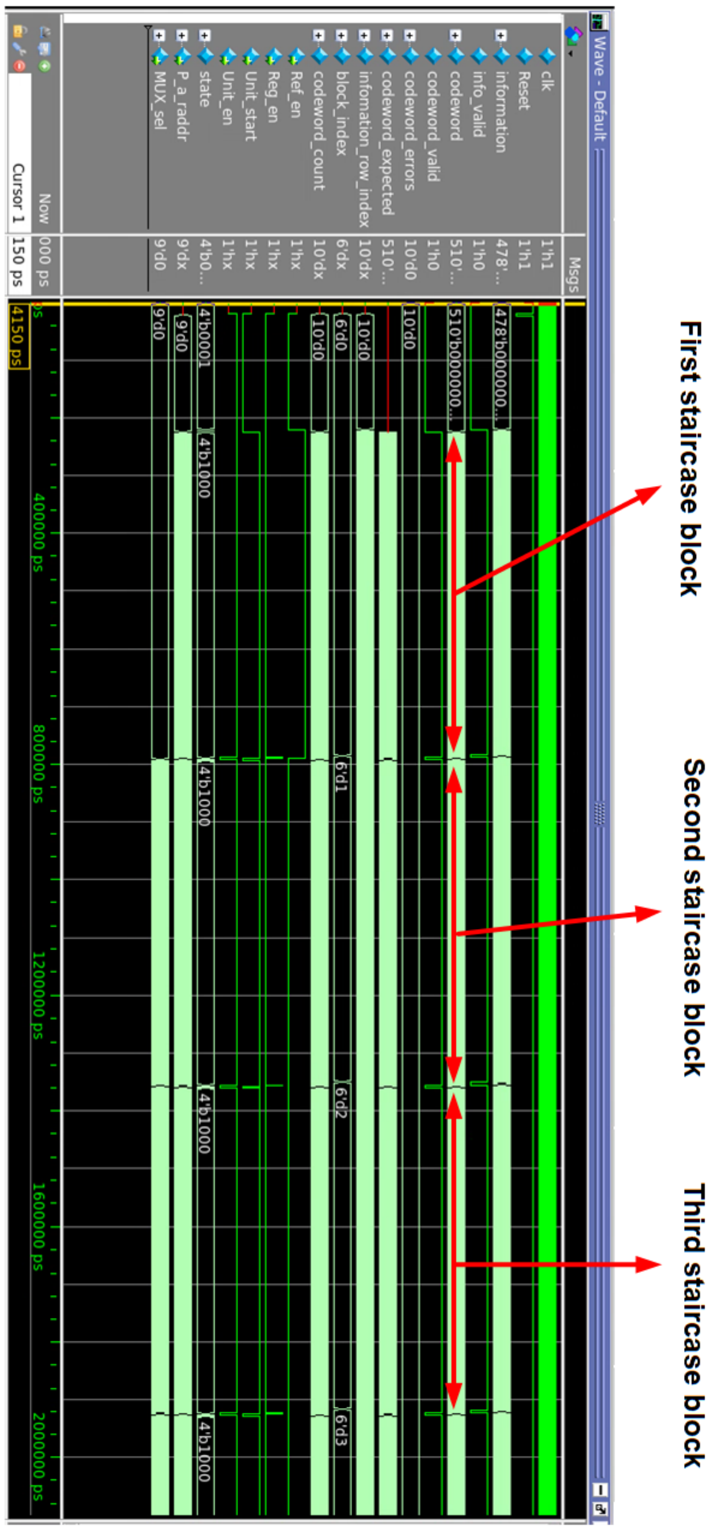

Figure 5.3: Simulation of 3 staircase blocks.

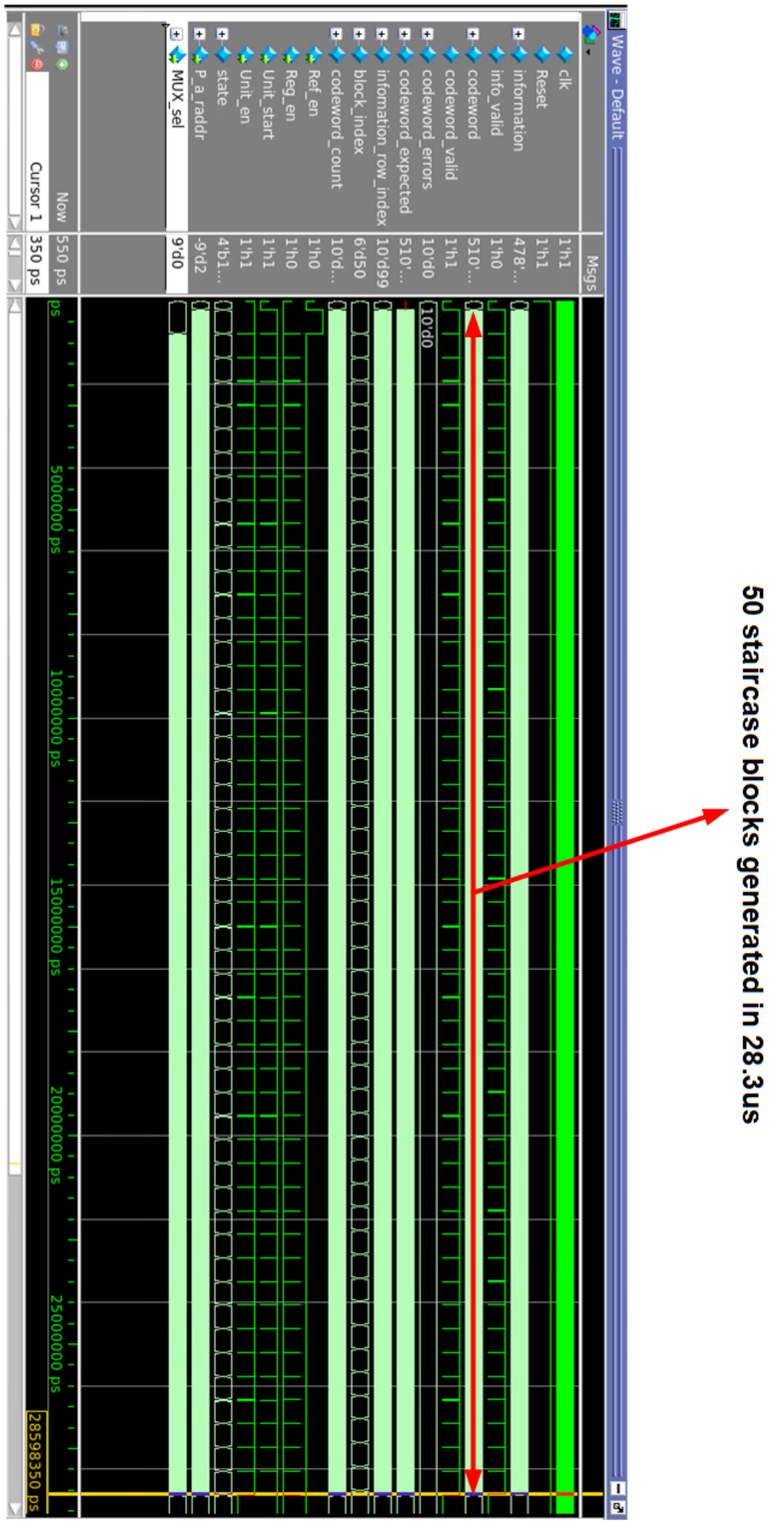

Figure 5.4: Simulation of 50 staircase blocks.

5.3 Synthesis Results

The Verilog implementation of the proposed staircase encoder was synthesized using Synopsys Design Compiler using a TSMC 65nm standard cell library. The target clock rates are set to 500MHz, 909MHz and 1.25GHz, respectively. The static timing analysis is applied to the synthesized staircase encoder to check the setup and hold timing violations. The encoder passed the timing analysis for the three clock frequencies. Table 5.1 presents the simulation results of the staircase encoder implementation. The synthesis results of the components of the proposed staircase encoder design at 909MHz are listed in Table 5.2.

Table 5.1: Synthesis Results of Proposed Staircase Encoder

Frequency	500MHz	909MHz	1.25GHz
Throughput	237Gbps	432Gbps	595Gbps
Power	193mW	323mW	435mW
Latency	8ns	4ns	3ns
Latency	4 clock cycles		
Area	0.587mm^2		
Gate Count	407,916		

Table 5.2: Synthesis Results of Staircase Encoder Blocks @909MHz

Blocks	**Power**	**Power Percent**	**Area**	**Area Percent**
Parity Generation Units and C$_{p1,k}$ registers	261mW	80.8%	0.446mm^2	76.0%
Controller	5mW	1.5%	0.009mm^2	1.6%
Bit-matrix multiplier	21mW	6.5%	0.073mm^2	12.4%
Information buffer	17mW	5.3%	0.022mm^2	3.8%
Total	323mW	100%	0.587mm^2	100%

Table 5.3: Comparison of Staircase Encoder Architectures @909MHz

Architectures	**Storing column parity bits**	**Storing the previous staircase block**
Throughput	432Gbps	432Gbps
Power	0.323W	1.914W
Area	0.587mm^2	2.813mm^2
Latency	4ns	4ns
Gate Count	407,916	1,953,472

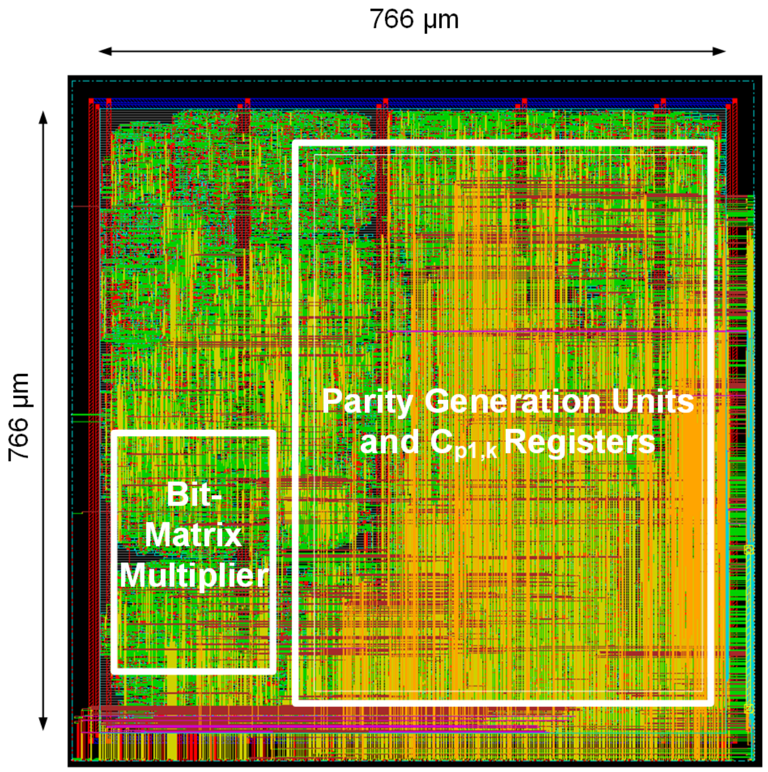

Figure 5.5: Layout of the proposed staircase encoder.

The total power consumption of the staircase encoder is 323mW when operating at 909MHz, and the encoder can achieve the throughput of 432Gbps. The Parity Generation Units of $\mathbf{C}_{p1,k+1}$ and $\mathbf{C}_{p1,k}$ registers consume 80.9% of the total power, approximately 261mW. These parts occupy 75.8% of the total area of the encoder design, and the total power consumption is reduced by 16% with clock gating implemented at $\mathbf{C}_{p1,k}$ registers.

Figure 5.5 shows the proposed staircase encoder layout produced from Cadence Encounter using 65nm CMOS standard cells. Table 5.3 shows simulation results comparison between different staircase encoder architectures. The first architecture is our proposed design which only stores the column parity bits of the previous staircase block. The second architecture stores the entire previous staircase block with the proposed parallel partial parity matrix pre-computing. From the results, we can see that by only storing the column parity bits, the power is reduced by 6 times and the area is reduced by 4.8 times, respectively.

Chapter 6

Conclusion

6.1 Conclusion

1. In this thesis, the BCH and staircase encoding process are presented based on ITU-T recommendation G.709.2/Y.1331.2 and OIF-400ZR implementation agreement.

2. The high-throughput and low-power VLSI design for a staircase encoder suitable for the 400G FEC system based on the standards is implemented. The high parallel level of the encoder design and operation, as well as pre-computing partial parity bits, ensure high throughput, low latency and memory overhead.

3. The staircase encoder is synthesized using 65nm CMOS technology, and the effective clock gating technique significantly reduces the power consumption of the encoder. The throughput of staircase encoder can reach 432Gbps @909MHz with the 323mW power consumption, and the latency of the encoder is four clock cycles.

6.2 Future Work

1. The control logic of the staircase encoder can be further optimized to reduce power consumption. Since some commands of the encoder Controller can be combined.

2. VLSI implementation of staircase decoder which suits the OIF 400ZR implementation agreement. The implemented staircase decoder and staircase encoder form a complete VLSI implementation of the staircase coding system.

3. FPGA emulations of staircase encoder and decoder to test the performance of the staircase codes system on the real hardware environments.

Bibliography

[1] International Telecommunication Union, *Recommendation ITU-T G.709.2/Y.1331.2 OTU4 long-reach interface.* 2018.

[2] M. H. Eiselt, A. Dochhan, and J. Elbers, "Data center interconnects at 400g and beyond," in *2018 23rd Opto-Electronics and Communications Conference (OECC)*, pp. 1–2, July 2018.

[3] Optical Internetworking Forum, *Implementation Agreement 400ZR.* No. IA # OIF-400ZR 0.6-Draft, 2018.

[4] B. P. Smith, A. Farhood, A. Hunt, F. R. Kschischang, and J. Lodge, "Staircase codes: FEC for 100 Gb/s OTN," *Journal of Lightwave Technology*, vol. 30, no. 1, pp. 110–117, 2012.

[5] L. M. Zhang, D. Truhachev, and F. R. Kschischang, "Spatially coupled split-component codes with iterative algebraic decoding," *IEEE Transactions on Information Theory*, vol. 64, pp. 205–224, Jan 2018.

[6] P. Elias, "Error-free coding," *IRE Transactions on Information Theory*, vol. PGIT-4, pp. 29–37, 1954.

[7] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. on Inf. Theory*, vol. 27, pp. 533–547, Sept. 1981.

[8] M. Lentmaier, A. Sridharan, D. J. Costello, Jr., and K. Sh. Zigangirov, "Iterative decoding threshold analysis for LDPC convolutional codes," *IEEE Trans. on Inf. Theory*, vol. 56, pp. 5274–5289, Oct. 2010.

[9] S. Kudekar, T. J. Richardson, and R. L. Urbanke, "Threshold saturation via spatial coupling: why convolutional LDPC ensembles perform so well over the BEC," *IEEE Transactions on Information Theory*, vol. 57, pp. 803–834, Feb. 2011.

[10] G. Hu, J. Sha, and Z. Wang, "Beyond 100Gbps encoder design for staircase codes," *IEEE Workshop on Signal Processing Systems, SiPS: Design and Implementation*, pp. 154–157, 2016.

[11] C. Fougstedt and P. Larsson-Edefors, "Energy-efficient high-throughput staircase decoders," in *2018 Optical Fiber Communications Conference and Exposition (OFC)*, pp. 1–3, March 2018.

[12] C. Fougstedt and P. Larsson-Edefors, “Energy-efficient high-throughput VLSI architectures for product-like codes,” *Journal of Lightwave Technology*, vol. 37, no. 2, pp. 477–485, 2019.

[13] Y. Cai, W. Wang, W. Qian, J. Xing, K. Tao, J. Yin, S. Zhang, M. Lei, E. Sun, H. Chien, Q. Liao, K. Yang, and H. Chen, “Fpga investigation on error-flare performance of a concatenated staircase and hamming fec code for 400g inter-data center interconnect,” *Journal of Lightwave Technology*, vol. 37, pp. 188–195, Jan 2019.

[14] L. Zhang and F. R. Kschischang, “Low-complexity soft-decision concatenated ldgm-staircase fec for high-bit-rate fiber-optic communication,” *Journal of Lightwave Technology*, vol. 35, pp. 3991–3999, Sep. 2017.

[15] M. Barakatain and F. R. Kschischang, “Low-complexity concatenated ldpc-staircase codes,” *Journal of Lightwave Technology*, vol. 36, pp. 2443–2449, June 2018.

[16] J. Moreira and P. Farrell, *Essentials of Error-Control Coding.* Wiley, 2006.

[17] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms.* New York, NY, USA: Wiley-Interscience, 2005.

[18] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near shannon limit error-correcting coding and decoding: Turbo-codes. 1,” in *Proceedings of ICC ’93 - IEEE International Conference on Communications*, vol. 2, pp. 1064–1070 vol.2, May 1993.

[19] G. Agrawal, *Fiber-Optic Communication Systems.* Wiley Series in Microwave and Optical Engineering, Wiley, 2012.

[20] S. Lin and D. Costello, *Error Control Coding: Fundamentals and Applications.* Pearson-Prentice Hall, 2004.

[21] H. Burton, “Inversionless decoding of binary bch codes,” *IEEE Transactions on Information Theory*, vol. 17, pp. 464–466, July 1971.

[22] C. Yang, Y. Emre, and C. Chakrabarti, “Product code schemes for error correction in mlc nand flash memories,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, pp. 2302–2314, Dec 2012.

[23] N. ul Hassan, M. Lentmaier, and G. P. Fettweis, “Comparison of ldpc block and ldpc convolutional codes based on their decoding latency,” in *2012 7th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, pp. 225–229, Aug 2012.

[24] J. Yu and J. Zhang, “Single-carrier 400g transmission and digital signal processing,” in *2015 Opto-Electronics and Communications Conference (OECC)*, pp. 1–3, June 2015.

[25] I. Lyubomirsky, J. Riani, B. Smith, S. Bhoja, I. Corp, R. Baca, M. Corp, B. Booth, and R. Yu, “Baseline Proposal for 400G / 80km,” 2018.

[26] N. Chabini and W. Wolf, “Reducing dynamic power consumption in synchronous sequential digital designs using retiming and supply voltage scaling,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, pp. 573–589, June 2004.

[27] Qing Wu, M. Pedram, and Xunwei Wu, “Clock-gating and its application to low power design of sequential circuits,” *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, pp. 415–420, March 2000.