Ensuring Noise Immunity of The Modbus Industrial Communication Protocol Based on Linear LDPC and BCH Codes

Stepan A. Chapliyov
Saint Petersburg Electrotechnical University "LETI", Saint Petersburg, 197022, Russian Federation
Saint Petersburg, Russian Federation
s.chapliyov@mail.ru

Alla B. Levina
Saint Petersburg Electrotechnical University "LETI", Saint Petersburg, 197022, Russian Federation
Saint Petersburg, Russian Federation
Alla_levina@mail.ru

Abstract– This paper presents a modernization of the Modbus industrial communication protocol using Low Density Parity Check and Bose–Chaudhuri–Hocquenghem linear block error-correcting codes. While traditional Modbus relies on basic checksum algorithms, the proposed approach replaces these with advanced linear codes to enhance noise immunity. This paper describes these algorithms and provide a comparative analysis of their performance. Results demonstrate that this modernized framework allows for reliable Modbus communication.

Keywords-component; Correcting Linear Block Codes, Bose–Chaudhuri–Hocquenghem codes, Low Density Parity Check codes, Information Integrity, Modbus.

I. INTRODUCTION

Information integrity is an important component of information security (IS). The importance of robust integrity measures increases significantly when its compromise is more probable than other threats, such as breaches of confidentiality or availability. A prime example is found in industrial communications: the use of isolated local networks and strict physical access controls minimizes the risk of external intrusion, thereby prioritizing the accuracy and correctness of data transmission.

To this day, Modbus – developed in 1979 – remains one of the most prevalent standards in the industrial sector. In its remote terminal unit (RTU) variant, the protocol natively uses the cyclic redundancy check (CRC) algorithm for integrity checks, which offers only a baseline level of protection.

This paper examines linear block codes, specifically low density parity check (LDPC) and Bose–Chaudhuri–Hocquenghem (BCH), as mechanisms for ensuring information integrity during transmission. Unlike the standard tools used in Modbus RTU, these algorithms not only detect errors but also correct them at the receiver's end, eliminating the need for data retransmission.

Implementing these algorithms removes the necessity of retransmitting messages when faults occur. Under the high bit error rate (BER) conditions, the number of retries could reach the dozens, significantly reducing transmission speeds and overall system responsiveness.

Manuscript received February 19, 2026; revised March 12, 2026; accepted June 4, 2026. Published June 15, 2026.
Issue category: Regular
Paper category: Regular
DOI: 10.64552/wipiec.v12i1.130

The remainder of this paper is organized as follows. Section II discusses the relevance of the research. Section III provides an overview of the Modbus protocol. Section IV introduces the general principles of linear block codes, while section V compares the performance of CRC, BCH and LDPC algorithms. The proposed algorithms for integrating these codes into Modbus are detailed in section VI, followed by a comparative analysis using Matlab simulations in Section VII. Section VIII describes the hardware implementation of the BCH codec on Verilog. Sections IX and X conclude the paper.

II. RELEVANCE OF THE RESEARCH

Information integrity is a fundamental component of information security, ensuring that data remains accurate and unchanged during processing and transmission. Since communication channels are inherently susceptible to noise and external interference, the implementation of error-correction methods is essential for maintaining system reliability. Linear codes, particularly BCH and LDPC codes, serve as the primary mathematical framework for effective error detection and correction.

BCH codes are widely utilized in data storage systems, such as NAND Flash memory, and in the ISO/IEC 18004:2015 [1] standard for quick response (QR) codes, which are prevalent in commercial and banking sectors. These codes are also integral to the DVB-S2 [2] digital satellite broadcasting standard due to their high noise immunity. A key advantage of BCH codes is the ability to specify code parameters, such as the minimum code distance, allowing for flexible optimization based on specific system requirements.

However, the increasing demand for high-speed data transmission and computational efficiency has shifted focus toward LDPC codes. These codes provide superior error-correction performance with relatively low computational overhead. Currently, LDPC codes are implemented in wireless and wired networking standards, including IEEE 802.11 [3] (Wi-Fi) and IEEE 802.3 (Ethernet).

Also LDPC codes are critical for high-interference environments, as evidenced by their use in the DVB-S2 [2] standard and the CCSDS 131.1-B [4] space communication

protocols. Their application in 5G mobile networks further underscores their importance in providing high throughput and stability in complex urban environments. Consequently, the development and optimization of LDPC and BCH codes remain vital for the advancement of modern information infrastructure.

Regarding the Modbus protocol, it is essential to emphasize that its reliability when operating over degraded communication channels remains a critical concern. Specifically, as demonstrated in [5], the worst-case BER in wireless channels governed by the IEEE 802.15.4 standard is approximately 0.01. This indicates a high error frequency, where one bit in every 100 transmitted is likely to be erroneous. Even under optimal conditions, the best recorded BER is approximately 0.0004, which still necessitates robust error-handling mechanisms.

The impact of these error rates is represented in Table I, which illustrates the relationship between the average number of retransmissions required to ensure the error-free delivery of a single packet and single error probability, based on Matlab simulation results.

TABLE I. Dependence of transmissions on the probability of errors

Single error probability	0.0002	0.004	0.007	0.009	0.0103	0.0104	0.012
Average number of retransmissions	1	2	3	6	7	8	11

Furthermore, Fig. 1 presents the time costs required to transmit Modbus packets as a function of the single-bit error probability. These data, derived through Matlab modeling, highlight the significant performance degradation of the protocol in high-interference environments, reinforcing the need for sophisticated error-correction. For this simulation, a sample size of 7000 packets was used for each error probability. This value was selected to ensure a sufficient number of errors while maintaining computational efficiency. Preliminary tests indicated that increasing the sample size beyond this value did not result in significant changes to the observed trends, but only increased the simulation time.

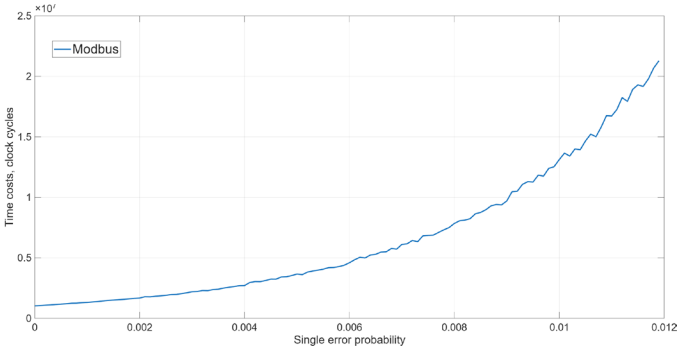

Fig. 1. Graph of time costs versus error probability

III. The Modbus Protocol

Modbus [6] is an application-layer protocol within the OSI model, based on a client-server (master-slave) architecture. This standard allows for the integration of up to 247 devices on a single bus; each node's address is encoded in a single byte, excluding reserved values.

The protocol strictly defines the message structure, data access methods, and response rules. Its primary feature is physical medium independence: Modbus can operate over serial interfaces as well as TCP/IP networks. Within the master-slave architecture, communication is organized such that the master device initiates requests and manages traffic, while the slave nodes only respond to them, lacking the capability to initiate data transmission independently.

A typical frame structure in the protocol includes the following elements:

1. Device Address;
2. Function Code;
3. Data;
4. Checksum.

The protocol is represented by three main versions:

- Modbus RTU;
- Modbus ASCII;
- Modbus TCP.

The key differences between these versions lie in their data representation and integrity assurance methods. In Modbus RTU, data is transmitted in binary format, with frames separated by time intervals, and integrity is verified using the CRC16 algorithm. The Modbus ASCII version uses hexadecimal format (doubling the data volume), separates messages with specific control characters, and uses the longitudinal redundancy check (LRC) algorithm for error detection. In Modbus TCP, only the block containing the function code and data is transmitted, encapsulated within a TCP packet; here, integrity mechanisms are handled directly by the TCP transport protocol.

IV. The Correcting Linear Block Codes

Linear codes are mathematical structures that utilize data redundancy to correct a specific number of errors (or, in certain cases, to detect errors if their quantity exceeds the code's correction capabilities).

Formally, a linear *(n, k)*-code is defined as a linear subspace C of dimension k within the vector space F_q^n, where F_q^n, is a finite field (Galois field) consisting of q elements [7].

Since linear codes are block codes, operations are performed on data blocks.

The parity check matrix H of the code [7] is defined as a matrix of size $(n-k) \times n$ having the form:

$$H = [A|I_{n-k}], \tag{1}$$

where A is a matrix of size $(n-k) \times k$, and I_{n-k} is an identity matrix of size $(n-k) \times (n-k)$.

The parity check matrix [5 (из Д)] must satisfy the equation:

$$Hx^T = H\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = 0, \quad (2)$$

where x is the codeword vector.

The generator matrix G [7] of the code is defined as a matrix of size $k \times n$ of the form:

$$G = [I_k \mid -A^T], \quad (3)$$

where I_k, is an identity matrix of size $k \times k$.

The generator matrix is used for message encoding. Let u be the source message and x be the encoded message, then:

$$x = uG. \quad (4)$$

The generator and parity check matrices are related by the following equations [7]:

$$GH^T = HG^T = 0. \quad (5)$$

BCH codes [7] are based on a proposition known as the BCH theorem or BCH bound: let ϑ be a cyclic code with a generator polynomial $g(x)$ such that for certain integers $b \geq 0$ and $\delta \geq 1$ the following equality holds:

$$g(a^b) = g(a^{b+1}) = \cdots = g\left(a^{a+\delta-2}\right) = 0, \quad (6)$$

where a is a primitive element of the field over which the code is constructed. Then the minimum (Hamming) distance of the code is at least δ. The value δ is called the designed distance of the BCH code.

LDPC codes [8] are linear block codes characterized by a sparse parity check matrix, meaning the number of zero elements significantly exceeds the number of non-zero elements. Codes in which the number of non-zero elements in every row and every column of the parity check matrix is constant are called regular.

Another requirement for LDPC codes is the absence of short cycles (e.g., length 4) in their graph structure. Short cycles degrade convergence during decoding, whereas increasing the length of the shortest cycle (girth) improves the code's error-correcting capability.

V. COMPARISON OF THE CODES

The modernization process of the Modbus protocol requires a performance analysis of standard integrity control mechanisms compared to the potential of error-correcting linear block codes.

The Modbus RTU specification utilizes the CRC16 algorithm for data verification, which is based on calculating the remainder of a division by a specified polynomial. According to [9], a software implementation of this algorithm requires 64 central processing unit (CPU) cycles per byte of information. Given that integrity verification requires repeating these calculations on the receiving side, the total overhead for generating and verifying the checksum reaches 128 cycles per byte. When using a hardware implementation, as stated in [9], these overheads are significantly lower: 8 cycles per byte for generation and another 8 for verification (totaling 16 cycles).

The performance analysis of BCH codecs involves separate calculations for encoding and decoding operations. A hardware implementation of a (*n,k*)-BCH encoder, built using linear feedback shift registers (LFSR), requires *n* cycles per block.

According to [10], hardware decoding for codes such as (15,11,1), (15,7,2), and (15,5,3) takes 33 cycles. Thus, a full codec cycle for a 15-bit block is 48 cycles, which translates to approximately 26 cycles per byte. Meanwhile, the author of [11] indicates that in the worst-case scenario, the time costs for a hardware (n,k,t)-decoder are $2n+2t$ cycles. For the aforementioned (15,11,1), (15,7,2), and (15,5,3) codes, the total complexity would be 47, 49, and 51 cycles respectively (roughly 25–27 cycles per byte).

Software versions of BCH codecs are not considered in this study due to their low performance. For example, [12] records that a software (255,131,18) codec running at a CPU frequency of 3.1 GHz performs operations in 73 μs. This is equivalent to 226,300 CPU cycles, or 888 cycles per single byte of data.

Performance parameters for LDPC codecs were also examined. The paper [13] notes that hardware LDPC (1536,1024) encoding requires 5632 cycles (44 cycles per byte). The decoding procedure for an equivalent volume of data takes the same amount of time, corresponding to 30 cycles per byte (factoring in redundancy). Ultimately, the total specific load on the system is 74 cycles per byte.

Regarding software implementation, for an LDPC (16384,4096) code on an Intel Xeon Gold 6148 processor, the decoding time is 34 μs, which is approximately 57 cycles per byte [14]. Despite the lack of direct encoding data in that specific study, based on [15], one can assume a linear complexity for the process that does not exceed the decoding complexity. Thus, the aggregate cost of such a codec can be estimated at within 114 cycles per byte.

The summarized results of these characteristics are presented in Table II.

TABLE II. Comparative characteristics of algorithms

Algorithm	Software impl., clock cycles per byte	Hardware impl., clock cycles per byte
CRC16	128	16
BCH	888	26
LDPC	114	74

VI. Algorithms of implementation correcting linear block codes as part of the Modbus

The prospects for integrating Error-Correcting Linear Block Codes (LBCs) into the Modbus protocol to ensure information integrity during transmission include the following options:

1. Checksum Replacement Algorithm

This method involves replacing the standard checksum bits (CRC) with parity bits calculated using LBC. While this increases the total packet length, it enables error correction in addition to detection. However, this approach requires modifying all network devices, as it is not backward compatible.

- Pros: Error correction capability, eliminates the need for retransmission upon error.
- Cons: Lack of backward compatibility, increased computational and transmission overhead.

2. LBC Extension Algorithm

In this scenario, LBC parity bits are appended after the original checksum bits. This maintains backward compatibility: if a legacy device receives the packet, it processes the required bytes and ignores the trailing LBC bits. (Success depends on how specific Modbus implementations handle trailing data).

- Pros: Error correction; no retransmission needed, backward compatible.
- Cons: Highest transmission overhead due to dual checksums.

3. Adaptive Coding Algorithm

This approach uses standard integrity checks for short messages and LBC encoding for long messages. This avoids LBC overhead when retransmitting a short packet is more efficient than the computational cost of decoding, while preventing the retransmission of large data blocks.

- Pros: Error correction, prevents full retransmission of long messages, adaptive overhead management.
- Cons: Complex implementation, lack of backward compatibility, increased overhead for long messages.

4. Channel Codec Algorithm

This approach treats the entire Modbus packet as "payload" to be delivered reliably over the communication channel. Instead of modifying Modbus, it establishes a robust "transport" or "link" layer underneath it. Since a Modbus packet can reach 255 bytes, using a fixed code length for the whole packet would create massive overhead for short messages. A more efficient implementation uses smaller blocks (e.g., k=11 bits, n=15 bits). The packet is fragmented into k-byte segments and reassembled at the receiver. Padding is only required for the final segment, significantly reducing redundancy.

- Pros: Error correction, no retransmission for long messages, full backward compatibility (no changes to Modbus protocol logic).
- Cons: Increased computational overhead, increased infrastructure complexity.

It should be noted that "backward compatibility" in these examples is somewhat relative. Since Modbus is an application-layer protocol implemented in software, a "new" version could theoretically support multiple checksum methods. For instance, devices could negotiate or detect supported algorithms during an initial handshake or initialization phase.

VII. Comparison of algorithms

To simulate the implementation of the Modbus protocol using linear codes, the Matlab software suite was utilized.

According to [5], the worst-case Bit Error Rate (BER) in a wireless communication channel based on the IEEE 802.15.4 standard is approximately 0.01, implying that one bit out of every 100 transmitted will be erroneous. The best recorded BER is roughly 0.00036, or one erroneous bit per 2778 bits.

Four distinct models were implemented for this study:

Model 1: A channel transmitting a standard Modbus packet (random message). Data is distorted with a specified probability of a single-bit error. If errors occur, the message is retransmitted.

Model 2: A channel where the Modbus packet is encoded using a BCH encoder. Data is distorted with a given error probability and decoded at the receiver. If errors persist beyond the code's correction capacity, the message is retransmitted.

Model 3: A channel implementing adaptive coding. If the packet length exceeds a predefined threshold, the data is encoded with a BCH encoder; otherwise, it is sent in its original form. Decoding and retransmission logic remain consistent with the previous models.

Model 4: A channel where the transmitted data is divided into small segments (link-layer coding) before transmission. These segments are decoded at the receiver, with retransmission occurring upon failure.

In essence, the first model represents a standard Modbus scenario, the second represents Modbus with a BCH add-on, the third implements adaptive coding, and the fourth represents a channel codec.

The simulation evaluated the total number of CPU cycles required to transmit 7000 packets. Transmission and reception occur sequentially, where transmitting N bits requires N clock cycles. The simulation also accounted for the computational costs of checksum calculations, encoding, decoding, and the overhead associated with retransmissions.

Fig. 2 illustrates the relationship between the required number of CPU cycles and the bit error probability in the channel during the transmission of 7000 packets. The blue line represents the time costs for the first model, while the yellow line represents the second.

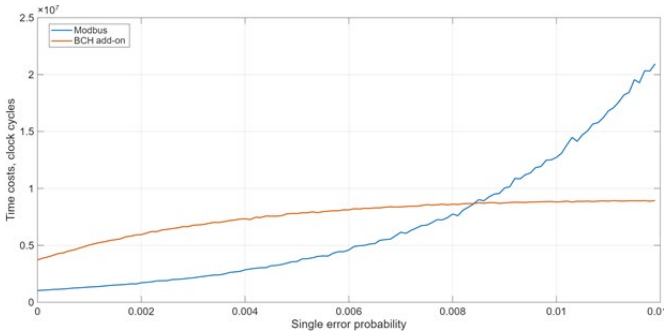

Fig. 2. Graph of time costs versus error probability

As seen in the graph in Fig. 1, for a channel with a relatively high error probability, the temporal costs of the standard Modbus protocol grow much faster than those utilizing linear codes. This observation remains valid even if the number of errors exceeds the code's correction capabilities. This is because if l errors ($l > t$) occur in the channel, the probability that a subsequent retransmission will result in fewer errors is higher than the probability of zero errors occurring in a standard transmission.

In cases of low channel error probability, the second model requires more time for data processing. However, these costs are considered acceptable and justified by the increased reliability and enhanced information integrity during transmission.

Fig. 3 presents a comparison between the BCH add-on and the adaptive BCH codec implementation. The adaptive variant proves superior in terms of packet processing time. On average, the adaptive implementation requires 1.89 times less time to transmit 7000 packets.

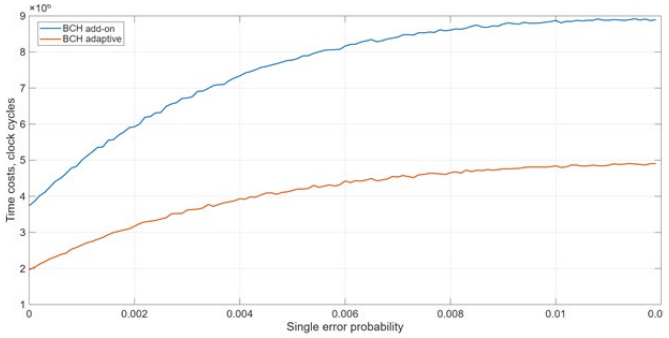

Fig. 3. Comparison of BCH add-on and adaptive coding implementations

Fig. 4 provides a comparison of the temporal costs for the first three models. The graph indicates that for channels with high error probabilities, the standard Modbus protocol is outperformed by models using linear codes. However, adaptive coding remains the closest to the original Modbus protocol in terms of temporal costs at low error probabilities, while spending 1.89 times less time on average than the constant application of BCH.

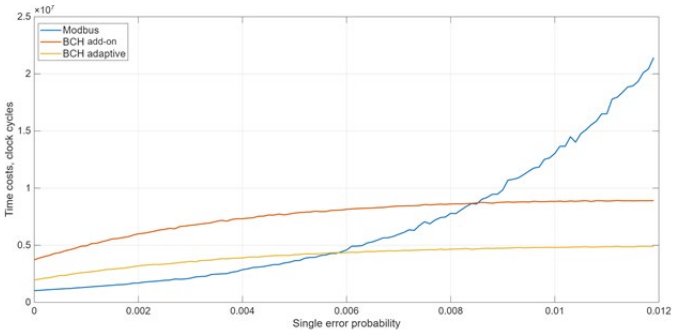

Fig. 4. Comparison of the first three models

Fig. 5 displays the temporal costs for all four models. The graph demonstrates that the link-layer codec model is outperformed by the adaptive codec in terms of overall performance.

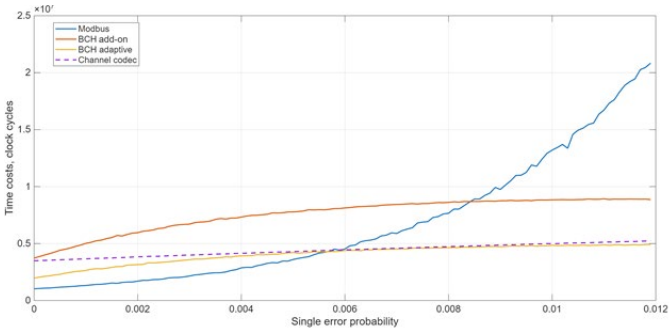

Fig. 5. Temporal costs of all four models

VIII. IMPLEMENTATION OF LINEAR BLOCK CODEC

Based on the data obtained in the previous sections, the BCH algorithm was selected for implementation. Among the linear codes considered, its hardware realization is the most optimal in terms of performance.

The Verilog hardware description language (HDL) was chosen for codec development, utilizing the Xilinx ISE Design Suite 14.7 environment. The simulation workflow is structured as follows:

1. A random bit sequence is generated as the source message;

2. The source message is encoded using a BCH code, resulting in a systematic encoded message consisting of the original information bits and parity bits;

3. A random error is introduced into the encoded message;

4. The corrupted encoded message is fed into the decoder, which outputs a bitmask of the same length as the source message, with non-zero bits indicating the positions of detected errors.

The results of the codec simulation are presented in Fig. 6 below. In addition to data processing information, the figure

includes timing details. The encoding duration is always equal to the codeword length in bits. Within this simulation, estimating the precise decoding duration is more complex; therefore, it is calculated as the difference between the clock counter values at the end of encoding (taken as the start of decoding) and the completion of decoding. While this interval may include overhead for intermediate calculations and may not fully reflect real-world results, it provides a sufficient estimate of temporal costs.

```
Message:            10000000
Encoded:            100000001011
Transmitted:        100000001011(0 errors)
Error vector:       00000000
Encoding time costs:        12(cycles)
Decoding time costs:        110

----------------------
Message:            10101010
Encoded:            101010100100
Transmitted:        101010100110(1 errors)
Error vector:       00000000
Encoding time costs:        12(cycles)
Decoding time costs:        110

----------------------
Message:            00010011
Encoded:            000100111101
Transmitted:        000100101101(1 errors)
Error vector:       00000001
Encoding time costs:        12(cycles)
Decoding time costs:        110
```

Fig. 6. Simulation output

As shown in Fig. 6, the first case depicts a message transmitted without errors. In the second case, an error occurred within the parity bits, resulting in an error vector of zero. In the third case, an error occurred at the eighth bit and was successfully identified by the decoder.

Fig. 7 demonstrates the codec performance when data is transmitted (input and output) via an 8-bit bus, processing 8 bits at a time. In this parallel mode, decoding occurs significantly faster compared to the serial (1-bit-per-cycle) mode.

```
Message:            10101110
Encoded:            110011010001
Transmitted:        110011010001(0 errors)
Error vector:       00000000
Encoding time costs:        12(cycles)
Decoding time costs:        40

----------------------
Message:            00101010
Encoded:            011001100100
Transmitted:        011001100000(1 errors)
Error vector:       00000000
Encoding time costs:        12(cycles)
Decoding time costs:        40

----------------------
Message:            10000101
Encoded:            101100110110
Transmitted:        101100100110(1 errors)
Error vector:       00000001
Encoding time costs:        12(cycles)
Decoding time costs:        40
```

Fig. 7. Simulation output

However, even in this case, the decoder's performance slightly exceeds the theoretical values calculated in previous sections. This discrepancy may stem from several factors, the most likely being a non-optimal implementation of the decoding algorithm. Since the algorithm is a complex mathematical structure, its implementation at such a low hardware level requires high expertise, and the performance is highly sensitive to any sub-optimal coding choices.

Now the codec's throughput can be calculated. The encoder processes 8 information bits in 12 cycles, resulting in an encoder speed of 0.67 bits/cycle. Throughput depends on the system's clock frequency. Table III presents the throughput values relative to the clock frequency.

TABLE III. Encoder throughput.

Clock frequency [MHz]	200	300	400	500
Bandwidth [Mbps]	133,3	200	266,7	333,3

The decoder in serial mode processes 8 information bits in 110 cycles, yielding a speed of 0.073 bits/cycle. Table IV displays the throughput values relative to the clock frequency for this mode.

TABLE IV. Serial decoder throughput.

Clock frequency [MHz]	200	300	400	500
Bandwidth [Mbps]	14,5	21,8	29,1	36,4

The decoder in 8-bit parallel mode processes 8 information bits in 40 cycles, resulting in a speed of 0.2 bits/cycle. Table V presents the corresponding throughput values.

TABLE V. Parallel decoder throughput.

Clock frequency [MHz]	200	300	400	500
Bandwidth [Mbps]	40	60	80	100

In conclusion, the results obtained remain sufficiently performant to ensure high throughput and prevent the codec from becoming a bottleneck within the overall system.

IX. Conclusion

This article provides an overview of possible options for ensuring the Modbus protocol's noise immunity using block linear codes (BCH and LDPC). Algorithms for implementing these codes into the protocol were developed, and their impact on performance was analyzed. According to Matlab modeling, the proposed algorithms insignificantly increase the time costs for the operation of the linear codec and the transmission of excess check bits, while increasing the operating speed several times under conditions of an increased probability of errors occurring during transmission (due to the absence of the need for retransmission). A BCH codec in Verilog was also developed, providing sufficient throughput (14.5 Mbps and higher). Future work may aim to improve the obtained results and create a working prototype using the developed codec and the proposed algorithms.

X. FUTURE RESEARCH

Future research can be focused on the practical implementation of the proposed BCH-based error correction framework within physical industrial automation hardware. While the current Verilog implementation demonstrates a high throughput of 14.5 Mbit/s, further efforts will involve integrating the developed codec into field-programmable gate array (FPGA) modules and specialized microcontrollers to evaluate real-time performance in actual Modbus RTU environments. This stage can include a comprehensive analysis of power consumption and hardware resource utilization across various platforms to ensure compatibility with low-power industrial sensors and programmable logic controllers.

Additionally, subsequent studies may involve a more rigorous comparative testing of the four proposed protocol modification algorithms under extreme interference conditions typical of industrial facilities. Investigating adaptive error-correction schemes, where the coding parameters dynamically adjust based on detected channel quality, presents another promising direction. Finally, the potential for extending this methodology to other legacy industrial communication protocols could further validate the versatility of replacing traditional algorithms like CRC with more robust linear codes.

ACKNOWLEDGMENT

This research was funded by the Ministry of Science and Higher Education of the Russian Science Foundation (Project “Goszadanie” No.075-00003-24-02, FSEE-2024-0003).

REFERENCES

[1] ISO/IEC 18004:2015, "Information technology – Automatic identification and data capture techniques – QR Code bar code symbology specification", International Organization for Standardization, 2015.

[2] ETSI EN 302 307 V1.2.1, "Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)", European Telecommunications Standards Institute (ETSI), 2009.

[3] IEEE Std 802.11-2020, "IEEE Standard for Information technology – Telecommunications and information exchange between systems Local and metropolitan area networks – Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", IEEE Computer Society, 2020.

[4] CCSDS 131.1-B-2, "TM Synchronization and Channel Coding – Summary of Concept and Rationale", Informational Report, Issue 2, Washington, D.C.: Consultative Committee for Space Data Systems, Nov. 2012.

[5] 1. H. Arora, D. Sharma, H. P. Singh and J. P. Singh, "Bit error rate analysis of wireless sensor nodes with different packet size and distance," 2016 International Conference on Advances in Computing, Communication, & Automation (ICACCA) (Spring), Dehradun, India, 2016.

[6] Minamitsumori, Nishinari-ku, MG CO., LTD “Modbus Protocol Reference Guide”, EM-5650 Rev.11.

[7] Peterson, W.W., Error Correcting Codes, MIT Press, Cambridge, MA, 1961.

[8] Lin S., Costello D. J. Error Control Coding: Fundamentals and applications. – Prentice-Hall, 1983.

[9] Sudhir Bommena, standard ISO/TS 16949:2002 AN1148 “Cyclic Redundancy Check (CRC)”, Microchip Technology Inc.

[10] Yathiraj H U, Mahasiddayya R Hiremath “Implementation of BCH Code (n, k) Encoder and Decoder for Multiple Error Correction Control”, International Journal of Computer Science and Mobile Applications, Vol.2 Issue. 5, May- 2014, pg. 45-54.

[11] Y. -M. Lin, H. -C. Chang and C. -Y. Lee, "Improved High Code-Rate Soft BCH Decoder Architectures With One Extra Error Compensation," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, no. 11, pp. 2160-2164, Nov. 2013.

[12] Dejan Azinović, Klaus Tittelbach-Helmrich, Zoran Stamenković, “Performance investigation on BCH codec implementations”, in IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), 27 March 2017.

[13] Liao, Xiong & Guo, Junxiong & Luo, Zhenghua & Xu, Yanghui & Chu, Yingjun. (2023). Research and Implementation of High-Efficiency and Low-Complexity LDPC Coding Algorithm. Electronics. 12. 3696. 10.3390/electronics12173696.

[14] V. Pignoly, B. Le Gal, C. Jego, B. Gadat and L. Barthe, "Fair comparison of hardware and software LDPC decoder implementations for SDR space links," 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Glasgow, UK, 2020.

[15] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," in IEEE Transactions on Information Theory, vol. 47, no. 2, pp. 638-656, Feb 2001.