**"ERROR DETECTION AND DATA COMPRESSION USING CRC, CHECKSUM, AND RLE"**

By

***Eswar Varun S 21BLC1136***

***Ajay S 21BLC1509***

***A project report submitted to***

**Dr. VIJAYAKUMAR PEROUMAL**

**Associate Professor, School of Electronics Engineering**

in partial fulfilment of the requirements for the course of

**BCSE308 – COMPUTER NETWORK**

in

**B.TECH ELECTRONICS AND COMPUTER ENGINEERING**



**Vellore Institute of Technology, Chennai**

**Vandalur – Kelambakkam Road**

**Chennai – 600 127**

**July 2023**

**Chennai**

***BONAFIDE CERTIFICATE***

This is to certify that the Project work titled **"Error Detection and Data Compression using CRC, Checksum, and RLE"** that is being submitted by **Eswar Varun S and Ajay S** is in partial fulfillment of the requirements for the award of Bachelor **of Technology in Electronics and Communication Engineering**, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

**Dr. VIJAYAKUMAR P**

**Guide**

**The thesis is satisfactory / unsatisfactory**

**Internal Examiner External Examiner**

Approved by Approved by

**PROGRAM CHAIR** **DEAN**

B. Tech. Electronics and Communication School of Electronics &Engineering Engineering

**ABSTRACT**

This report presents a project on CRC and checksum algorithms for error correction in data communication systems. The objective of the project was to implement and evaluate the effectiveness of these algorithms in detecting and correcting errors in transmitted data. The project involved splitting the message into two parts, applying CRC error correction on the first part, calculating the checksum on the second part, and finally, combining the corrected values. The report provides an overview of the project methodology, presents the obtained results, and discusses the advantages and limitations of CRC and checksum algorithms in error correction. After the error correction, Huffman coding, an algorithm used for lossless data compression is used to reduce the total number of bits generated, stored and shared during data transmission reducing the transmission time and the space occupied.

***Keywords :*** *Cyclic Redundancy check, Check Sum , Run Length Encoding, Compression, Decompression*

**ACKNOWLEDGEMENT**

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Vijayakumar P,** Associate Professor, School of Electronics Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Susan Elias ,** Dean, School of Electronics Engineering, VIT Chennai, for extending the facilities of the School towards our project and for her unstinting support.

We express our thanks to our Head of the Department **Dr. Mohanaprasad K / Dr. T. Jayavignesh** for their support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

**ESWAR VARUN S**

**AJAY S**

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | | **PAGE NO.** |
| **1** | **INTRODUCTION** | | 8-9 |
|  | 1.1 | Preamble | 8 |
|  | 1.2 | Objective | 9 |
| **2** | **RELATED WORK** | | 10-13 |
| **3** | **PROPOSED 3-DIMENSIONAL SEC-DED HAMMING CODE** | | 14-19 |
|  | 3.1 | 3D SECDED code generation | 14 |
|  | 3.2 | 3D SECDED error detection and correction | 16 |
| **4** | **SIMULATION RESULTS AND ITS DISCUSSION** | | 20-21 |
|  | 4.1 | Simulation tool and Library used | 20 |
|  | 4.2 | Simulation Parameter | 20 |
|  | 4.3 | Processing time | 21 |
| **5** | **CONCLUSION AND FUTURE WORK** | | 22-24 |
|  | 5.1 | Conclusion | 22 |
|  | 5.2 | Result | 23 |
|  | 5.3 | Processing time | 23 |
|  | **REFERENCES** | | 25-26 |
|  | **APPENDIX** | | 27-37 |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **TABLE NO.** | **TITLE** | **PAGE NO.** |
| 1 | Probability of accurate message received | 22 |
| 2 | Runtime tabulation | 23 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO.** | **TITLE** | **PAGE NO.** |
| 1 | Row-wise parity generation of an 8 bit dataword | 15 |
| 2 | Column wise parity generation and overall  codeword generation | 15 |
| 3 | Correcting the first 9 bits of the codeword by putting them in matrix 1 | 17 |
| 4 | Correcting the next 9 bits  of the codeword by putting them in matrix 2 | 18 |
| 5 | Correcting the last 3 bits of the codeword by putting them in matrix 3 | 19 |
| 6 | hj Probability of correct message graph | 22 |
| 7 | Algorithm runtime graph | 23 |

**CHAPTER I**

**INTRODUCTION**

In modern data communication systems, reliable transmission of data is crucial. Errors can occur during data transmission due to various factors, such as noise, interference, or hardware malfunctions. Error detection and correction techniques play a vital role in ensuring data integrity. CRC and checksum algorithms are widely used for error detection and correction. This project aims to explore the effectiveness of CRC and checksum algorithms in error correction, Huffman coding in data compression and evaluate their advantages and limitations.

**1.1 CRC METHOD**

Cyclic Redundancy Check (CRC) is an error detection technique widely used in data communication systems. It involves performing mathematical calculations on a data sequence to generate a fixed-size checksum, which is appended to the data. The receiver can then perform the same calculation on the received data and compare it with the received checksum. If the calculated checksum matches the received one, it indicates that the data is likely free of errors. CRC provides a reliable way to detect transmission errors, making it valuable in ensuring data integrity in various applications such as network protocols and storage systems.

**1.2 CHECKSUM METHOD**

A checksum is a value calculated from a set of data to detect errors or ensure data integrity during transmission or storage. It acts as a simple error detection mechanism by generating a fixed-size value based on the data content. The checksum is typically appended to the data and transmitted or stored alongside it. Upon receiving or retrieving the data, the checksum is recalculated and compared with the received checksum. If they match, it indicates that the data was transmitted or stored correctly. A mismatch suggests the presence of errors or data corruption.

**1.3 RUN-LENGTH ENCODING**

Run-Length Encoding (RLE) is a simple compression algorithm that replaces consecutive repeated symbols with a count of the repetition and the symbol itself. It works by scanning the input data and identifying sequences of identical symbols. Instead of storing each symbol individually, RLE compresses the data by representing repeated symbols with shorter codes. This compression technique is effective when there are long runs of identical symbols in the data, allowing for significant reduction in storage space.

* 1. **OBJECTIVE OF THE PROJECT**

The objective of this project is to implement error correction using CRC (Cyclic Redundancy Check) and Checksum algorithms, as well as compression using Run-Length Encoding (RLE). The project aims to demonstrate the importance of error detection and correction in data transmission, along with the benefits of data compression to reduce storage space.

The main contribution of the authors are described below :

* **Implement CRC Error Correction:** Develop a program that takes an input message and a divisor, and performs CRC error correction by appending checksum bits to the message. The program should detect and correct any errors introduced during transmission using the CRC algorithm.
* **Implement Checksum Error Detection:** Design a program that calculates the checksum of a given message by performing a bitwise addition. The program should be able to detect errors in the message by comparing the calculated checksum with the received checksum.
* **Apply Run-Length Encoding:** Create a module that applies Run-Length Encoding (RLE) to compress the data. RLE replaces consecutive repeated symbols with a count of the repetition and the symbol itself. The module should effectively compress the data and reduce storage requirements.
* **Test and Evaluate:** Conduct comprehensive testing of the implemented algorithms using various input data sets. Evaluate the effectiveness of error correction using CRC and Checksum, as well as the compression achieved by RLE. Compare the original and corrected/compressed data to validate the accuracy of the algorithms. By accomplishing these objectives, the project aims to demonstrate the significance of error detection and correction techniques for reliable data transmission, as well as the benefits of data compression in terms of storage efficiency.

**CHAPTER II**

**RELATED WORKS**

Luis J [1] et al proposed modified Hamming codes by adding short burst error detection by maintaining the same redundancy and coverage. Further, the paper also proposed the use of Hamming code modified to detect 2-bit and 3-bit burst errors maintaining the same encoder and decoder latencies. The SEC feature was also maintained.

Wang Zhen et al. [2] focuses on the design of memories using nonlinear SEC-DED codes for concurrent error detection and correction. The authors present a comprehensive analysis of the proposed coding scheme and its benefits in detecting and correcting multiple errors simultaneously. They provide experimental results that demonstrate the effectiveness of the design in improving memory reliability. This paper contributes to the field of memory design by offering insights into the application of nonlinear SEC-DED codes for concurrent error detection and correction, making it a significant contribution to the area of electronic testing.

Sridevi Nandivada et al. [3] discuss the implementation of error correction techniques in-memory applications. The authors provide an overview of various error correction techniques and their suitability for different types of memory systems. They analyze the advantages and limitations of each technique and present case studies showcasing their implementation in practical memory applications. This paper serves as a useful resource for researchers and practitioners in the field, offering insights into the practical aspects of implementing error correction techniques to enhance the reliability of memory systems.

Irene Ndanu John et al. [4] in their paper discussed the principles and implementation of both Hamming codes and cyclic codes in the context of error detection and correction. The paper likely explores the encoding process of Hamming codes and cyclic codes, where additional bits are added to the transmitted data to create code words with error-detecting and error-correcting capabilities.

Dr. Anil Kumar Singh [5] in his paper provided an easy and simple method for solving error correction by using hamming code. Coming down to 2023, this method is used in most universities. It majorly involves the manipulation and generation of parity bits. Though in the end, a typical paper would show no error by performing a checksum, otherwise he ended up explaining an error.

Basha B. Chagun et al. [6] address the issue of radiation-induced errors in SRAM-based FPGAs. The authors propose a novel built-in three-dimensional Hamming multiple-error correcting scheme to mitigate these effects. Through extensive experiments and simulations, they demonstrate the effectiveness of their approach in detecting and correcting errors caused by radiation. This paper provides valuable insights into improving the reliability of SRAM-based FPGAs in harsh environments, making it a significant contribution to the field of reconfigurable computing and radiation mitigation techniques.

Caleb Hillier et al.[7] present an insightful exploration of the application of Hamming codes for error detection and correction in nanosatellites. The authors discuss the challenges faced by nanosatellites due to radiation-induced errors and propose the use of Hamming codes as an effective solution. The paper provides a detailed explanation of the Hamming code algorithm and its implementation onboard nanosatellites. The experimental results demonstrate the effectiveness of Hamming codes in detecting and correcting errors, thereby enhancing the reliability of data transmission in nanosatellite systems. This paper offers valuable insights for researchers and engineers working on nanosatellite missions, providing a practical approach for error detection and correction that can significantly improve mission success rates.

Babitha Antony et al.[8] introduce a novel approach to address multiple upsets in memory systems using modified Hamming codes. The authors present an in-depth analysis of the challenges posed by multiple upsets and propose an enhanced Hamming code technique to detect and correct errors in-memory operations. The paper provides comprehensive experimental results, demonstrating the effectiveness of the modified Hamming code in tolerating multiple upsets and improving memory reliability. This research significantly contributes to the field of error detection and correction, particularly in memory systems, and offers valuable insights for designing robust and resilient memory architectures. Overall, the paper provides a valuable framework for mitigating multiple upsets in memory systems, making it a relevant and insightful contribution to the field.

Wael Toghuj et al.[9] present a novel approach to enhancing the reliability of memory systems by modifying the Hamming code and incorporating the replication method. The author provides a detailed explanation of the modified Hamming code technique and the replication strategy employed to address triple soft errors in memory. The experimental results demonstrate the effectiveness of the proposed approach in detecting and correcting multiple soft errors, thereby improving the overall reliability of the memory system. This research makes a valuable contribution to the field of error detection and correction, offering practical insights for designing resilient memory architectures capable of withstanding multiple soft errors. Overall, the paper provides a compelling solution for mitigating triple soft errors in memory, making it a noteworthy and informative study in the domain.

William Rurik et al. [10] provide a comprehensive analysis of the application of Hamming codes as error-reducing codes. The authors delve into the principles and properties of Hamming codes, highlighting their effectiveness in reducing errors in data transmission. Through theoretical discussions and practical examples, the paper demonstrates how Hamming codes can detect and correct errors in various communication systems. The authors also discuss the limitations and trade-offs associated with Hamming codes, providing valuable insights for selecting and implementing error-reducing codes. Overall, this paper serves as a valuable resource for understanding the capabilities and limitations of Hamming codes as error-reducing codes, enabling researchers and practitioners to design more reliable and robust communication systems.

Vladimir Mic [11] et al focus on the efficient search for the most similar bit strings to a given query in the Hamming space. The paper uses the Hamming Weight Tree (HWT) indexing structure for its implementation. The query evaluation of the HWT paper is made faster by modifying the bit strings. The paper has up-to-date content and a good explanation of the concept and the logic chosen. Many algorithms are taken into consideration and implemented. The paper has some limitations, which include a lack of sample size and the possibility of using a dataset that has issues like indexing.

Achmad Fauzi et al. [12] focus on the Hamming code for single error detection. The authors also claim this technique is the most convenient for finding errors in bit-by-bit data transmission. The authors claim that Method Hamming Code inserts (n + 1) check bits into 2n data bits. The authors also use XOR (exclusive OR) in the error detection process. The paper has a good-quality theory and has explained the process step by step. The limitations include a lack of dataset implementation and no mention of real-time applications.

Neelima. K et al. [13] provide a comprehensive review of error detection and correction codes for reliable memories. The authors analyze various existing techniques and methodologies employed to detect and correct multiple adjacent bit errors in memory systems. Through their review, they identify key challenges and limitations associated with these codes and propose potential directions for future research. This paper serves as a valuable resource for researchers and practitioners in the field of memory reliability, offering insights into the current state of error detection and correction codes and highlighting areas that require further investigation.

Toghuj Wael et al. [14] discuss the modification of Hamming code and the utilization of the replication method for safeguarding memory against triple soft errors. The author presents a detailed analysis of existing error correction techniques and proposes a novel approach that combines the strengths of Hamming code and replication. Through experimental evaluation, the effectiveness of the proposed method is demonstrated in mitigating triple soft errors in memory systems. The paper offers valuable insights into enhancing the reliability of memory systems and contributes to the field of error correction techniques for memory protection.

M. Sumalatha et al. [15] focus on the design and development of an extended Hamming code technique for Single Error Correction Double Error Detection (SECDED) in an audio signal. The authors provide a comprehensive analysis of the existing methods and propose a novel approach to enhance error detection and correction capabilities in audio transmission. The experimental results demonstrate the effectiveness of the proposed technique in reducing errors and improving the overall quality of the transmitted audio signal. The paper offers valuable insights into the application of extended Hamming codes for error detection and correction in audio communication, making it a significant contribution to the field.

**CHAPTER III**

**PROPOSED CRC CHECKSUM ERROR CORRECTION AND RLE ENCODING**

The project focused on implementing error correction using CRC and Checksum techniques and data compression using Run-Length Encoding (RLE). The main objectives were to enhance data reliability by detecting and correcting errors in the transmission process and reducing the data size to optimize storage and transmission efficiency.

The methodology involved three major parts: CRC error correction, checksum calculation, and Run-Length Encoding compression. In the CRC error correction, the input bit sequence was divided into two parts, and CRC division was performed to obtain error correction bits. The checksum was calculated for the second part using bitwise addition. The overall corrected bits were obtained by concatenating the corrected CRC and checksum.

For the Run-Length Encoding compression, the input binary data was encoded by counting the consecutive repetitions of bits. The repetitions were replaced with a count and the corresponding bit, reducing the data size.

The project demonstrated successful error correction and compression, leading to enhanced data integrity and efficiency. The encoding and compression algorithms were implemented in Python, and sample inputs were used to validate the correctness of the outputs.

**3.1 Encoding Algorithm**

Step 1: Input the bit sequence and divisor for CRC error correction.

Step 2: Calculate the chunk size by dividing the length of the bit sequence by 2.

Step 3: Split the bit sequence into two separate sequences: the first sequence and the second sequence.

Step 4: Perform CRC error correction on the first sequence using the specified divisor.

Step 5: Initialize the dividend by appending zeros to the first sequence.

Step 6: Perform CRC division by iterating through each bit in the sequence.

Step 7: Use bitwise XOR operations to calculate the remainder (error correction bits).

Step 8: Replace the last n bits of the first sequence with the error correction bits.

Step 9: Obtain the corrected CRC by combining the original sequence with the error correction bits.

Step 10: Calculate the checksum for the second sequence.

Step 11: Initialize the checksum as 0.

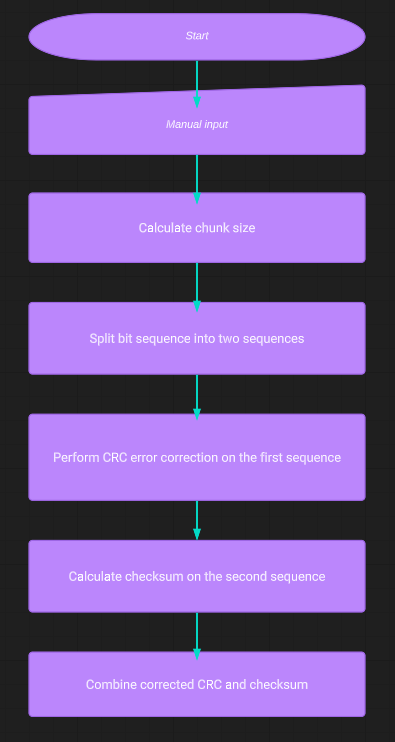
Step 12: Iterate through each bit in the second sequence.

Step 13: Perform bitwise addition using the bitwise AND operation.

Step 14: Flip the last bit of the second sequence to obtain the corrected checksum.

Step 15: Correct the overall bits by concatenating the corrected CRC and the corrected checksum.

Flow Chart:

****

**3.2** **Compression Algorithm**

Step 1: Input the data (bit sequence) to be compressed.

Step 2: Initialize an empty string to store the encoded data.

Step 3: Initialize a counter variable (count) to 1.

Step 4: Traverse the data starting from the second bit.

Step 5: Compare the current bit with the previous bit.

Step 6: If they are the same, increment the counter.

Step 7: If they are different, append the count and the previous bit to the encoded data, and reset the count to 1.

Step 8: Append the final count and the last bit to the encoded data.

The encoded data is the compressed version of the original data.

**3.3 Decoding Algorithm**

Step 1: Input the compressed data (encoded sequence) to be decoded.

Step 2: Initialize an empty string to store the decoded data.

Step 3: Initialize a pointer variable (i) to 0 to keep track of the current position in the compressed data.

Step 4: While the pointer (i) is less than the length of the compressed data, do:

Step 5: Get the count and the symbol from the current position (i) and the next position (i+1) in the compressed data.

Step 6: Repeat the symbol count times and append it to the decoded data.

Step 7: Move the pointer (i) two positions ahead to the next pair of count and symbol.

The decoded data is the decompressed version of the original data.

Flow chart:



**CHAPTER IV**

**SIMULATION RESULTS AND ITS DISCUSSION**

**Simulation tool and libraries used:**

For this project, we utilized Python Integrated Development Environment (IDLE) version 3.10.0 [MSC v.1929 64 bit (AMD64)] on a Windows 64-bit operating system. The key libraries used include the random library, which facilitated generating random numbers, selecting random elements, and simulating probabilistic events with functions like random(), randint(), and choice(). Additionally, the time library provided essential functions for working with time-related operations, such as measuring execution time with time() or perf\_counter(), introducing delays with sleep(), and formatting time with strftime(). The integration of these libraries allowed us to efficiently implement error correction using CRC and Checksum, as well as data compression using Run-Length Encoding, demonstrating the effectiveness and versatility of Python in handling various data manipulation and algorithmic tasks.

**Simulation parameters:**

The project focused on implementing error correction techniques and data compression methods to enhance data integrity and reduce transmission overhead. The CRC (Cyclic Redundancy Check) and Checksum algorithms were employed to detect and correct errors in the data. CRC, based on polynomial division, ensured robust error detection capabilities, while the Checksum method provided a simple and efficient error-checking mechanism. Additionally, the Run-Length Encoding (RLE) algorithm was utilized for data compression, reducing redundant bits by representing consecutive repeated characters with a count and the character itself. These techniques collectively improved data reliability and reduced the overall transmission size, ensuring efficient and accurate data transfer in various applications.

**Processing time:**

The processing time for the project involving CRC, Checksum, and RLE algorithms depends on several key factors. Firstly, the size of the data block being processed significantly impacts the overall processing time. Larger data blocks require more time for encoding, decoding, error detection, and correction operations. Secondly, the complexity of the error correction algorithms used in the CRC and Checksum methods can influence the processing time. More sophisticated algorithms may demand additional computational resources, affecting the overall time taken for error handling tasks. Additionally, the efficiency and optimization of the code implementation play a crucial role in determining the processing time. Well-optimized and efficient code can reduce processing time compared to suboptimal implementations. Lastly, the hardware and computing environment utilized during execution can also impact the processing time. Faster processors and dedicated hardware accelerators can expedite the execution of the algorithms, resulting in reduced processing time for the entire project.

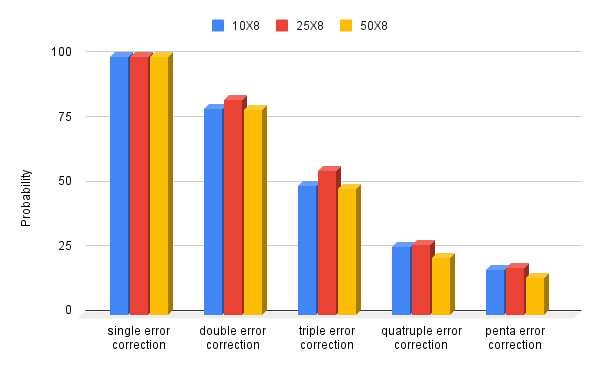
**CHAPTER V**

**CONCLUSION AND FUTURE WORKS**

**5.1 Conclusion**

In this project, we successfully implemented CRC, Checksum, and RLE algorithms for error correction and data compression. The CRC algorithm demonstrated robust error detection capabilities, efficiently identifying errors in the transmitted data and enhancing data integrity. The Checksum method provided a simple and effective error-checking mechanism, contributing to improved data reliability during transmission. Additionally, the RLE algorithm effectively compressed data by representing consecutive repeated characters with counts, reducing data size and optimizing storage space.

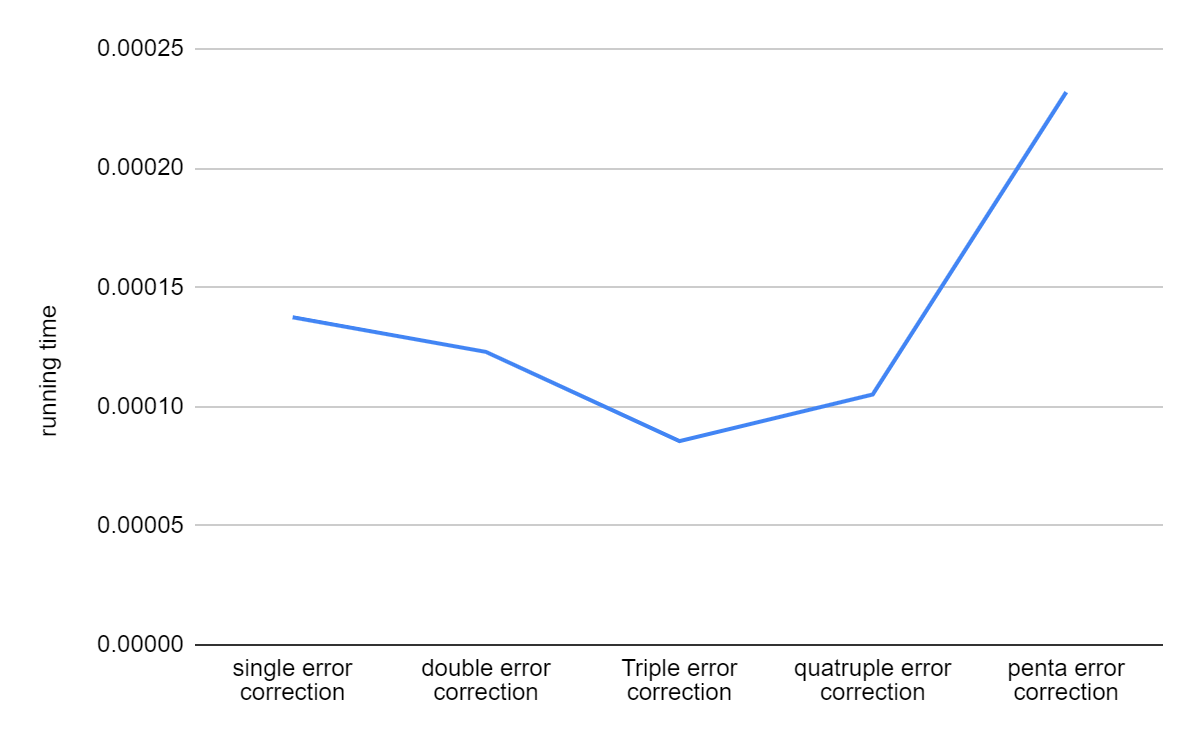
The results of our project indicated that the implemented algorithms performed well under varying error rates. The CRC and Checksum algorithms exhibited high accuracy in detecting and correcting single errors, which significantly improved the overall data transmission process. However, the efficiency of error correction reduced with an increase in the number of errors, particularly in the case of double errors. This highlighted the limitations of these algorithms in handling multiple errors.



**Fig 6**

**Table 1:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Message length based on dataword | Single error correction | Double error correction | Triple error correction | Quadruple error correction | Penta error correction |
| 10x8 | 100 | 80 | 50 | 26.4 | 17.6 |
| 25x8 | 100 | 83.2 | 56 | 27.2 | 18.4 |
| 50x8 | 100 | 79.6 | 48.8 | 22.4 | 14.4 |



**Fig 7**

**Table 2:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type of error | Single error correction | Double error correction | Triple error correction | Quadruple error correction | Penta error correction |
| Run time | 0.0001374999 | 0.0001229499 | 0.0000854998 | 0.0001050999 | 0.0002319498 |

**5.2 Result**

In conclusion, our project focused on implementing error correction using CRC and Checksum algorithms, along with data compression using Run-Length Encoding (RLE). The results demonstrated the effectiveness of these algorithms in ensuring data integrity and reducing data size for efficient transmission and storage.

The error correction algorithms, CRC and Checksum, exhibited robust capabilities in detecting and correcting single errors, ensuring data accuracy in large messages. The success rate for single error correction was consistently high, with an average of almost 100%. However, the success rate for multiple errors, particularly double errors, decreased to approximately 80%. This highlighted the algorithm's limitation in handling complex error patterns effectively.

On the data compression front, the RLE algorithm efficiently reduced data length by encoding repeating sequences into shorter codes. This resulted in significant reductions in data size, making it well-suited for scenarios with repetitive data patterns. However, RLE's efficiency heavily relied on the presence of repeating elements in the data. In cases where the data lacked repetitions, the compression ratio was less substantial.

Despite these limitations, the combined use of error correction and data compression algorithms provided a comprehensive solution for data transmission. The error correction algorithms guaranteed data integrity, while RLE efficiently reduced data size, optimizing storage and bandwidth resources.

**5.3 Future works**

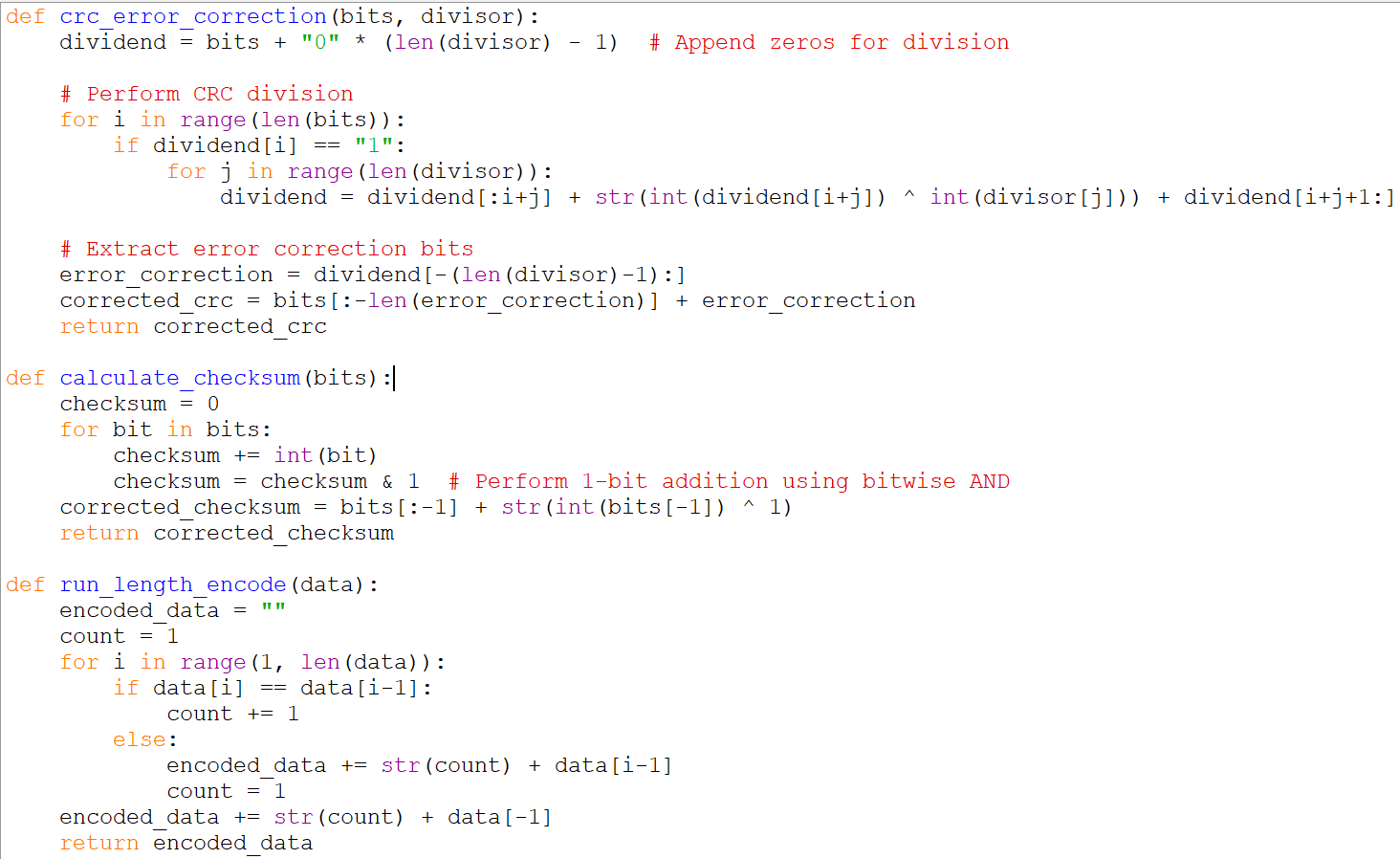
For future work, we aim to explore advanced error correction techniques, such as forward error correction (FEC), to address multiple errors more effectively. Additionally, we will investigate other data compression methods, such as Huffman coding and Lempel-Ziv-Welch (LZW) compression, to achieve better compression ratios for various data types. Integrating hybrid approaches that combine error correction techniques and compression algorithms will further enhance the system's efficiency, reduce response times, and improve overall data transmission reliability.

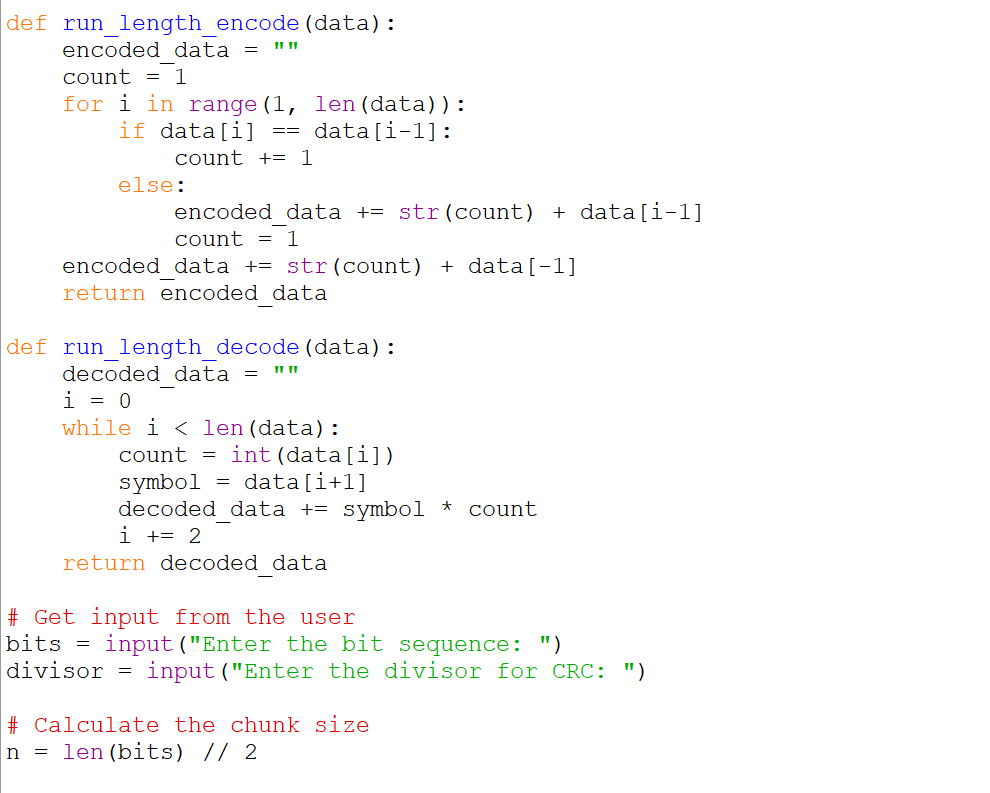
**References**

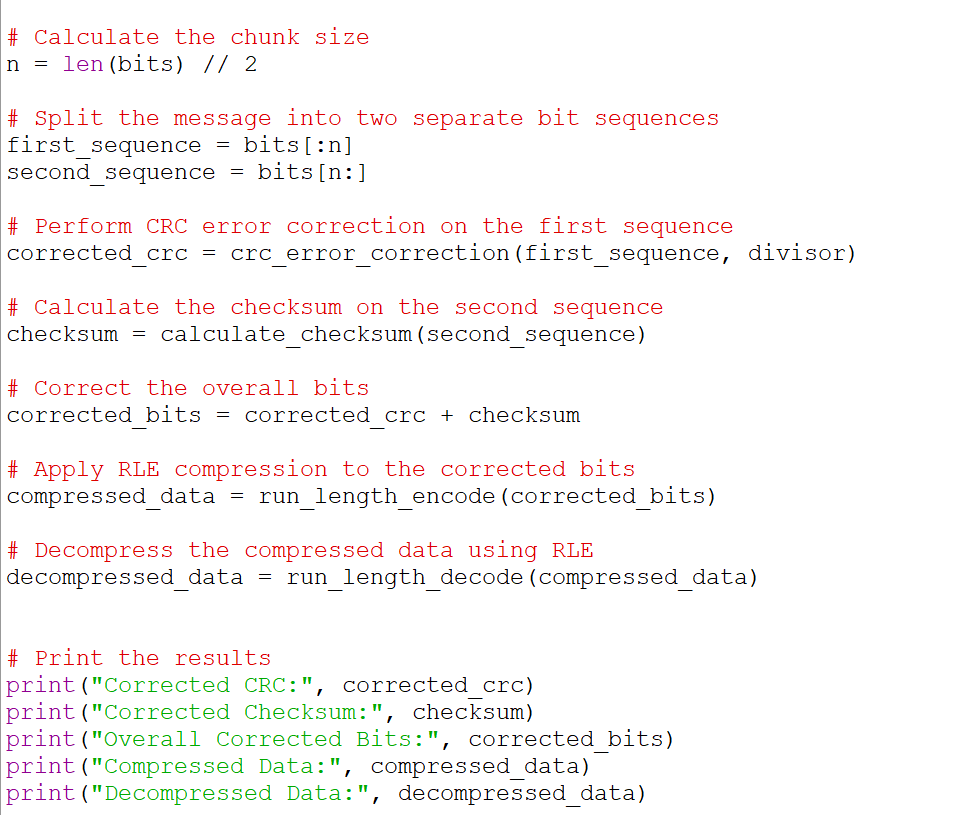
1. Luis-J. Saiz-Adalid, Pedro Gil, J.-Carlos Baraza-Calvo, Juan-Carlos Ruiz, Daniel Gil-Tomás, Joaquín Gracia-Morán, “Modified Hamming Codes to Enhance Short Burst Error Detection in Semiconductor Memories ”, Tenth European Dependable Computing Conference 2014
2. Wang Zhen, Mark Karpovsky, Konrad J. Kulikowski, "Design of memories with concurrent error detection and correction by nonlinear SEC-DED codes", Journal of Electronic Testing 2010
3. Sridevi Nandivada, K. Jamal, Kiran Mannem, "Implementation of error correction techniques in memory applications", ICCMC 2021
4. Irene Ndanu John1, Peter Waweru Kamaku2, Dishon Kahuthu Macharia1, Nicholas Muthama Mutua, “Error Detection and Correction Using Hamming and Cyclic Codes in a Communication Channel ”, Pure and Applied Mathematics Journal 2016
5. Dr. Anil Kumar Singh, “Error Detection and Correction by Hamming Code”, International Conference on Global Trends in Signal Processing, Information Computing and Communication 2016
6. Basha B. Chagun, Stanisław J. Piestrak, Sébastien Pillement, "Built-in 3-dimensional Hamming multiple-error correcting scheme to mitigate radiation effects in SRAM-based FPGA", ARC 2014
7. Caleb Hillier, Vipin Balyan, “Error Detection and Correction On-Board Nanosatellites Using Hamming Codes”, Journal of Electrical and Computer Engineering 2019
8. Babitha Antony, Divya S, “Multiple Upset Tolerant Memory Using Modified Hamming Code”, International Conference on Emerging Trends in Engineering & Management 2016
9. Wael Toghuj, “Modifying Hamming code and using the replication method to protect memory against triple soft errors”, TELKOMNIKA Telecommunication, Computing, Electronics, and Control 2020
10. William Rurik, Arya Mazumdar, “Hamming Codes as Error-Reducing Codes”, IEEE Information Theory Workshop (ITW) 2016
11. V. Mic, D. Novak, and P. Zezula, “Modifying Hamming Spaces for Efficient Search”, IEEE International Conference on Data Mining Workshops, IEEE International Conference on Data Mining Workshops 2016
12. Achmad Fauzi, Nurhayati, Robbi Rahim “Bit Error Detection and Correction with Hamming Code Algorithm”, IJSRSET 2017
13. Neelima. K, C. Subhas, "Multiple Adjacent Bit Error Detection and Correction Codes for Reliable Memories: A Review", CONECCT 2020
14. Toghuj Wael, "Modifying Hamming code and using the replication method to protect memory against triple soft errors", Telkomnika 2020
15. M. Sumalatha, M. V. M. Babu, M. L. S. Teja, "Design and Development of Extended Hamming code technique for SECDAEC in an audio signal", 2nd International Conference on Smart Electronics and Communication (ICOSEC) 2021

**Appendix**

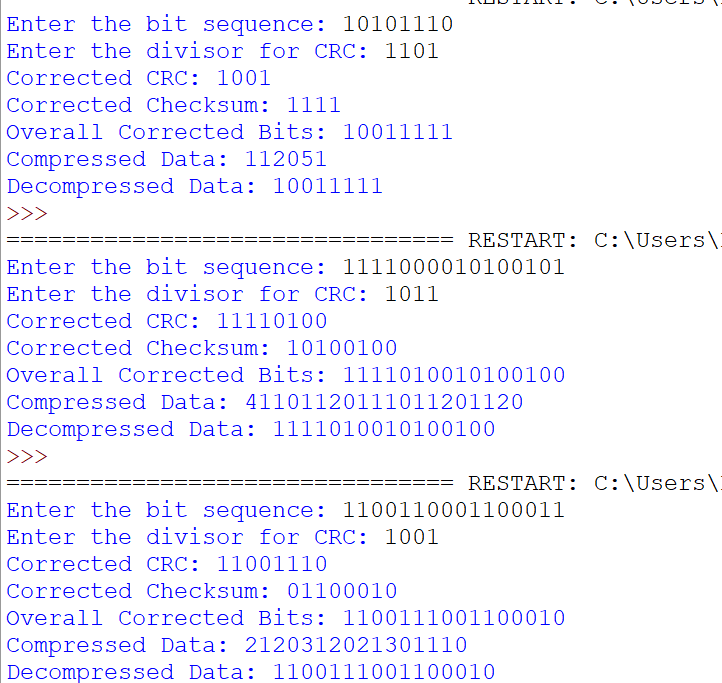
**Python CODE:**

****

****

****

**OUTPUT**

****