Hindawi
Journal of Sensors
Volume 2021, Article ID 6699555, 13 pages
https://doi.org/10.1155/2021/6699555

*Research Article*

# Hardware Decoding Accelerator of (73, 37, 13) QR Code for Power Line Carrier in UPIoT

**Jiye Huang,[1] Shanggang Xie,[1] Tongdong Guo,[1] and Zhijin Zhao[2]**

[1]*School of Electronics and Information, Hangzhou Dianzi University, Hangzhou, China*
[2]*School of Communication Engineering, Hangzhou Dianzi University, Hangzhou, China*

Correspondence should be addressed to Zhijin Zhao; zhaozj03@hdu.edu.cn

Received 25 December 2020; Revised 20 January 2021; Accepted 10 February 2021; Published 17 March 2021

Academic Editor: Kai Huang



The proposal of the ubiquitous power Internet of Things (UPIoT) has increased the demand for communication coverage and data collection of smart grid; the quantity and quality of communication networks are facing greater challenges. This brief applies (73, 37, 13) quadratic residue (QR) codes to power line carrier technology to improve the quality of local data communication in UPIoT. In order to improve the decoding performance of the QR codes, an induction method for the error pattern is proposed, which can divide the originally coupled error pattern into six parts and reuse the same module for decoding. This method greatly reduces the resource requirements, so that (73, 37, 13) QR code can be implemented on FPGA hardware. Notably, the hardware architecture is a modular framework, which can fit into an FPGA with different sizes. As an example (73, 37, 13), QR code is implemented on Intel Arria10 FPGA; the experimental result shows that the maximum decoding frequency of this architecture is 21.7 M Hz, which achieves 4121x speedup compared to CPU. Moreover, the proposed architecture benefits from high flexibility, such as modular design and decoding framework in the form of the pipeline which can be seen as an alternative scheme for decoding long-length QR codes.

## 1. Introduction

With the development of power systems, the types and quantities of electrical equipment are increasing rapidly, for example, Energy Storage (ES) is being widely adopted in the grid to meet the needs of intermittent power generation [1]. This brings a growing demand for information sharing [2] and analysing [3] between all nodes in the power grid system. Ubiquitous power Internet of Things (UPIoT) is proposed as an advanced technology in the construction of smart grids. It can integrate the combination of the Internet of Things (IoT) and the power business, efficiently integrate IoT and power system, and improve the information transparency of the power system. The "Ubiquitous" of UPIoT is embodied in the various nodes of the power system (transmission, transformation, distribution, and consumption), that means the real-time interconnection of people, machinery, power networks, and platforms can be achieved at any time [4].

UPIoT is proposed based on IoT, so it has a structure similar to the traditional IoT, including perception layer, network layer, platform layer, and application layer [5]. The perception layer is composed of sensors with different functions in each link and node of the power network. The network layer is mainly composed of the internet or the dedicated network, which is responsible for a large amount of information transmission. The platform layer is the data sharing and publishing center. The application layer provides users with data display and control signal transmission function. It is recognized that UPIoT mainly includes three parts [6] of communication, local communication, edge IoT gateway, and remote communication. As an intermediate carrier between the perception layer and the platform layer, the edge IoT gateway assumes the function of gathering data and the local edge fast computing. The remote network is usually implemented by 4G/5G and dedicated networks, while the local communication network can choose wired or microwave wireless networks due

to its short distance. The types and quantities of data measured by local sensors in UPIoT will increase greatly with the development of power networks. Therefore, the local transmission network should have the following characteristics:

(i) Two-way real-time communication capability

(ii) Low power consumption and low cost

(iii) Strong data compatibility

Power line carrier communication (PLC) technology, as a local communication technology, can carry both power and data and has been widely used for high-speed data transmission, even in relatively remote areas [7]. The good characteristics of PLC, such as direct application of power line transmission data, no need to modify the wiring layout, easy implementation, and two-way communication, can meet the basic characteristics of the local transmission network. However, using power carrier line leads to a relatively large attenuation of carrier signal, and the damage to performance from multiple noises is still a challenge. In order to deal with the shortcomings, some researches have proposed Reed-Solomon (RS), Low Density Parity Check (LDPC) [8, 9], Convolution Code (CC), and other advanced codes [10, 11]. In addition, using modulation techniques such as Quadrature Frequency Shift keying (QFSK) and Orthogonal Frequency Division Multiplexing (OFDM) [12] can improve the reliability of the system.

The LDPC codes are selected as the channel coding schemes for the data channels in 5G new radios because of its excellent long code error correction performance. In local communication, most of the transmitted signals between the sensors and the IoT gateway are expected to be power status data and control signals, which can be regard as short data; the advantages of Low-Density Parity Check (LDPC) codes cannot reflect due to the dense and short communication data [8]. In recent years, with continuous development, the root-protograph (RP) LDPC code appeared to solve the interruption limitation of the BF channel due to its nontraversal characteristics [9]. The CC decoding has high computational complexity due to the Viterbi algorithm and the use of interleave causes significant end-to-end delay. Reed-Solomon (RS) codes, which are used in many PLCs, will cause a huge amount of calculation when the redundancy is large. Due to the insufficient error correction capability of the channel encoder, the performance of the receiver still cannot achieve satisfactory performance [10]. Therefore, we propose to use quadratic residue (QR) codes [13] to encode data and control signals in front of the PLC channel, and still can use more commonly used modulation methods such as OFDM; it can obtain the following advantages:

(i) The requirement of transmission power can be reduced in the same environment

(ii) Improve the reliability of communication and thereby improve communication efficiency

The QR code gains powerful error correction capabilities by increasing the complexity of decoding; although the QR code decoding algorithm has been improved several times, it is still difficult to make a stand out in practical applications due to its unsatisfactory decoding speed. Based on the existing decoding algorithm, difference syndrome (DS) algorithm [14], and optimized decoding algorithm cyclic weight (OCW) [15], we propose a method suitable for digital circuit hardware implementation, which is named as error pattern induction (EPI). By using EPI, we reduce the (73, 37, 13) QR code into 6 parts so that it can use same pipeline processor, and (73,37, 13) QR code can be realized on limited hardware resources. At the same time, we implemented the hardware decoding architecture of (73, 37, 13) QR code on the Intel Arria10 10AX115-U4F45I1SG FPGA platform, achieving a maximum clock frequency (fmax) of 260.42 MHz, which is equivalent to 21.7 MHz decoding frequency.

## 2. QR Code and Decoding Algorithm

The QR code was proposed by E. Prange in 1958 and has excellent error detection and correction capability due to its large minimum Hamming distance [3]. The QR code is defined on the finite field $GF(2^m)$, and $GF(2^m)$ is a finite field containing $2^m$ elements. The (73, 37, 13) QR code is constructed over $GF(2^9)$, and its quadratic residue set is given by

$$Q_{73} = \left\{ l \mid l \equiv x^2 \mod 73, 1 \le x \le \left[\frac{73}{2}\right] \right\},$$

$$= \left\{ \begin{array}{c} 1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 19, 23, 24, 25, \\ 27, 32, 35, 36, 37, 38, 41, 46, 48, 49, 50, \\ 54, 55, 57, 61, 64, 65, 67, 69, 70, 71, 72 \end{array} \right\}. \quad (1)$$

Let $m$ be the smallest positive integer which makes $(2^m \mod n) = 1$, and for (73, 37, 13) QR code, $m = 9$. Let $\alpha \in GF(2^9)$ be a root of the primitive polynomial $p(x) = x^9 + x^4 + 1$, then $\alpha$ generates all nonzero elements in the finite field $GF(2^9)$. Obviously, $(2^9 - 1)/73 = 7$, so $\beta = \alpha^7$ is the 73rd root of unity in $GF(2^9)$, and the generator polynomial of the (73, 37, 13) QR code is given by

$$\begin{aligned} g(x) = \prod_{i \in Q_{73}} \left(x - \beta^i\right) &= x^{36} + x^{34} + x^{32} + x^{31} \\ &+ x^{30} + x^{28} + x^{25} + x^{24} + x^{22} + x^{21} + x^{20} \\ &+ x^{19} + x^{18} + x^{17} + x^{16} + x^{15} + x^{14} + x^{12} \\ &+ x^{11} + x^8 + x^6 + x^5 + x^4 + x^2 + x + 1. \end{aligned} \quad (2)$$

The message part of the (73, 37, 13) QR code is a vector of length 37 and can be expressed by a polynomial as $m(x) = \sum_{i=0}^{36} m_i x^i$. Then, the codeword can be expressed as

$$c(x) = m(x) \cdot x^{36} + m(x) \cdot x^{36} \mod g(x) = \sum_{i=0}^{72} c_i x^i, \quad (3)$$

by using systematic coding, where high 37 bits are message bits, expressed as $m(x) \cdot x^{36}$, and lower 36 bits are parity bits, expressed as $m(x) \cdot x^{36} \mod g(x)$. If the codeword is passed

through an additive white Gaussian noise (AWGN) channel, the noise can be expressed as $e(x) = \sum_{i=0}^{72} e_i x^i$, then the received codeword can be expressed as

$$r(x) = c(x) + e(x) = \sum_{i=0}^{72} r_i x^i, \tag{4}$$

let $s(x) = r(x) \bmod g(x) = \sum_{i=0}^{35} s_i x^i$, and the $s(x)$ is called syndrome polynomial.

We will describe the decoding algorithms by integrating DS algorithm and OCW algorithm; of course, there are other decoding algorithms such as Berlekamp-Massey (BM) [16], Fast Algebraic Decoding Algorithm (ADA) [17], Syndrome and Syndrome Difference Decoding Algorithm (SSDDA) [18, 19], all of them are implemented in software.

For simplicity, split codeword $r$ to a message part $r_m$ and a parity part $r_p$, and split decoded codeword $d_r$ to a message part $dr_m$ and a parity part $dr_p$. Let $e_i$ be an error in the $i^{\text{th}}$ bit of $r_m$, and the syndrome corresponds to $e_i$ is expressed as $s_i$. The syndromes $s_i (0 \le i \le 36)$ is shown in Table 1.

For a received codeword $r$, the syndrome $s_r$ of $r$ is first calculated, then the weight $W(s_r)$ of the $s_r$ is calculated, and the codeword can be decoded by the following steps.

*Step 1.* If $W(s_r) \le 6$, all errors of the received codeword $r$ are in $r_p$. Then, the decoded message part $dr_m = r_m$, and the decoding is completed at this time. If $W(s_r) > 6$, there is at least one error existing in the message part $r_m$.

*Step 2.* Calculate $s_{ri} = s_r \wedge s_i (0 \le i \le 36)$ where '$\wedge$' means the bit operation XOR, and calculate the weight $W(s_{ri})$ of $s_{ri}$. If $i$ exists in the range of $0 \le i \le 36$ such that $W(s_{ri}) \le 5$, there is only one error in $r_m$, and we can obtain the decoded message part $dr_m = r_m \wedge e_i$. Then, the decoding is completed at this time, and the calculation complexity of this step is $C_{37}^1$. If $i$ does not exist such that $W(s_{ri} \le 5)$, there are at least two errors in the message part $r_m$.

*Step 3.* Calculate $s_{rij} = s_r \wedge s_i \wedge s_j (0 \le i < j \le 36)$, and calculate the weight $W(s_{rij})$ of $s_{rij}$. If $i, j$ exist in the range of $0 \le i < j \le 36$ such that $W(s_{rij}) \le 4$, there are two errors in $r_m$, and we can obtain decoded message part $dr_m = r_m \wedge e_i \wedge e_j$. Then, the decoding is completed at this time, and the calculation complexity of this step is $C_{37}^2$. If $i, j$ does not exist which makes $W(s_{rij} \le 4)$, there are at least three errors in $r_m$.

*Step 4.* Calculate $s_{rijk} = s_r \wedge s_i \wedge s_j \wedge s_k (0 \le i < j < k \le 36)$, and calculate the weight $W(s_{rijk})$ of $s_{rijk}$. If $i, j, k$ exist in the range of $0 \le i < j < k \le 36$ such that $W(s_{rijk} \le 3)$, there are three errors in $r_m$, and we can obtain decoded message part $dr_m = r_m \wedge e_i \wedge e_j \wedge e_k$. Then, the decoding is completed at this time, and the calculation complexity of this step is $c_{37}^3$. If $i, j, k$ does not exist which makes $W(s_{rijk} \le 3)$, there are at least four errors $r_m$.

Table 1: Syndromes and error patterns.

| $s_i$ | Error pattern |
|---|---|
| $s_0$ | 1000_1111_0010_0010_1110_1000_1001_1110_0011 |
| $s_1$ | 1001_0001_0110_0111_0011_1001_1010_0010_0101 |
| $s_2$ | 1010_1101_1110_1100_1001_1011_1101_1010_1001 |
| $s_3$ | 1101_0100_1111_1011_1101_1111_0010_1011_0001 |
| $s_4$ | 0010_0110_1101_0101_0101_0110_1100_1000_0001 |
| $s_5$ | 0100_1101_1010_1010_1010_1101_1001_0000_0010 |
| $s_6$ | 1001_1011_0101_0101_0101_1011_0010_0000_0100 |
| $s_7$ | 1011_1001_1000_1000_0101_1110_1101_1110_1011 |
| $s_8$ | 1111_1100_0011_0010_0101_0101_0010_0011_0101 |
| $s_9$ | 0111_0111_0100_0110_0100_0010_1101_1000_1001 |
| $s_{10}$ | 1110_1110_1000_1100_1000_0101_1011_0001_0010 |
| $s_{11}$ | 0101_0010_0011_1011_1110_0011_1111_1100_0111 |
| $s_{12}$ | 1010_0100_0111_0111_1100_0111_1111_1000_1110 |
| $s_{13}$ | 1100_0111_1100_1101_0110_0111_0110_1111_1111 |
| $s_{14}$ | 0000_0000_1011_1000_0010_0110_0100_0001_1101 |
| $s_{15}$ | 0000_0001_0111_0000_0100_1100_1000_0011_1010 |
| $s_{16}$ | 0000_0010_1110_0000_1001_1001_0000_0111_0100 |
| $s_{17}$ | 0000_0101_1100_0001_0011_0010_0000_1110_1000 |
| $s_{18}$ | 0000_1011_1000_0010_0110_0100_0001_1101_0000 |
| $s_{19}$ | 0001_0111_0000_0100_1100_1000_0011_1010_0000 |
| $s_{20}$ | 0010_1110_0000_1001_1001_0000_0111_0100_0000 |
| $s_{21}$ | 0101_1100_0001_0011_0010_0000_1110_1000_0000 |
| $s_{22}$ | 1011_1000_0010_0110_0100_0001_1101_0000_0000 |
| $s_{23}$ | 1111_1111_0110_1110_0110_1011_0011_1110_0011 |
| $s_{24}$ | 0111_0001_1111_1110_0011_1110_1110_0010_0101 |
| $s_{25}$ | 1110_0011_1111_1100_0111_1101_1100_0100_1010 |
| $s_{26}$ | 0100_1000_1101_1010_0001_0011_0001_0111_0111 |
| $s_{27}$ | 1001_0001_1011_0100_0010_0110_0010_1110_1110 |
| $s_{28}$ | 1010_1100_0100_1010_1010_0100_1100_0011_1111 |
| $s_{29}$ | 1101_0111_1011_0111_1010_0001_0001_1001_1101 |
| $s_{30}$ | 0010_0000_0100_1101_1010_1010_1010_1101_1001 |
| $s_{31}$ | 0100_0000_1001_1011_0101_0101_0101_1011_0010 |
| $s_{32}$ | 1000_0001_0011_0110_1010_1010_1011_0110_0100 |
| $s_{33}$ | 1000_1101_0100_1111_1011_1101_1111_0010_1011 |
| $s_{34}$ | 1001_0101_1011_1101_1001_0011_0111_1011_0101 |
| $s_{35}$ | 1010_0100_0101_1001_1100_1110_0110_1000_1001 |
| $s_{36}$ | 1100_0111_1001_0001_0111_0100_0100_1111_0001 |

*Step 5.* Cyclic shift $r$ by 36 bits to left, the result of shifting is expressed as $r'$. Then, calculate the syndrome $s_r{}'$ of $r'$, and calculate the weight $W(s_r{}')$ of $s_r{}'$. Repeat steps 1~4, then 4~6 errors in $r_m$ can be correct. After these, the decoding is completed. We can infer that the total computational complexity of the DS algorithm is $(C_{37}^1 + C_{37}^2 + C_{37}^3) * 2$.

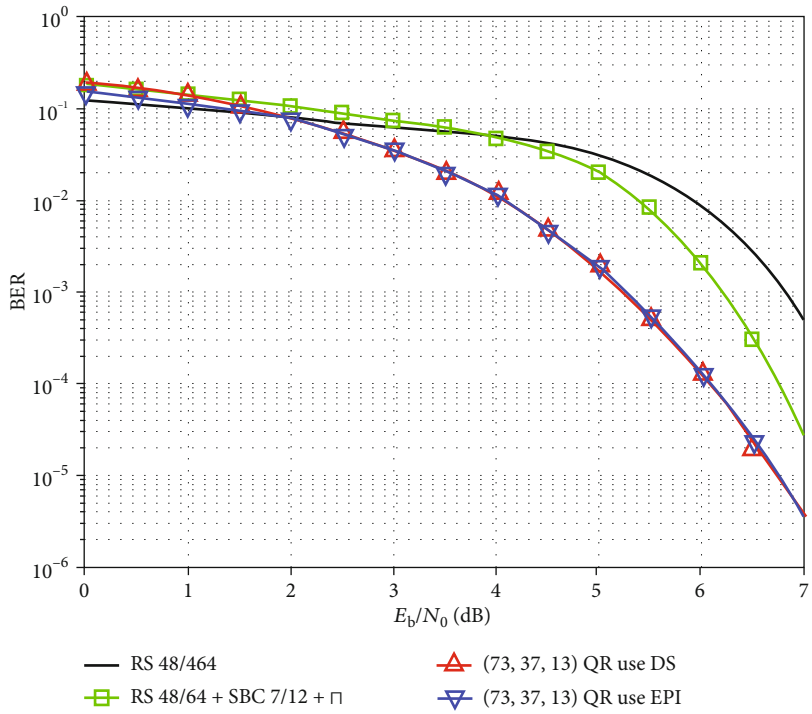


Figure 1: BER performance of the two RS codes, the (73, 37, 13) QR algorithm use DS algorithm in [10] and use EPI in this paper.

The simulation of QR code is indispensable, 37-bit random message is generated and encoded by (73, 37, 13) QR code. Then, calculate the decoding performance after mixing the encoded signal with Gaussian white noise. The simulation results of different bit error rate (BER) performance are shown in Figure 1. We also show the curves of the two RS codes in [20] and the EPI algorithm proposed in this paper. From Figure 1, we can see that QR codes have better performance than RS codes, and QR codes will also have better performance in practical applications. The specific EPI algorithm will be explained in Section 3.

## 3. Error Pattern Induction Method

In order to improve the decoding speed and ensure the controllability of the decoding time, it is necessary to calculate all the decoding possibilities, then the decoding could be completed within a fixed time. However, the long codeword length of (73, 37, 13) QR code leads to high decoding complexity, and decoding with a full parallel module will result in a great amount of resource consumption. This decoding framework summarizes the error patterns to implement module multiplexing based on DS algorithm; then, we can reduce resource consumption and hardware burden, and we can make it possible to implement in low-cost hardware.

*3.1. Error Pattern Induction and Classification.* Section 2 shows that when there are three errors in $r_m$, the decoding complexity is the highest, so we consider to optimize it firstly.

*3.1.1. Three-Error Patterns.* The number of three-error patterns is $c_{37}^3$, a total of 7770 kinds, which can be decomposed into several parts, and we call them as inductive error patterns; the inductive error patterns are shown in Table 2.

Table 2: Three-error inductive error patterns.

| Inductive error patterns | Numbers |
|---|---|
| $e_0e_1e_n(2 \le n \le 35)$ | 34 |
| $e_0e_2e_n(4 \le n \le 34)$ | 31 |
| $e_0e_3e_n(6 \le n \le 33)$ | 28 |
| $e_0e_4e_n(8 \le n \le 32)$ | 25 |
| $e_0e_5e_n(10 \le n \le 31)$ | 22 |
| $e_0e_6e_n(12 \le n \le 30)$ | 19 |
| $e_0e_7e_n(14 \le n \le 29)$ | 16 |
| $e_0e_8e_n(16 \le n \le 28)$ | 13 |
| $e_0e_9e_n(18 \le n \le 27)$ | 10 |
| $e_0e_{10}e_n(20 \le n \le 26)$ | 7 |
| $e_0e_{11}e_n(22 \le n \le 25)$ | 4 |
| $e_0e_{12}e_n(n = 24)$ | 1 |
| Total numbers | 210 |

From Table 2, there are 210 kinds of inductive error pattern. Cyclic shift each pattern to the left by $m(1 \le m \le 36)$ bits so that we can obtain all $210 * 37$ kinds of three-error patterns, and these patterns are further decomposed to the form of 7770.

At this point, the first decomposition is completed, but due to the uneven distribution of the number of errors, the hardware implementation will result in different pipeline lengths, so we need to process the inductive error patterns.

Table 3: Three-error combination.

| Group | Three-error combination patterns | Numbers |
|---|---|---|
| 1 | $e_0e_1e_n(2 \le n \le 35), e_0e_{12}e_n(n = 24)$ | 34 + 1 = 35 |
| 2 | $e_0e_2e_n(4 \le n \le 34), e_0e_{11}e_n(22 \le n \le 25)$ | 31 + 4 = 35 |
| 3 | $e_0e_3e_n(6 \le n \le 33), e_0e_9e_n(18 \le n \le 27)$ | 28 + 7 = 35 |
| 4 | $e_0e_4e_n(8 \le n \le 32), e_0e_9e_n(18 \le n \le 27)$ | 25 + 10 = 35 |
| 5 | $e_0e_5e_n(10 \le n \le 31), e_0e_8e_n(16 \le n \le 28)$ | 22 + 13 = 35 |
| 6 | $e_0e_6e_n(12 \le n \le 30), e_0e_7e_n(14 \le n \le 29)$ | 19 + 16 = 35 |

As is shown in Table 3, we combine the inductive error patterns and divide them into 6 groups to make the patterns evenly distributed, and each combination pattern group has 35 inductive error patterns.

We call the inductive patterns in the same group as combination patterns and implement them in same pipeline. Then, the three-error patterns are decomposed into the form as $C_{37}^3 = 7770 = 6 * 35 * 37$.

*3.1.2. Two-Error Patterns.* The number of two-error patterns is $d\_flag\_\text{in} = 0$, a total of 666 kinds can be decomposed into the 18 kinds of inductive patterns, expressed as $e_0e_n(1 \le n \le 18)$. Then, we cyclically shift each pattern to the left by $m$ $(1 \le m \le 36)$ bits so that we can get all $18 * 36$ kinds of two-error patterns.

Since the three-error patterns are divided into 6 groups, we also combine two-error patterns into 6 groups to maintain the consistency of the pipeline, and each group has 3 two-error inductive patterns. The two-error combination patterns are shown in Table 4.

At this point, the two-error patterns are decomposed into the form as $C_{37}^2 = 6 * 3 * 37$.

*3.1.3. Single-Error Pattern.* The number of single-error patterns is $C_{37}^1$, a total of 37 kinds. It is the minimum error correction unit and can be grouped without decomposing. Since the number 37 cannot be evenly divided, the single-error patterns are divided into 6 groups. Each group has 7 single-error patterns, and no operation is performed in redundant part to ensure the consistency of the pipeline. The single-error division groups are shown in Table 5.

*3.2. Error Pattern Traversal and Decoding Framework.* Unlike the DS algorithm, the priority of the memory footprint optimization in the FPGA application is not high. Therefore, we propose a decoding framework that separates decoding operation into an error pattern traversal part and a codeword decoding part. We use memory cells to reduce register consumption, which is feasible in FPGA designs.

The specific operation of this decoding framework is shown in Section 4. After the preprocessing operation, the codeword $r/r'$ and the flag bit are stored in First Input First Output (FIFO) memory. Then, the codeword $r/r'$ is calculated to obtain the initial syndrome $s_r$, and we can traversal of all error patterns by $s_r$. When traversing, if the corresponding condition is met (see Section 4 for details), the decoding flag bit denoting as '*d_flag*' will be set to 1, and the current

Table 4: Two-error combination patterns.

| Group | Two-error combination patterns | Numbers |
|---|---|---|
| 1 | $e_0e_1, e_0e_2, e_0e_3$ | 3 |
| 2 | $e_0e_4, e_0e_5, e_0e_6$ | 3 |
| 3 | $e_0e_7, e_0e_8, e_0e_9$ | 3 |
| 4 | $e_0e_{10}, e_0e_{11}, e_0e_{12}$ | 3 |
| 5 | $e_0e_{13}, e_0e_{14}, e_0e_{15}$ | 3 |
| 6 | $e_0e_{16}, e_0e_{17}, e_0e_{18}$ | 3 |

Table 5: Single-error combination patterns.

| Group | Single-error combination patterns | Numbers |
|---|---|---|
| 1 | $e_n(0 \le n \le 6)$ | 7 |
| 2 | $e_n(7 \le n \le 13)$ | 7 |
| 3 | $e_n(14 \le n \le 20)$ | 7 |
| 4 | $e_n(21 \le n \le 27)$ | 7 |
| 5 | $e_n(28 \le n \le 34)$ | 7 |
| 6 | $e_n(n = 35)$ | 1 |

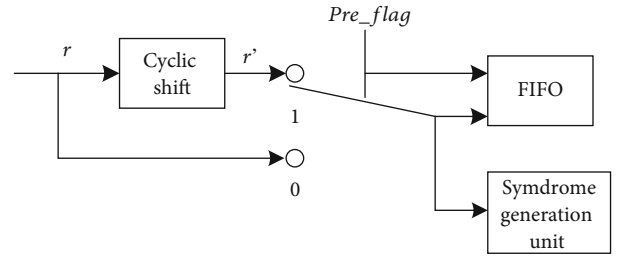


Figure 2: Preprocessing unit.

error pattern denoting as '*ep1*', '*ep2*', and '*ep3*' will be recorded to the register '*e_reg_out1*', '*e_reg_out2*', and '*e_reg_out3*'. When the error pattern traversal is completed, decoding will be performed according to the decoding flag bit, the recorded error pattern positions, and the original codeword stored in FIFO.

This framework reduces the register consumption of each stage of the pipeline from store 73 bits of entire codeword to store 18 bits of the three error pattern positions (each error pattern position is 6 bits), which greatly reduces the register consumption. The registers saved here can be used for other intermediate amounts of storage to increase the speed of the entire framework.

## 4. Hardware Architecture Design

According to Section 3, the operations before and after the cyclic shift are the same, so we can implement these two operations in same hardware framework by preprocessing and postprocessing of the received codeword, then we can realize pipeline multiplexing to improve resource utilization and reduce resource consumption. All the error patterns are summarized and divided into 6 parts, then the decoding

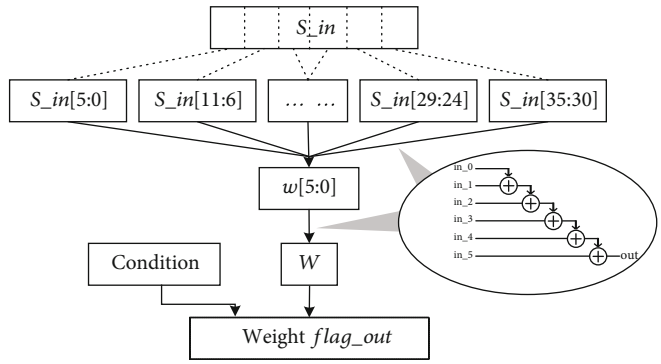


Figure 3: Weight calculation unit.

modules can be multiplexed. We traverse one part each clock, and all the error patterns are traversed by six clocks. The specific modules are as follows.

*4.1. Code Words Preprocessing Module.* Preprocess the input codeword $r$ by the cyclic shift operation, the original codeword $r$, and the cyclic shifted codeword $r'$ are given by

$$\begin{aligned} r &= \left(r_{m36}, r_{m35}, \cdots, r_{m0}, r_{p35}, r_{p34}, \cdots, r_{p1}, r_{p0}\right), \\ r' &= \left(r_{m0}, r_{p35}, r_{p34}, \cdots, r_{p0}, r_{m36}, r_{m35}, \cdots, r_{m1}\right). \end{aligned} \tag{5}$$

The specific structure is shown in Figure 2; the current data is determined according to the state of preprocessing flag denoted as 'pre_flag'(0/1); when $pre\_flag = 0$, the current data is original codeword $r$; when $pre\_flag = 1$, the current data is cyclic shifted codeword $r'$. Then, the data sequence $r, r'$ are input into initial syndrome generation unit one by one. Note that in order to make the figures clear, the clock signal is not shown in the figures below.

*4.2. Weight Calculation Unit.* The weight calculation unit can accumulate the sum of $s_i(\sum_{i=0}^{35} s_i)$ (as shown in Figure 3). In order to reduce the path delay and increase the operation frequency, we divide the *S_in* into 6 parts and complete the sum operation in 2 steps.

In addition, a module identification signal named "Condition" is introduced to make the weight calculation unit to adapt to different modules.

In initial syndrome generation module, "flag_out" will be enabled when $W(S_{\text{in}}) \le 6$.

In single-error traversal module, "flag_out" will be enabled when $W(S_{\text{in}}) \le 5$.

In two-error traversal module, "flag_out" will be enabled when $W(S_{\text{in}}) \le 4$.

In three-error traversal module, "flag_out" will be enabled when $W(S_{\text{in}}) \le 3$.

*4.3. Syndrome Generation Unit.* The syndrome generation unit has two kinds of units, one for $s_r$ and another for $s_{ri}$, $s_{rij}$, $s_{rijk}$.

The calculation of initial syndrome is given by

$$s_r = r_p \wedge (r_{m0} \cdot s_0) \wedge (r_{m1} \cdot s_1) \wedge \cdots \wedge (r_{m36} \cdot s_{36}). \tag{6}$$

Other syndrome calculations are given by

$$\begin{aligned} s_{r,i} &= s_r \wedge s_i (0 \le i \le 36), \\ s_{rij} &= s_r \wedge s_i \wedge s_j (0 \le i < j \le 36), \\ s_{rijk} &= s_r \wedge s_i \wedge s_j \wedge s_k (0 \le i < j < k \le 36), \end{aligned} \tag{7}$$

where $i, j, k$ are the error positions of $r_m$.

The specific structure is shown in Figure 4. The initial syndrome $s_r$ is calculated by formula (6) according to $r_m$, $r_p$ and syndromes $s_0 \sim s_{36}$. Then, the other syndromes $s_{ri}$, $s_{rij}$, and $s_{rijk}$ can be calculated by $s_r$, $s_i$, $s_j$, and $s_k$.

According to the algorithm, when the calculation of initial syndrome $s_r$ is completed, the weight $W(s_r)$ of the initial syndrome $s_r$ is calculated firstly. If $W(s_r) \le 6$, the decoding flag bit '*d_flag_init*' is set to 1, the register '*e_reg_out3*' of the current error pattern is assigned to "40", which can provide an indication for postprocessing unit to decoding.

*4.4. Error Traversal Units.* The error traversal units are divided into three types according to the number of errors in $r_m$, which are single-error traversal units, two-error traversal units, and three-error traversal units. In order to reduce the consumption of registers in this design, after each traversal unit calculates the weight of the new syndrome, only the current decoding flag bit 'd_flag_out', and the current error pattern 'ep1', 'ep2', 'ep3' are recorded instead of direct decoding. After all pattern traversal is completed, decoding is performed by decoding flag bit and the recorded error patterns.

*4.4.1. Single-Error Traversal Units.* The specific structure is shown in Figure 5. According to the input error pattern position 'ep1', calculate $s_{r,ep1} = s_r \wedge s_{ep1}$, and calculate the weight $W(s_{r,ep1})$ of $s_{r,ep1}$, then the flag bit 'd_flag_out' will be given according to $W(s_{r,ep1})$. Regardless of whether the current error pattern meets the decoding condition, the current error pattern position 'ep1' is recorded to 'ep1_out', which simplifies the logic and accurately records the error pattern.

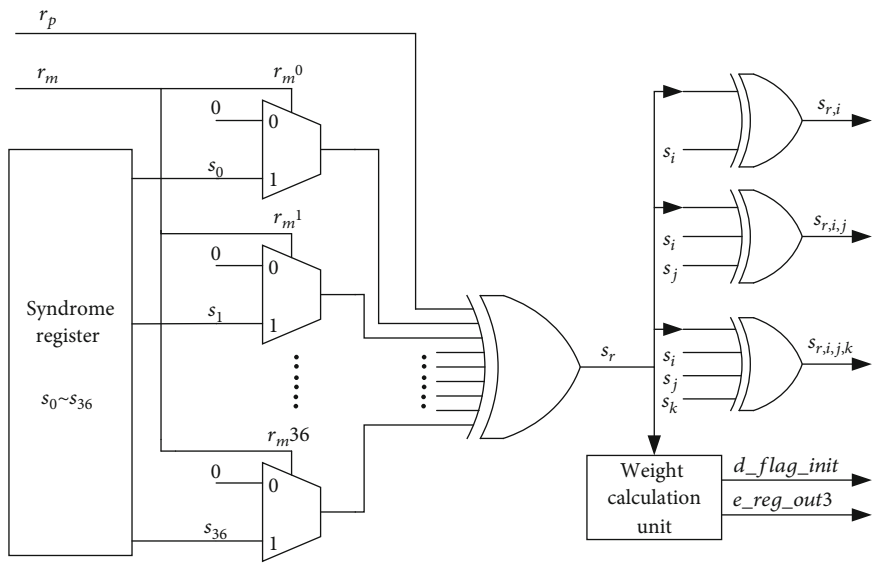


Figure 4: Syndrome generation unit.

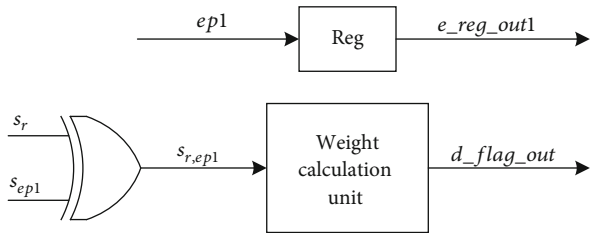


Figure 5: Single-error traversal unit.

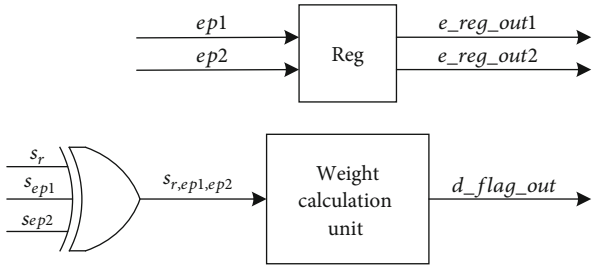


Figure 6: Two-error traversal unit.

*4.4.2. Two-Error Traversal Units.* The specific structure is shown in Figure 6. According to the input error pattern positions 'ep1', 'ep2', calculates $s_{r,ep1,ep2} = s_r \wedge s_{ep1} \wedge s_{ep2}$, and calculate the weight $W(s_{r,ep1,ep2})$ of $s_{r,ep1,ep2}$. Then, the flag bit 'd_flag_out' will be given according to $W(s_{r,ep1,ep2})$. At the same time, the current error pattern positions 'ep1', 'ep2' are recorded to 'e_reg_out1', 'e_reg_out2'.

*4.4.3. Three-Error Traversal Units.* The specific structure is shown in Figure 7. According to the input error pattern positions '*ep1*', '*ep2*', '*ep3*', calculate $s_{r,ep1,ep2,ep3} = s_r \wedge s_{ep1} \wedge s_{ep2} \wedge s_{ep3}$, and calculate the weight $W(s_{r,ep1,ep2,ep3})$ of $s_{r,ep1,ep2,ep3}$. Then, the flag bit '*d_flag_out*' will be given according to $W(s_{r,ep1,ep2,ep3})$. At the same time, the current error pattern

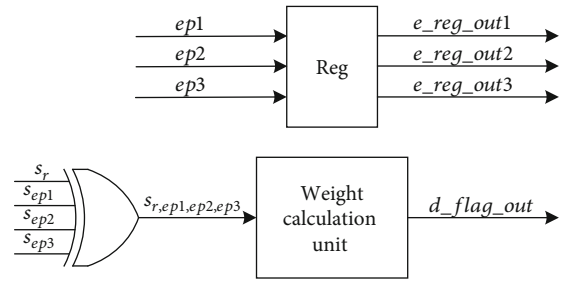


Figure 7: Three-error traversal unit.

Table 6: Traversal the error patterns.

| pctr | Three-error combination patterns | Two-error combination patterns | Single-error combination patterns |
|---|---|---|---|
| 1 | $e_0e_1e_n(2 \le n \le 35)$<br>$e_0e_{12}e_n(n = 24)$ | $e_0e_1, e_0e_2, e_0e_3$ | $e_n(0 \le n \le 6)$ |
| 2 | $e_0e_2e_n(4 \le n \le 34)$<br>$e_0e_{11}e_n(22 \le n \le 25)$ | $e_0e_4, e_0e_5, e_0e_6$ | $e_n(7 \le n \le 13)$ |
| 3 | $e_0e_3e_n(6 \le n \le 33)$<br>$e_0e_9e_n(18 \le n \le 27)$ | $e_0e_7, e_0e_8, e_0e_9$ | $e_n(14 \le n \le 20)$ |
| 4 | $e_0e_4e_n(8 \le n \le 32)$<br>$e_0e_9e_n(18 \le n \le 27)$ | $e_0e_{10}, e_0e_{11}, e_0e_{12}$ | $e_n(21 \le n \le 27)$ |
| 5 | $e_0e_5e_n(10 \le n \le 31)$<br>$e_0e_8e_n(16 \le n \le 28)$ | $e_0e_{13}, e_0e_{14}, e_0e_{15}$ | $e_n(28 \le n \le 34)$ |
| 6 | $e_0e_6e_n(12 \le n \le 30)$<br>$e_0e_7e_n(14 \le n \le 29)$ | $e_0e_{16}, e_0e_{17}, e_0e_{18}$ | $e_n(n = 35)$ |

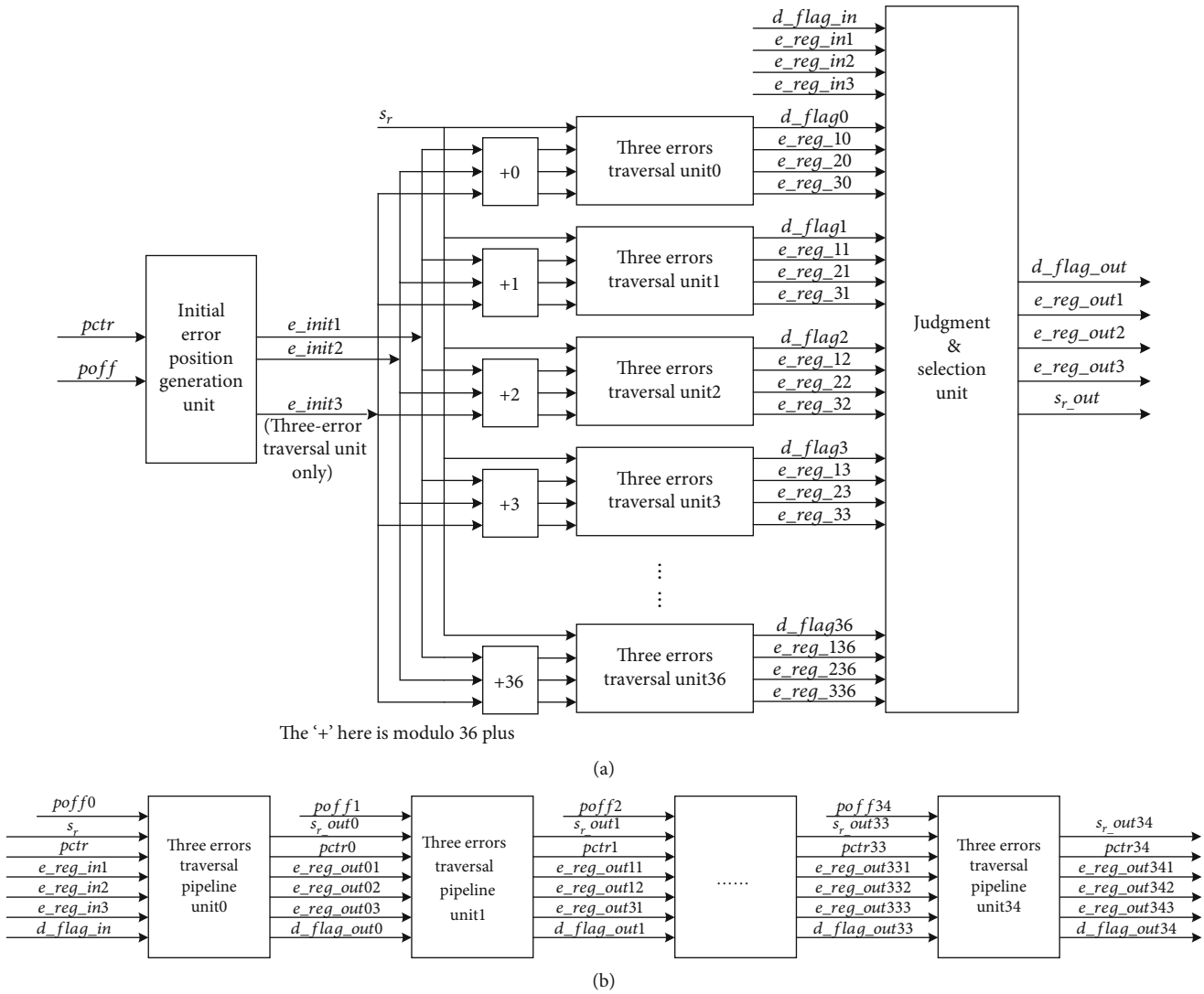


Figure 8: (a) Three-error traversal pipeline unit; (b) three-error traversal module.

positions '*ep1*', '*ep2*', '*ep3*' are recorded to '*e_reg_out1*', '*e_reg_out2*', '*e_reg_out3*'.

*4.5. Error Traversal Modules.* Each error traversal module can complete one kind of error traversal. To facilitate pipeline implementation, the error traversal units need to be integrated. An appropriate integration method can improve resource utilization and improve the running speed of the entire system.

*4.5.1. Three-Error Traversal Module.* The inductive error patterns are combined according to Table 3. After combining, the number of each combination pattern is 35. At this time, module multiplexing can be performed by controlling the value of the input parameter '*pctr*'. In each clock, we traverse 35 kinds of inductive error patterns; all three-error patterns will be finished in 6 clocks (as is shown in Table 6).

Change the input parameter '*pctr*' $(1 \le pctr \le 6)$ to control the switching of the combination patterns, and according to the offset parameter '*poff*' $(0 \le poff \le 34)$, we can determine the initialization pattern $e_{init1}e_{init2}e_{init3}$ to construct the basic pipeline unit. The calculation method of the initialization parameter is given by

$$init1 = 0,$$

$$init2 = \begin{cases} 13 - pctr, \text{when } (pctr * 2 + poff) > (36 - pctr), \\ pctr, \text{when } (pctr * 2 + poff) < (36 - pctr), \end{cases}$$

$$init3 = \begin{cases} pctr + poff - 11, \text{when } pctr * 2 + poff > 36 - pctr, \\ pctr * 2 + poff, \text{when } pctr * 2 + poff < 36 - pctr. \end{cases} \quad (8)$$

According to the input decoding flag bit '*d_flag_in*', if *d_flag_*in = 1, the error pattern of the previous stage traversal unit has met the decoding condition, set '*d_flag_out*' to 1, and register '*e_reg_in1*', '*e_reg_in2*', '*e_reg_in3*' directly to '*e_reg_out1*', '*e_reg_out2*', '*e_reg_out3*'; if *d_flag_*in = 0, the

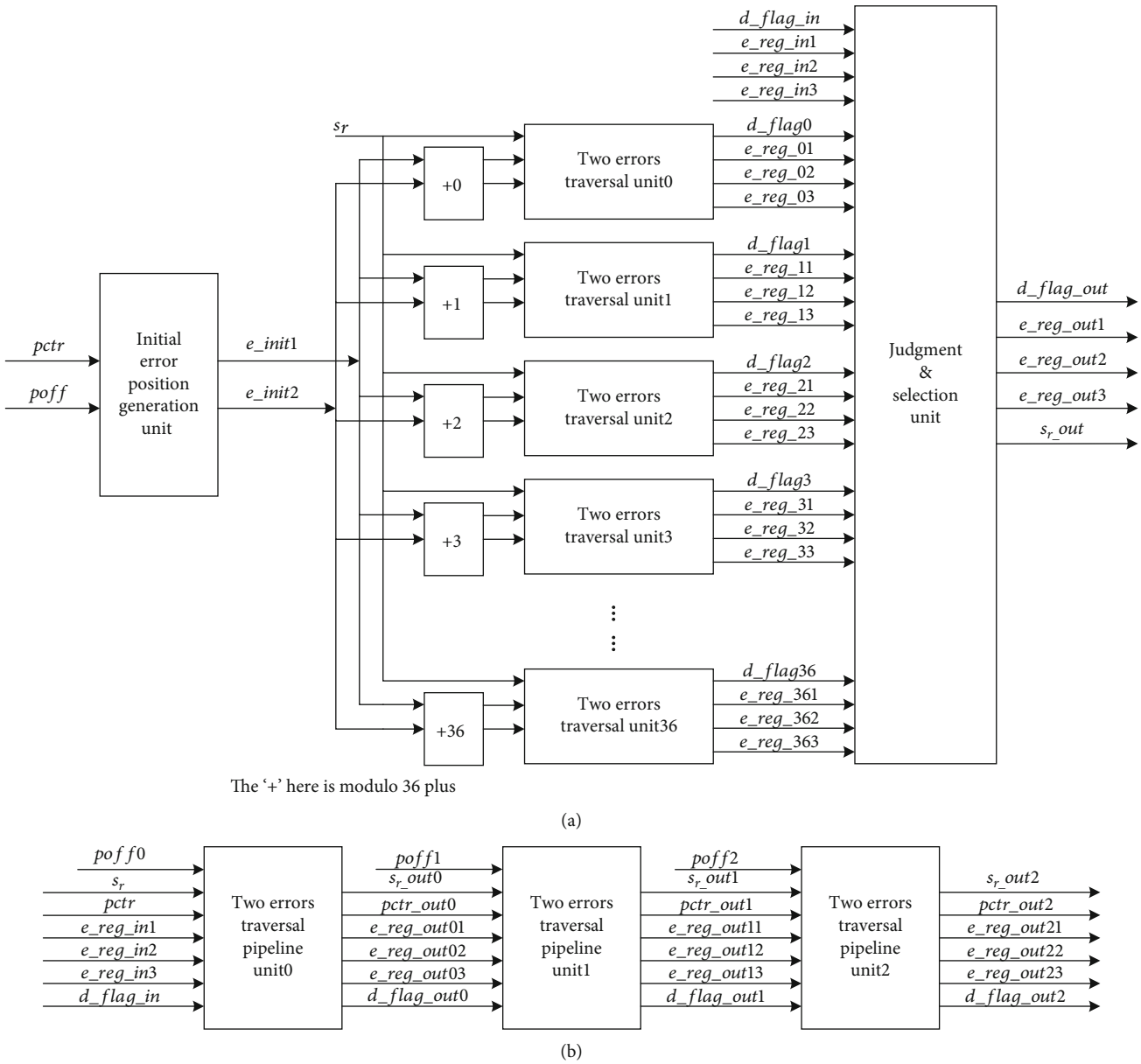


Figure 9: (a) Two-error traversal pipeline unit; (b) three-error traversal module.

error pattern of the previous stage traversal unit does not meet the decoding condition, and the error pattern traversal result of this level is considered.

The specific structure of the basic pipeline unit is shown in Figure 8(a). The initial error positions '*e_init1*', '*e_init2*', '*e_init3*' are determined according to the parameters '*pctr*' and '*poff*'. Then, modulo 36 plus $l(0 \le l \le 36)$ bits, respectively, with initial error positions to obtain new error patterns for traversal. Combined with the input initial syndrome $s_r$, 37 error traversal units are obtained to form a basic pipeline unit. After traversing these error patterns, the 37 results of traversal are judged and selected. If the error patterns in this unit satisfy the decoding condition, the '*d_flag_out*' is assigned to 1, and the current error positions are output to the error pattern record registers '*e_reg_out1*', '*e_reg_out2*', '*e_reg_out3*'. Then, the input initial syndrome $s_r$ is passed to the next level for traversal.

35 three-error traversal pipeline units are connected step by step to form a complete three-error traversal module. The specific structure is shown in Figure 8(b). The initial syndrome $s_r$, parameter '*pctr*', decoding flag '*d_flag_out*' and error pattern record register '*e_reg_out1*', '*e_reg_out2*', '*e_reg_out3*' are passed in stages to ensure the integrity of the error pattern traversal and the stability of the data.

*4.5.2. Two-Error Traversal Module.* In the three-error traversal pipeline unit, all three-error patterns have been distributed by 6 clocks, so we also distribute all two-error patterns into 6 clocks. In each clock, we traverse 3 inductive two-

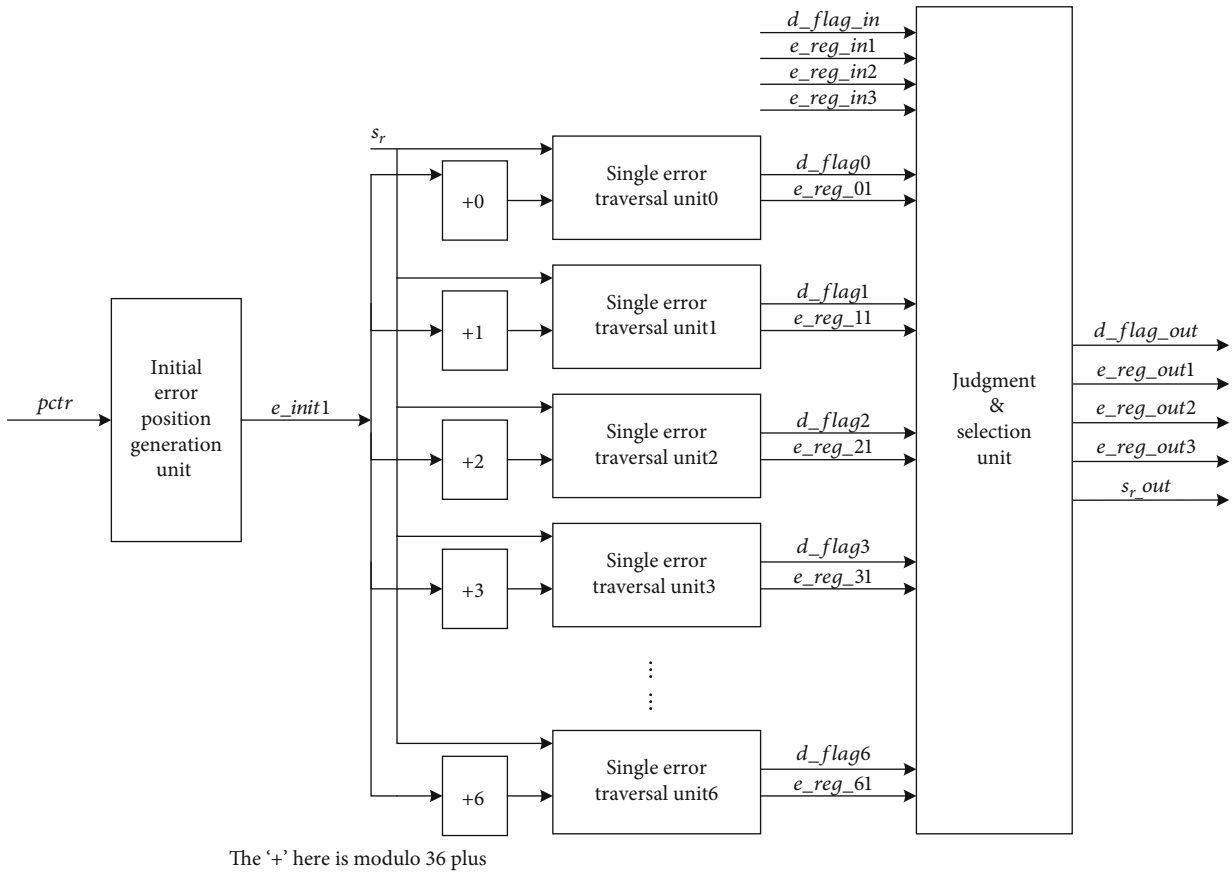


Figure 10: Single-error traversal pipeline unit.

error patterns, all two-error patterns will be finished in 6 clocks (as shown in Table 6).

Change the input parameter '*pctr*' $(1 \le pctr \le 6)$ to control the switching of the combination patterns, and according to the offset parameter '*poff*' $(0 \le poff \le 3)$, we can determine the initialization pattern $e_{init1}e_{init2}$ to construct the basic pipeline unit. The calculation method of the initialization pattern is given by

$$\begin{aligned} init1 &= 0, \\ init2 &= pctr * 3 + poff - 3. \end{aligned} \tag{9}$$

According to the input decoding flag bit '*d_flag_in*', if $d\_flag\_\text{in} = 1$, the error pattern of the previous stage traversal unit has met the decoding condition, set '*d_flag_out*' to 1, and register '*e_reg_in1*', '*e_reg_in2*', '*e_reg_in3*' directly to '*e_reg_out1*', '*e_reg_out2*', '*e_reg_out3*'; if $d\_flag\_\text{in} = 0$, the error pattern of the previous stage traversal unit does not meet the decoding condition, and the error pattern traversal result of this level is considered.

The specific structure is as shown in Figure 9(a), and the initial error positions 'e_init1', 'e_init2' are determined according to the parameters 'pctr' and '*poff*'. Then, modulo 36 plus $l(0 \le l \le 36)$ bits, respectively, with initial error positions to obtain new error patterns for traversal. Combined with the input initial syndrome $s_r$, 37 error traversal units are obtained to form a basic pipeline unit. After traversing these error patterns, the 37 results of traversal are judged and selected. If the error patterns in this module satisfy the decoding condition, the 'd_flag_out' is assigned to 1, the current error positions are output to the error pattern record registers 'e_reg_out1', 'e_reg_out2', and the value of 'e_reg_out3' is set to 42, providing an indication for the postprocessing unit. Then, the input initial syndrome is passed to the next level for traversal.

Three two-error traversal pipeline units are connected step by step to form a complete two-error traversal module. The specific structure is shown in Figure 9(b). The initial syndrome, parameters 'pctr', 'poff', decoding flag 'd_flag_out', and error pattern record registers 'e_reg_out1', 'e_reg_

Table 7: Decoding operations.

| e_reg_out3 | m_decode | p_decode |
|---|---|---|
| 40 | $m\_delay$ | $p\_delay \wedge s\_delay$ |
| 41 | $m\_delay \wedge e_{e\_reg1}$ | $p\_delay \wedge s\_delay \wedge s_{e\_reg1}$ |
| 42 | $m\_delay \wedge e_{e\_reg1} \wedge e_{e\_reg2}$ | $p\_delay \wedge s\_delay \wedge s_{e\_reg1} \wedge s_{e\_reg2}$ |
| 0~36 | $m\_delay \wedge e_{e\_reg1} \wedge e_{e\_reg2} \wedge e_{e\_reg3}$ | $p\_delay \wedge s\_delay \wedge s_{e\_reg1} \wedge s_{e\_reg2} \wedge s_{e\_reg3}$ |

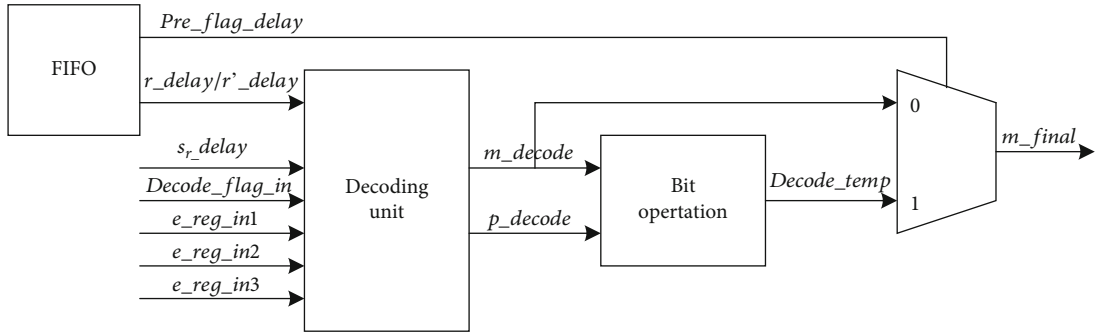


Figure 11: Decoding and postprocessing unit.

out2', 'e_reg_out3' are passed in stages to ensure the integrity of the error pattern traversal and the stability of the data.

*4.5.3. Single-Error Traversal Module.* The same as three-error and two-error traversal pipeline unit, all the single-error traversal units are distributed to 6 clocks (as shown in Table 6).

Change the input parameter 'pctr' $(1 \le pctr \le 6)$ to control the switching of the combination patterns, we can determine the initialization pattern $e_{init1}$ to construct the basic pipeline unit. The calculation method of the initialization pattern is given by

$$init1 = (para\_ctrl - 1) * 7. \tag{10}$$

According to the input decoding flag bit 'd_flag_in', if $d\_flag\_in = 1$, the error pattern of the previous stage traversal unit has met the decoding condition, set 'd_flag_out' to 1, and register 'e_reg_in1', 'e_reg_in2', 'e_reg_in3' directly to 'e_reg_out1', 'e_reg_out2', 'e_reg_out3'; if $d\_flag\_in = 0$, the error pattern of the previous stage traversal unit does not meet the decoding condition, and the error pattern traversal result of this level is considered.

The specific structure is as shown in Figure 10, and the initial error position 'e_init1' is determined according to the parameter 'pctr'. Then, modulo 36 plus $l(0 \le l \le 36)$ bits, respectively, with initial error position to obtain new error patterns for traversal. Combined with the input initial syndrome $s_r$, 7 single-error traversal units are obtained to form a complete single-error traversal module.

After traversing the error patterns, the 7 results of traversal are judged and selected. If the error patterns satisfy the decoding condition in this module, the 'd_flag_out' is set to 1, the current error positions are output to the error pattern record registers 'err_reg_out1'. And 'err_reg_out2' is set to 0, 'err_reg_out3' is set to 41, which provide indications for postprocessing unit. Then, the input initial syndrome $s_r$ is passed for final decoding.

*4.6. Decoding and Postprocessing Unit.* Read the data stored in the FIFO, denoted as 'r_delay' and 'pre_flag_delay', and 'r_delay' contains a message part 'm_delay' and a parity part 'p_delay'. According to the decoding status flag bit 'd_flag1', error pattern record registers 'e_reg_out11', 'e_reg_out12', 'e_reg_out13', preprocessing status flag bit 'pre_flag_delay' and the pipeline passed syndrome 's_delay', the decoding can be completed. Denote the decoded codeword message part as 'm_decode' and denote decoded codeword parity part 'p_decode'. The operations of decoding are shown in Table 7.

After the decoding is completed, postprocessing is required to restore the original codeword order. The decoded codeword is cyclically shifted and restored according to the preprocessing status register '*pre_flag_delay*', thereby we can obtain the final decoded codeword '*m_final*' as

$$m\_final = \begin{cases} m\_decode, \text{when } pre\_flag\_delay = 0, \\ \{p\_decode, m\_decode[36]\}, \text{when } pre\_flag\_delay = 1. \end{cases} \tag{11}$$

At this time, the decoding is completed, 'm_final' is the final decoded codeword, and the specific structure is as shown in Figure 11.

*4.7. Pipeline Architecture.* The preprocessing module, the initial syndrome generation module, the three-error traversal module, the two-error traversal module, the single-error traversal module, and the decoding and postprocessing module are connected step by step to form a 42-stage pipeline. The specific architecture is shown in Figure 12.

*4.8. Hardware Implementation Results.* The experiment and verification are divided into two parts, software verification and hardware verification. Software verification tests the performance of QR codes while hardware verification tests the acceleration effects of QR codes. Software verification part is carried out on the Personal Computer (PC), which is also mentioned in Section 2. The verification software runs in VS2019, Intel core I5-6500, windows 10 environment, and the decoding time is shown in Table 8. In order to show the difference between various decoding algorithms, we have recorded the decoding error correction time of various decoding algorithms under different number of errors. There is no doubt that the more errors an encoded message has, the longer decoding time cost, and the unstable decoding delay will have an impact on the real-time performance of the IOT device. In addition, the software experiment content also included the average decoding time of various decoding algorithms by calculating average based on the probability distribution of the number of errors, the results can be used as a

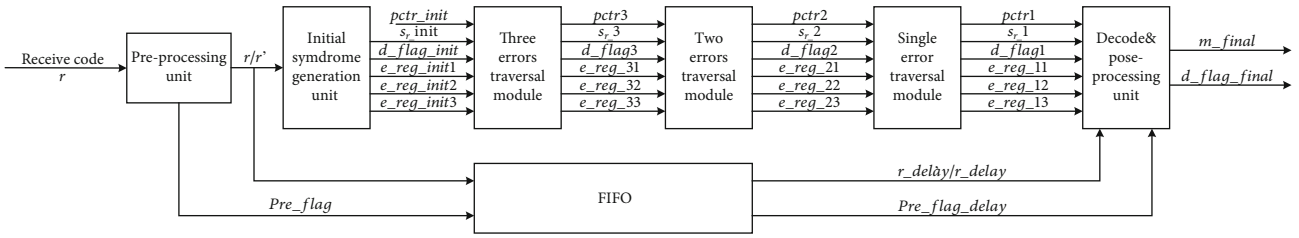


Figure 12: Pipeline architecture.

Table 8: Decoding time under various conditions.

| Environment | Works | Error numbers (us) 1 | 2 | 3 | 4 | 5 | 6 | Average 1 | Average 2 |
|---|---|---|---|---|---|---|---|---|---|
| PC | *DS in this paper* | 1.12 | 2.16 | 4.24 | 19.76 | 48.36 | 203.66 | 190.00 | 3.863 |
| | IFBM in [17] | 205.5 | 142.7 | 219.3 | 2955.9 | 3104.6 | 11824 | 11065.57 | 378.42 |
| | ADA in [18] | 1.8771 | 17.412 | 71.391 | 269.36 | 536.31 | 5954.5 | 5482.66 | 49.97 |
| | SSDDA in [19] | 1.7002 | 2.5378 | 22.863 | 286.47 | 482.94 | 701.54 | 681.27 | 26.4 |
| | Algorithm in [20] | 1134 | 1848 | 2611 | 6822 | 11417 | 30915 | 29193.32 | 2089 |
| | DS in [14] | 0.854 | 5.69 | 38.3 | 95.8 | 198 | 245 | 240.2691 | 16.02 |
| Arria10 | *Hardware implemented in this paper* | 0.0461 | 0.0461 | 0.0461 | 0.0461 | 0.0461 | 0.0461 | 0.0461 | 0.0461 |

Note that the average decoding time 1 is calculated by supposing all error patterns have the same probability of occurrence. And the average decoding time 2 is calculated by supposing there must be 1~5 errors in the received code and is under the environment of 7 dB SNR, where the occurring probability is 41.27% of one error, 36.49% of two errors, 15.92% of three errors, 5.03% of four errors, 1.1% of five errors, and 0.2% of six errors.

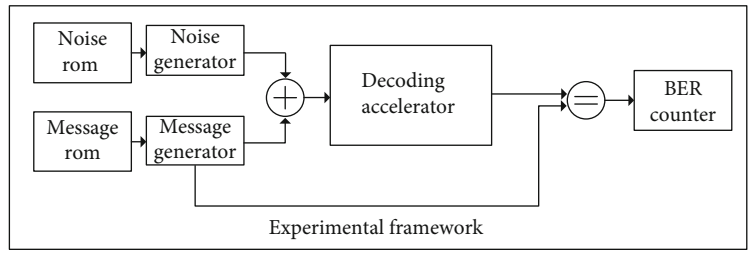


Figure 13: Hardware experimental framework.

reference for the real-time performance of decoding algorithms.

The hardware verification framework is shown in Figure 13. Random messages and noise are generated by ROM lookup table. The sum of encoded message and noise will input into the decoding accelerator. The BER counter records the error rate by comparing the decoding result with the original message.

The decoding accelerator is deployed in Intel Arria10 10AX115U4F45I1SG FPGA; the results is given in the Table 9.

In this decoding framework, all error patterns are evenly distributed into the pipeline, so the decoding time is fixed. A complete decoding requires 125 clocks, the traverse modules cost 12 clocks, 6 clocks to traverse all patterns, and 6 clocks repeated after shifting. Therefore, after 12 clocks of pipeline processing, a valid decoding result can be obtained. At the operation frequency of 260.42 M Hz, every 12 clock outputs can be equivalent to a decoding speed of 21.7 M Hz, the decoding time is 46.07 ns, and the decoding time required of each condition is shown in Table 8. Compared with PC, hardware decoding acceleration can obtain 4121 times of decoding speed increase and can maintain a constant decoding time.

Table 9: Results of accelerator.

| | Available | Result |
|---|---|---|
| ALMs | 43.72 k | 12.7 k |
| Block RAM | 55.6 Mbit | 0.15 Mbit |
| Max clock freq. | — | 260.42 MHz |
| Latency | — | 46.07 ns |

## 5. Conclusion

This article introduces the feasibility and advantages of QR code in power carrier technology based on the concept of

UPIoT, and according to its shortcomings (decoding complexity and defects with long decoding time) proposed a solution. By improving the DS decoding algorithm, the error pattern is split according to the characteristics of the inductive combination of error patterns, the proposed Error Pattern Induction method has been simulated to prove that the performance is not lost and is better than RS code. This article also analyses the hardware feasibility of separating error pattern traversal and decoding operations and implements an FPGA-based hardware decoding architecture on this basis, on the Intel Arria10 10AX115U4F45I1SG FPGA platform, an equivalent decoding frequency of up to 21.7 MHz is realized, which is 4121 times the decoding speed of software. To our best knowledge, the hardware decoding architecture proposed in this paper is the first hardware implementation of (73, 37, 13) QR code decoding architecture.

While providing a new implementation scheme for PLC error correction coding based on UPIoT, this architecture also shows that QR codes with longer codewords can also be quickly decoded by our hardware frame.

## Data Availability

Our data is not public because of confidentiality, readers can contact hjynet@hdu.edu.cn for available data.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

This work was supported by the Key Research and Development Program of Zhejiang Province (2020C01110).

## References

[1] C. S. Lai, G. Locatelli, A. Pimm, X. Wu, and L. LeiLai, "A review on long-term electrical power system modeling with energy storage," *Journal of Cleaner Production*, vol. 280, no. 1, p. 124298, 2021.

[2] J. Zhong and T. Zheng, "Ubiquitous power Internet of Things architecture construction plan," in *2019 IEEE 8th International Conference on Advanced Power System Automation and Protection (APAP)*, pp. 1549–1552, Xi'an, China, 2019.

[3] C. S. Lai, L. L. Lai, and Q. H. Lai, *Smart Grids and Big Data Analytics for Smart Cities*, Springer International Publishing, New York, NY, USA, 2020.

[4] Q. Wang and Y. G. Wang, "Research on power Internet of Things architecture for smart grid demand," in *2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2)*, pp. 1–9, Beijing, China, 2018.

[5] C. Lin, K. Xiang, Y. Du, Y. Li, and H. Lin, "Research on the energy Internet under the background of ubiquitous power Internet of Things," in *2019 IEEE 3rd Conference on Energy Internet and Energy System Integration (EI2)*, pp. 403–407, Changsha, China, 2019.

[6] Z. Shi, H. Wang, C. Wang, X. Wang, J. Wang, and S. Li, "Research on UPIoT application of low-voltage station area based on dual-mode communication," in *2020 12th IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, pp. 1–6, Nanjing, China, 2020.

[7] Y. M. Chung, "Overview and characteristics of IoT PLC," in *2020 International Conference on Electronics, Information, and Communication (ICEIC)*, pp. 1–3, Barcelona, Spain, 2020.

[8] Y. Fang, G. Han, G. Cai, F. C. M. Lau, P. Chen, and Y. L. Guan, "Design guidelines of low-density parity-check codes for magnetic recording systems," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1574–1606, 2018.

[9] Y. Fang, P. Chen, G. Cai, F. C. M. Lau, S. C. Liew, and G. Han, "Outage-limit-approaching channel coding for future wireless communications: root-protograph low-density parity-check codes," *IEEE Vehicular Technology Magazine*, vol. 14, no. 2, pp. 85–93, 2019.

[10] G. Cai, Y. Fang, P. Chen, G. Han, G. Cai, and Y. Song, "Design of an MISO-SWIPT-aided code-index modulated Multi-CarrierM-DCSK system for e-health IoT," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 2, pp. 311–324, 2021.

[11] P. Chen, Z. Xie, Y. Fang, Z. Chen, S. Mumtaz, and J. J. P. C. Rodrigues, "Physical-layer network coding: an efficient technique for wireless communications," *IEEE Network*, vol. 34, no. 2, pp. 270–276, 2020.

[12] H. Zhang and A. Zhang, "Error correction for PLC systems based on OFDM," in *2017 36th Chinese Control Conference (CCC)*, pp. 5439–5443, Dalian, China, 2017.

[13] E. Prange, *Some cyclic error-correcting codes with simple decoding algorithms*, Air Force Cambridge Research Center-TN-58-156, Cambridge, MA, 1958.

[14] Y. Li, Y. Duan, H.-C. Chang, H. Liu, and T.-K. Truong, "Using the difference of syndromes to decode quadratic residue codes," *IEEE Transactions on Information Theory*, vol. 64, no. 7, pp. 5179–5190, 2018.

[15] J. Huang, T. Zhou, H.-C. Chang, and D. Xie, "An optimized cyclic weight algorithm of (47, 24, 11) QR code and hardware implementation," *IEEE Access*, vol. 6, pp. 36995–37002, 2018.

[16] D. Yansen, L. Wang, and L. Yong, "New algebraic decoding of the (73, 37, 13) quadratic residue code," *Journal of Chongqing University of Posts and Telecommunications*, vol. 25, no. 5, pp. 622–627, 2013.

[17] H.-P. Lee, H.-C. Chang, and T.-K. Truong, "Algebraic decoding of the (73, 37, 13) quadratic residue code," *IET Communications*, vol. 6, no. 10, pp. 1326–1333, 2012.

[18] H.-P. Lee and H.-C. Chang, "An efficient decoding algorithm for the (73, 37, 13) quadratic residue code," in *Advances in Information Technology and Education*, pp. 224–231, Springer, 2011.

[19] Y. Li, H. Liu, Q. Chen, and T.-K. Truong, "On decoding of the (73, 37, 13) quadratic residue code," *IEEE Transactions on Communications*, vol. 62, no. 8, pp. 2615–2625, 2014.

[20] L. G. de Oliveira, G. R. Colen, M. V. Ribeiro, and A. J. H. Vinck, "Narrow-band interference error correction in coded OFDM-based PLC systems," in *2016 International Symposium on Power Line Communications and its Applications (ISPLC)*, pp. 13–18, Bottrop, Germany, 2016.