Journal of Technology Informatics and Engineering (JTIE)
Vol. 3 No. 2 August 2024
E-ISSN: 2987-9434; P-ISSN : 2988-0343, Pages 169-187

Reliability of Arduino Serial Communication Systems: A Case Study on the Application of Cyclic Redundancy Check (CRC)

Budi Raharjo*[1], Ahmad Zainudin[1]
Email: budiraharjo@stekom.ac.id, zaenudin@stekom.ac.id
Orcid: 0000-0001-6192-0888, 0009-0007-7069-6452
[1]*University of Science and Computer Technology, Semarang, Indonesia, 50192*
*Corresponding Author

Abstract
In embedded systems, serial communication plays a crucial role in data transfer, particularly in Arduino-based projects. However, factors such as electromagnetic interference, noise, and signal degradation can compromise data integrity, leading to significant errors. Effective error detection systems are essential to ensure reliable data exchange. The Cyclic Redundancy Check (CRC) is one such method known for its ability to detect errors. Despite its potential, the practical application and impact of CRC on Arduino communication systems have not been extensively explored. This study implements CRC within Arduino serial communication by designing and developing software that integrates CRC for real-time error detection. The study rigorously tests this implementation in various scenarios to evaluate its performance, comparing data integrity with and without CRC. The results show that incorporating CRC significantly improves the reliability of data transmission in Arduino applications, enhancing error detection accuracy. This improvement strengthens existing systems and provides a solid foundation for developing more complex communication frameworks. The research advances reliable communication systems in embedded technologies. By demonstrating CRC's effectiveness in enhancing data integrity, the study offers valuable insights for developers and researchers seeking to improve serial communication across different applications.

Keywords*: Error Detection, Cyclic Redundancy Check (CRC), Arduino, Serial Communication, Data Integrity.*

I. INTRODUCTION

In the realm of modern data communication, particularly in embedded system applications like Arduino, the reliability of data transmission is paramount. Serial communication, a common method for transferring data between devices, often encounters challenges related to data integrity. Various factors, such as electromagnetic interference, noise, and physical damage to transmission media, can cause errors in the transmitted data. Research by (Gunduz et al., 2023; Yang et al., 2023) indicates that error rates in data transmission can reach up to 10% under certain conditions, leading to serious consequences for applications that depend on accurate data. Although many communication systems currently utilize basic error detection methods such as parity bits or checksums (Das & Touba, 2020; Hadi & Mohammed, 2024; Rajeswari, 2021), these methods often prove insufficient to address the wide range of errors that may occur. For example, parity bits (Häring, 2021; Senekane et al., 2021) can only detect single-bit errors, while checksums may not be sensitive enough to detect more complex errors or significant data

Received on May 29, 2024; Revised on June 22, 2024; Accepted on July 25, 2024; Published on August 21, 2024.
Doi: 10.51903/jtie.v3i2.186

alterations. This creates a gap in existing error detection systems, where data integrity cannot be fully guaranteed. Previous studies suggest that more advanced error detection methods, such as Cyclic Redundancy Check (CRC) (Hadi & Mohammed, 2024; Li et al., 2024; Yen & Wu, 2006; Yin et al., 2024), offer a more effective solution to this issue.

However, although CRC has proven effective in many applications, its implementation in Arduino-based systems has not been extensively discussed. Many existing studies do not comprehensively explore how CRC can be integrated into serial communication on this platform, nor do they examine its impact on system performance, such as latency and throughput (Alena et al., 2007; Gianioudis et al., 2023. Therefore, a significant gap exists in the literature that needs to be addressed to understand the potential of CRC in enhancing data communication reliability in embedded systems. This research aims to fill that gap by implementing an error detection system using the CRC method in Arduino-based serial communication. The primary focus of this study is to evaluate the effectiveness of CRC in detecting errors and analyze its impact on communication system performance. Through systematic testing and in-depth analysis, this research is expected to provide deeper insights into how CRC can improve reliability and efficiency in data communication applications, as well as offer a solid foundation for the development of more complex communication systems in the future.

II. LITERATURE REVIEW

A. System Implementation

The implementation process serves to ensure that the planned solution can be effectively applied to achieve the desired outcomes. Implementation involves integrating technical solutions into the operational environment and encompasses several stages to ensure success (Yang, K., & Wu, 1995). The initial step in the implementation process is the needs analysis, where the goal is to ensure that the designed solution meets all identified needs and expectations. Following the needs analysis, the process proceeds to the design phase. Here, the solution developed during the analysis is further refined into a more detailed plan. This includes creating the system architecture, technical specifications, and implementation strategies that will be used to deploy the solution. The purpose of the design phase is to create a blueprint that will guide the implementation phase (Rao, S., & Kumar, 2015). System design involves various elements, ranging from technical architecture design to operational details. The system architecture defines how the system components will interact, while the technical specifications describe the necessary technical features. The implementation strategy outlines the concrete steps for integrating the solution into the operational environment.

Once the design plan is ready, the next phase is testing. The purpose of testing is to ensure that the system or solution functions according to the expectations and specifications that have been established. This process involves testing various features and functions to identify and resolve any issues before the solution is fully deployed (Yang & Wu, 1995). Testing can include different types, such as functional testing to ensure that all features work correctly, and performance testing to evaluate the system's speed and efficiency. Any issues identified during this phase must be resolved before the system can be deployed in the operational environment.

After testing is completed and all issues have been addressed, the deployment phase begins. Deployment is the process of fully integrating the solution into the operational environment. This involves system installation, configuration, and the necessary setup to ensure that the solution functions effectively in real-world conditions. During the deployment process, end-user training is often conducted to ensure that users understand how to operate the new system. This training is crucial for ensuring a smooth transition and enabling users to take full advantage of the available features (Jiang & Xu, 2002).

Post-implementation support is also a critical part of the process. After the system is deployed, technical support and maintenance are necessary to address any issues that may arise and to make any required updates or improvements. This support ensures that the system continues to function properly and meets user needs. Overall, implementation is a crucial stage in any project or initiative. The success of the implementation determines how effectively the designed solution can achieve the established goals. By following this process systematically, organizations can ensure that the implemented solution provides maximum benefits and meets the set expectations (Kumar & Jain, 2009).

B. *Error Detection System*

An error detection system is a core technology in the field of information and communication technology, responsible for identifying and handling errors that may arise during data transmission or computational processes. In this context, errors refer to inaccuracies or corruptions in the data being processed or transmitted. This technology is crucial for ensuring that data remains intact and accurate, preventing potential issues that could arise from corrupted data (Lestari & Susanto, 2022). The primary goal of an error detection system is to identify any errors that may occur during data transmission or processing. These errors can manifest as lost or altered bits or interference caused by unstable signals. In some cases, data can also become corrupted during storage. The system functions by checking the data to ensure that the transmitted and received data remain consistent and by triggering correction procedures or retransmission if necessary.

Various error detection methods are employed depending on the specific needs of the application or system. One common method is the checksum technique. This involves calculating a checksum value for the data to be transmitted, which is then sent along with the data. The receiver calculates the checksum of the received data and compares it with the transmitted value. A discrepancy between the two indicates an error in transmission (Patel & Chauhan, 2010). Another method is the parity bit, which adds an additional bit to the data to ensure that the number of 1s or 0s is even (even parity) or odd (odd parity). Although this method is relatively simple, it is effective in detecting small errors, such as single-bit errors. An error is detected if the parity of the received data does not match the expected value.

Hamming code is a more advanced error detection method that can both detect and correct errors. This method works by adding extra bits to the data to create a code capable of detecting and correcting single-bit errors. Hamming code is particularly useful in applications where error correction is essential, such as in critical data communications (Gao & Zheng, 1998). Another widely used technique is the Cyclic Redundancy Check (CRC). CRC involves polynomial division to generate a control value that is sent along with the data. The receiver uses this control value to verify the integrity of the received data. CRC is highly effective in detecting errors in larger and more complex data sets.

The application of error detection systems spans various fields, including computer networks. In the context of networks, these systems ensure that the data received by the destination device matches the data sent by the source. This is essential for maintaining the quality and consistency of communication between devices within a network. In storage systems, such as hard drives or SSDs, error detection systems function to detect and correct errors in stored data. This helps protect data integrity and prevents damage that could result in data loss or system failure (Cheng & Lu, 2007).

In the realm of software, error detection systems are used to handle and report errors that occur during program execution. This includes runtime or logic errors that could impact the application's functionality. With effective error detection systems in place, developers can address issues before the application reaches the end users. The presence of error detection systems allows information technology systems to operate more stably and reliably, reducing the likelihood of issues that could affect end users. Therefore, these systems are essential components in ensuring data integrity and reliability across various applications and systems (Smith & Miller, 2017).

C. *Serial Communication*

Serial communication transmits bits sequentially, unlike parallel communication, which sends multiple bits simultaneously over several lines. This technique is widely utilized in various applications, ranging from computer hardware to long-distance communication and device communication protocols (Wu & Chen, 2021). In serial communication, data is organized into a sequence of bits following a specific format, which typically includes start bits, data bits, stop bits, and parity bits. Start bits signify the beginning of data transmission, data bits contain the actual information being sent, stop bits indicate the end of the data, and parity bits are employed for error detection. This process involves two main devices: the transmitter and the receiver.

The transmitter's role is to convert data into signals for transmission through the communication channel, while the receiver converts these signals back into understandable data. There are two types of serial communication: synchronous and asynchronous, differing in how they manage and synchronize data transmission. In synchronous serial communication, data is sent in coordinated blocks or frames with clock signals. Both the transmitter and receiver use the same clock signal to synchronize the sending and receiving of data. This allows for high data transfer rates and more efficient bandwidth usage. Protocols utilizing synchronous methods include SPI (Seria Peripheral Interface) and I2C (Inter-Integrated Circuit) (Raj & Agarwal, 2005).

Conversely, asynchronous serial communication does not require a clock signal. Data is transmitted with start and stop bits marking the beginning and end of each byte. While this method may reduce data transfer speed, it allows for communication without strict synchronization. Examples of asynchronous protocols are RS-232 and UART (Universal Asynchronous Receiver/Transmitter).

Serial communication has numerous applications in daily life, including computer device communication such as between modems and printers. It is also extensively used in serial networks like RS-485, commonly employed for industrial communication. In embedded systems, serial communication plays a critical role, often providing an efficient and reliable method of communication. Serial communication can be used in industrial control systems or medical devices to transmit data effectively (Lestari & Susanto, 2022). The main advantage of serial communication is the reduction in the number of required cables. This makes serial communication simpler and more cost-effective compared to parallel communication, particularly over long distances. Using a single cable to transmit data sequentially reduces complexity and installation costs.

Additionally, serial communication can address issues of interference and signal degradation that often occur in parallel communication. Since parallel communication involves multiple lines that may interfere with each other, serial communication, with its single cable channel, tends to be more stable and resistant to disruptions. From simple applications such as connecting computer devices to more complex systems requiring fast and stable data transfer, serial communication offers an effective method to meet various communication demands in the modern world (Gong & Wang, 2008).

D. Arduino

Arduino consists of two main elements: hardware and software. The Arduino hardware includes a printed circuit board equipped with a microcontroller and various additional components, while the software is the Integrated Development Environment (IDE) used for writing and uploading code to the Arduino board. The Arduino board typically features a microcontroller that acts as the central processing unit, along with a number of input and output pins that allow users to connect various electronic components such as sensors, motors, and LEDs. It also includes a USB connector for linking to a computer, as well as additional components like voltage regulators, oscillators, and a reset button. Popular Arduino board models include the Arduino Uno, Arduino Mega, and Arduino Nano, each offering different features and capacities to meet various project needs (Amin & Patel, 1999).\

The Arduino IDE is the software used to write, edit, and upload code to the Arduino board. It provides an easy-to-use programming environment that supports C and C++ languages. Programs written in this IDE are called "sketches" and consist of two main functions: `setup()`, which runs once when the board is powered on, and `loop()`, which runs repeatedly while the device is on. The IDE also includes various libraries that facilitate the use of different electronic components and additional modules. One of Arduino’s key strengths is its large and active user community. This community frequently shares projects, tutorials, and source code, making it easier for beginners to learn and develop their skills. Additionally, many companies and individuals develop and sell various shields and modules that can be used with Arduino boards to extend their functionality. This ecosystem supports a wide range of projects, from simple measuring instruments to complex automation systems (Rao & Kumar, 2015).

Arduino has a wide range of applications, from DIY projects and electronic hobbies to prototype development and scientific research. Common applications include creating automated control systems, robotics, environmental sensor data collection, and developing interactive tools. Due to its ease of use and flexibility, Arduino is highly popular among students, engineers, and electronics enthusiasts. Its main advantages include user-friendliness, affordability, and extensive community support. The platform is designed to simplify prototyping and electronic experimentation and supports various sensors and actuators. However, Arduino also has some limitations, such as lower processing and memory capacities compared to other microcontrollers or development boards.

Although the Arduino IDE is easy to use, some users may find its flexibility and features lacking compared to more advanced programming environments. This could be a consideration for those who require more control or features in their project development. With its combination of user-friendly hardware, intuitive software, and strong community support, Arduino facilitates the creation of a wide range of innovative projects with relative ease. Despite its limitations, its advantages make it a popular choice among various users, from beginners to professionals. Arduino also plays a significant role in education and technical skill development, allowing students and developers to gain practical experience with electronics and programming, making it a highly valuable tool in the fields of education and technological innovation (Rao & Kumar, 2015).

E. Cyclic Redundancy Check (CRC) Method

The Cyclic Redundancy Check (CRC) method is a technique employed to identify errors in data transmitted over networks or stored in storage media. CRC is a highly effective error-checking method widely used in various computer and communication applications. This technique ensures that transmitted or stored data remains intact and detects any errors that may occur during these processes. CRC functions as an algorithm that generates a hash value or checksum from the data. This value is used to verify data integrity and detect errors that might arise during transmission or storage. The CRC process involves mathematical operations on data bits to produce a unique checksum, which is then sent with the data or stored for future verification.

In the CRC process, a polynomial divisor, known as the "generator polynomial," is used to divide and process the data. This polynomial determines the method of division and processing during the CRC calculation. Data division is performed using polynomial division, employing XOR (Exclusive OR) operations and bit shifting. The result of this division is the CRC value, which is the remainder of the division and appended to the end of the original data. When the data is received or retrieved, the CRC included with the data (or stored) is used to verify data integrity. This verification is done similarly to the CRC formation: the data with CRC is divided again using the generator polynomial. If the remainder is zero, the data is considered error-free. Conversely, if the remainder is non-zero, it indicates that an error has occurred in the data.

Implementing CRC involves several steps, beginning with the selection of the appropriate generator polynomial, which affects the error-detection capability of the algorithm. The generator polynomial can vary depending on the standards and applications, such as CRC-16-CCITT or CRC-32. Next, the CRC register is initialized with an initial value, usually zero or according to CRC specifications. The data is then processed with the generator polynomial through XOR operations and bit shifting. This process is performed bit by bit, and the final result is the CRC value. After computing the CRC, this value is appended to the end of the data before transmission or storage. When the data is received or accessed, the CRC is recalculated and compared with the received or stored CRC. If the CRC matches, the data is deemed correct; otherwise, the data is considered corrupted.

CRC offers several advantages, including its ability to detect random and burst errors very effectively. This method can identify single-bit errors, double-bit errors, and long burst errors. CRC is also relatively fast and efficient to implement, both in hardware and software, and supports interoperability across various communication protocols and file formats. However, CRC also has limitations. Despite its effectiveness, no error-checking method is perfect. CRC may not detect all types of errors, especially under certain conditions or if the data is severely corrupted. Additionally, selecting the appropriate polynomial is crucial for CRC effectiveness, and using an unsuitable polynomial can reduce its error-detection capabilities. CRC is widely applied in various technological contexts, such as network protocols, storage media, and data compression. In communication protocols like Ethernet and Point-to-Point Protocol (PPP), CRC checks the integrity of transmitted data. In storage media, CRC is used to detect errors in stored data, while in data compression, CRC verifies compressed data to ensure that no errors occur during decompression.

III. MATERIAL AND METHODS

A. System Design

The implementation of an error detection system using the Cyclic Redundancy Check (CRC) method in Arduino-based serial communication is carried out through several systematic steps. The aim of this design is to ensure the integrity of data transmitted between two Arduino microcontrollers. This process involves calculating the CRC value for each data packet sent and verifying the CRC value on the receiver's side.

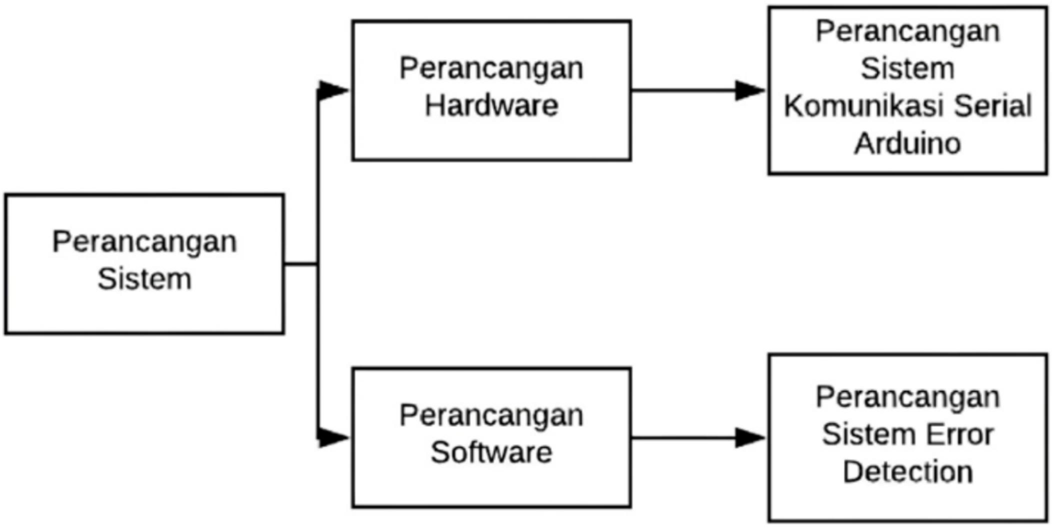

Perancangan Sistem
Perancangan Hardware
Perancangan Sistem Komunikasi Serial Arduino
Perancangan Software
Perancangan Sistem Error Detection

Figure 1. System Design

B. Hardware Design

The hardware design for this system focuses on configuring serial communication between Arduino units using the Rx (Receive) and Tx (Transmit) pins. In this study, pins 10 and 11 on the Arduino are selected for the Rx and Tx functions, respectively. Pin 10 serves as the Tx pin for sending data, while pin 11 functions as the Rx pin for receiving data. The placement of these pins, as illustrated in the diagram below, is designed to ensure efficient and stable data exchange between the Arduino units, which is crucial for meeting the requirements of the serial communication system in this research..

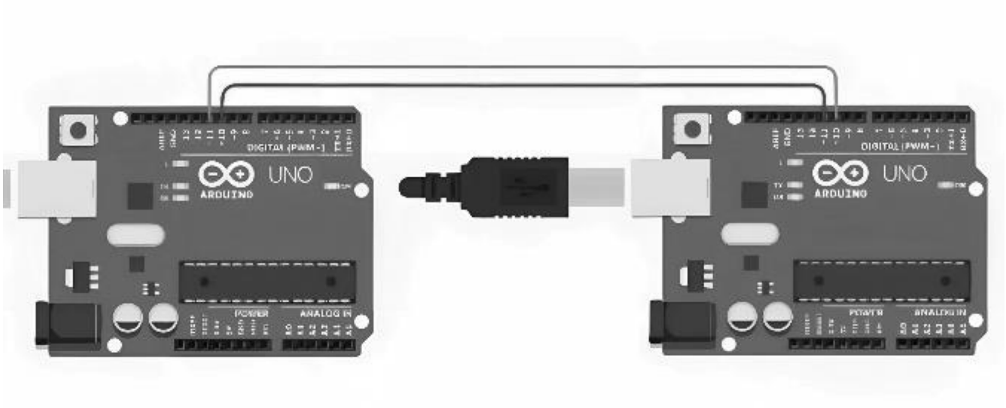
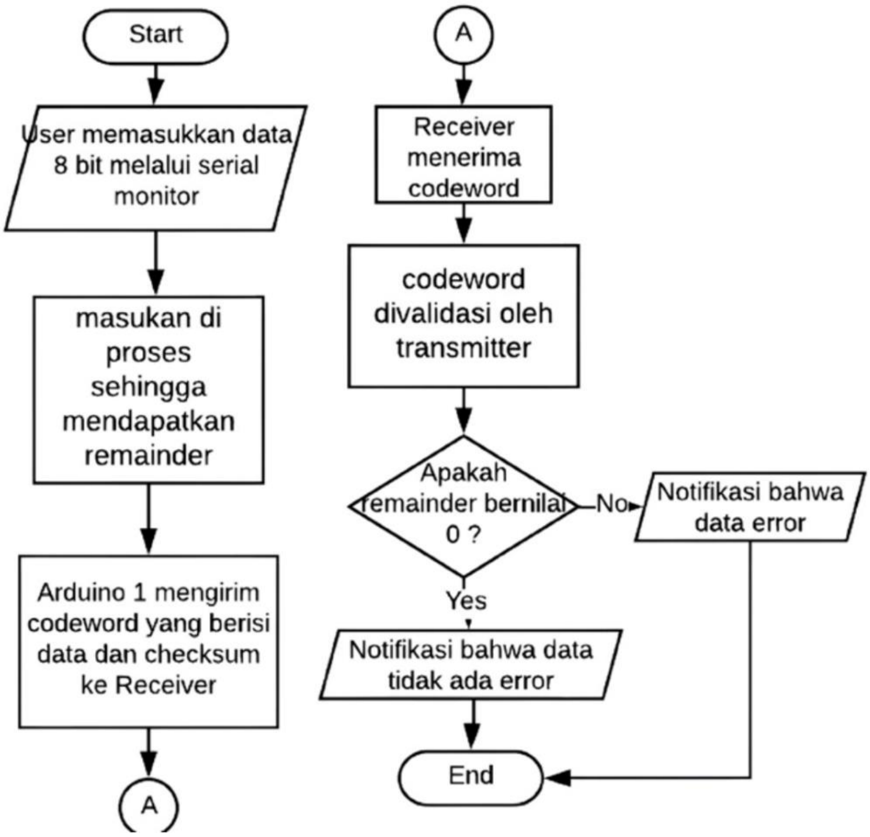


Figure 2. Serial Arduino Design

C. Software Design

The software design for this system encompasses two main components: configuring serial communication on the Arduino and implementing the Cyclic Redundancy Check (CRC) method for error detection. This design focuses on creating scripts to manage data transmission and reception through serial communication, including initializing the serial port, adjusting the baud rate, and setting up data handling processes between Arduino units or with external devices to ensure accurate and efficient transmission. The CRC method is applied to enhance data reliability by adding redundancy codes that check for errors during transmission; the CRC algorithm is used to compute and verify the CRC values of sent and received data, enabling the system to detect and correct errors. The flowchart in the diagram below illustrates the steps and process flow involved in implementing serial communication and the CRC method, showing how data is processed and checked. With this software design, the system is expected to function reliably, manage data with precision, and handle errors effectively.


Start
User memasukkan data 8 bit melalui serial monitor
masukan di proses sehingga mendapatkan remainder
Arduino 1 mengirim codeword yang berisi data dan checksum ke Receiver
A
A
Receiver menerima codeword
codeword divalidasi oleh transmitter
Apakah remainder bernilai 0 ?
No
Notifikasi bahwa data error
Yes
Notifikasi bahwa data tidak ada error
End

Gambar 3. Diagram alir perancangan sistem

D. *Materials and Tools*

- Arduino Microcontrollers: Two Arduino units (e.g., Arduino Uno) are used as the transmitter and receiver.
- Jumper Wires: To connect the two Arduino units.
- Arduino IDE Software: For programming and testing.

- CRC Library: Utilizing available libraries for calculating CRC, such as CRC32 or CRC16.

E. Implementation Procedure

The procedure for implementing a CRC system consists of the following steps:

1. Initialization: In this process, both Arduinos are prepared and connected with jumper cables to ensure a robust connection for serial communication.
2. Data Calculation and Transmission: Before sending data, the CRC value of the data is computed using the chosen CRC algorithm (e.g., CRC-16 or CRC-32) on the sender's side. Next, the CRC value is appended to the end of the data packet, and the complete data packet, including the CRC value, is transmitted via serial communication.
3. Data Reception and Verification: On the receiver's side, the data packet is received, the data and CRC values are separated, and the CRC is recalculated and compared with the received CRC value. If both CRC values match, the data is considered valid; if not, an error is detected, and an error-handling procedure (e.g., requesting a retransmission of the data) may be implemented.

F. Design of Error Detection System Using CRC Method

The design of an error detection system using the Cyclic Redundancy Check (CRC) method in serial communication between two Arduino Uno microcontrollers involves one Arduino acting as the data transmitter and codeword processor, while the other functions as the receiver. The transmitting Arduino receives data from the user via the serial monitor and processes this data to generate a codeword. This codeword combines the original data with a remainder obtained from polynomial division calculations, which involve XOR operations and bit shifts. Once the data and remainder are combined, the codeword is sent to the receiving Arduino. On the receiver's side, the remainder is recalculated using the same method. If the remainder is zero, the data is considered error-free; if the remainder is non-zero, it indicates an error in the received data. The flowchart in Figure 4 illustrates the steps in designing this error detection system, visualizing the process and data verification to ensure integrity during transmission. With this CRC system, the serial communication between the two Arduinos is expected to detect and identify errors, ensuring the reliability of the transmitted and received data.

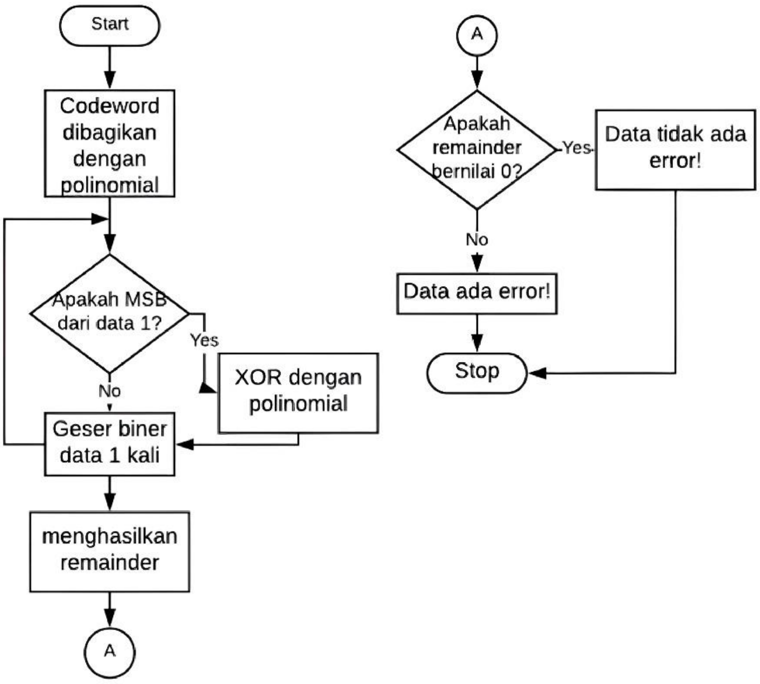


Figure 6. Flowchart for error detection systems

G. Testing and Evaluation

After implementation, conduct testing to evaluate the effectiveness of the CRC system in error detection. Test the system by sending both correct data and modified data to assess whether it can accurately detect errors. Record the test results and analyze the system's performance, including latency and throughput.

IV. RESULT/FINDINGS AND DUSCUSSION

A. System Implementation

During the implementation phase of the serial communication system with Arduino, both hardware and software configurations were conducted to ensure efficient data transmission between components. The Arduino was connected to the serial communication module, and the necessary software was uploaded to the microcontroller to manage the data sending and receiving processes, with testing performed to ensure accuracy and address any potential issues. Additionally, the error detection system was developed to identify and address errors in data transmission, thereby enhancing system reliability. Techniques such as Hamming code or checksums were used to monitor data integrity, and the system was equipped with features to detect and correct errors or signal the user. Testing was carried out to ensure that the error detection system operated effectively without impacting the overall system performance. Figure 5 presents the implementation results, including hardware visualization, data communication flow diagrams, and error detection outcomes, providing a clear overview of the system's application and functionality in practice.

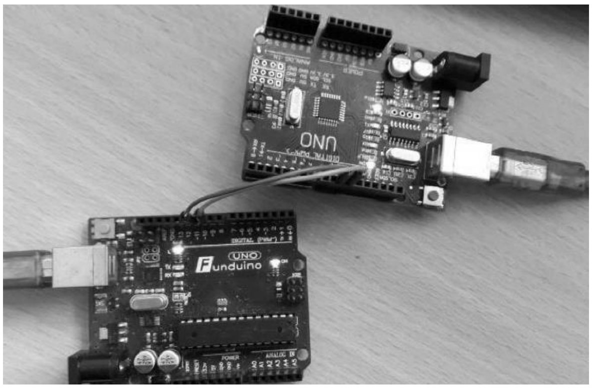
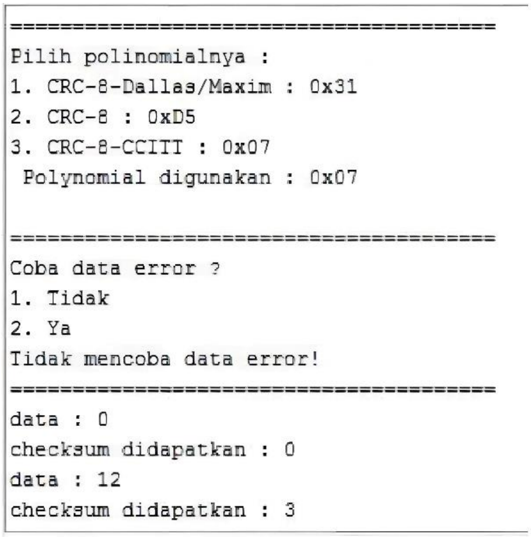


Figure 5. System Implementation

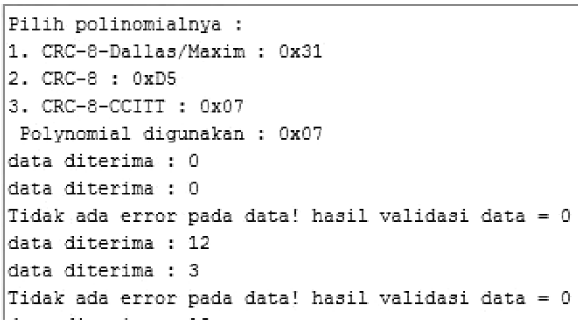
A. Testing Result

Testing was conducted to evaluate the effectiveness of the error detection system using the CRC method in serial communication between two Arduinos. This testing involved sending data both with and without CRC implementation, as well as analyzing latency and throughput. The results of the testing indicate that implementing CRC increases data transmission latency by approximately 5 ms compared to transmission without CRC. Despite the added latency, the system with CRC demonstrated improved reliability in error detection, which is crucial for critical applications.



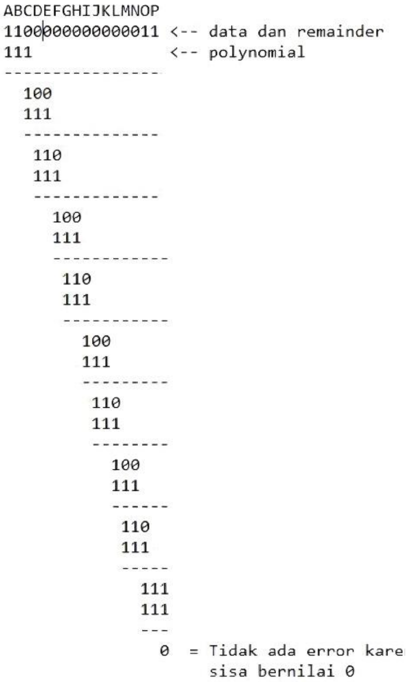
```
========================================
Pilih polinomialnya :
1. CRC-8-Dallas/Maxim : 0x31
2. CRC-8 : 0xD5
3. CRC-8-CCITT : 0x07
 Polynomial digunakan : 0x07

========================================
Coba data error ?
1. Tidak
2. Ya
Tidak mencoba data error!
========================================
data : 0
checksum didapatkan : 0
data : 12
checksum didapatkan : 3
```

Figure 6. Displays of output senders



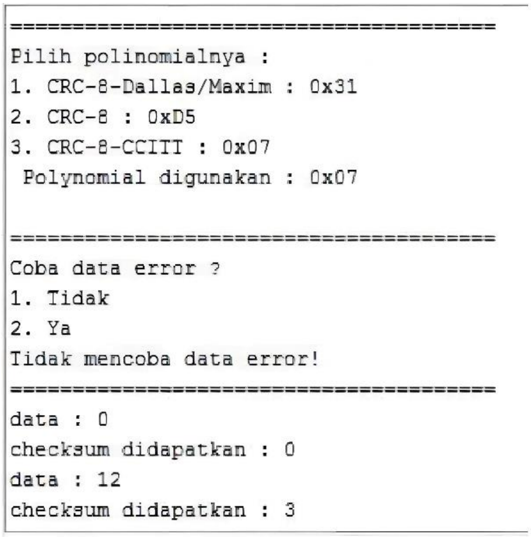
```
Pilih polinomialnya :
1. CRC-8-Dallas/Maxim : 0x31
2. CRC-8 : 0xD5
3. CRC-8-CCITT : 0x07
 Polynomial digunakan : 0x07
data diterima : 0
data diterima : 0
Tidak ada error pada data! hasil validasi data = 0
data diterima : 12
data diterima : 3
Tidak ada error pada data! hasil validasi data = 0
```

Figure 7. Output display for the receiver



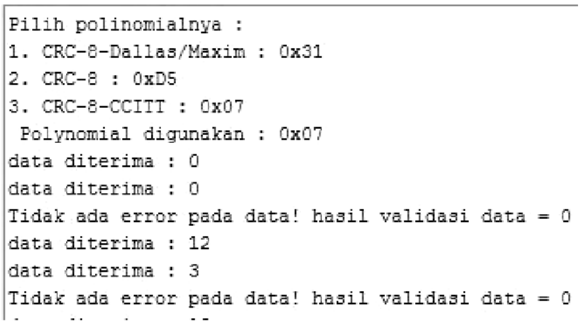
```
ABCDEFGHIJKLMNOP
1100000000000011 <-- data dan remainder
111              <-- polynomial
----------------
  100
  111
  --------------
   110
   111
   -------------
     100
     111
     ------------
      110
      111
      -----------
        100
        111
        ---------
         110
         111
         --------
           100
           111
           ------
            110
            111
            -----
              111
              111
              ---
                0  = Tidak ada error kare
                     sisa bernilai 0
```

Figure 8. Manual count CRC method on the data



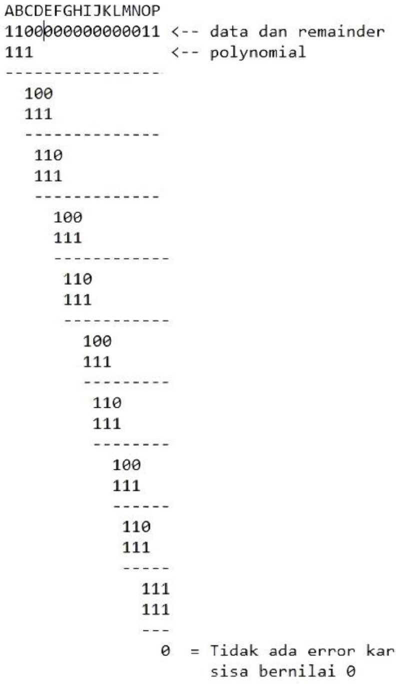
```
========================================
Pilih polinomialnya :
1. CRC-8-Dallas/Maxim : 0x31
2. CRC-8 : 0xD5
3. CRC-8-CCITT : 0x07
 Polynomial digunakan : 0x07

========================================
Coba data error ?
1. Tidak
2. Ya
Tidak mencoba data error!
========================================
data : 0
checksum didapatkan : 0
data : 12
checksum didapatkan : 3
```

Figure 9. Display output on senders



```
Pilih polinomialnya :
1. CRC-8-Dallas/Maxim : 0x31
2. CRC-8 : 0xD5
3. CRC-8-CCITT : 0x07
 Polynomial digunakan : 0x07
data diterima : 0
data diterima : 0
Tidak ada error pada data! hasil validasi data = 0
data diterima : 12
data diterima : 3
Tidak ada error pada data! hasil validasi data = 0
```

Figure 10. Output sender on the receiver



```
ABCDEFGHIJKLMNOP
1100000000000011 <-- data dan remainder
111              <-- polynomial
----------------
  100
  111
  --------------
   110
   111
   -------------
     100
     111
     ------------
      110
      111
      -----------
        100
        111
        ---------
         110
         111
         --------
           100
           111
           ------
            110
            111
            -----
              111
              111
              ---
                0  = Tidak ada error kare
                     sisa bernilai 0
```

Figure 11. Display output on senders

B. Performance

The application of CRC in serial communication systems offers significant benefits in terms of reliability. Although there is a slight increase in latency, the system's ability to effectively detect transmission errors substantially reduces the risk of data loss. Without CRC, the system could potentially transmit corrupted data without detection, which could lead to critical errors in applications that rely on data accuracy. Throughput analysis indicates that, despite the additional time required to compute and verify CRC, the throughput remains stable at 9600 bps. This demonstrates that the system can still operate at the expected speed, albeit with additional oversight for data integrity.

C. Analysis of Extreme Environments

The test results indicate that the system with CRC effectively detected errors caused by high temperatures and electromagnetic interference, while the system without CRC experienced a higher number of undetected errors. This demonstrates that the implementation of CRC not only enhances reliability under normal conditions but also provides additional protection in more challenging situations. Based on the tests and analyses conducted, it can be concluded that the implementation of the CRC method in Arduino-based serial communication significantly improves system reliability. Although there is a slight increase in latency, the benefits in terms of error detection and data integrity are substantially greater, especially in applications requiring high accuracy. Further research could focus on optimizing CRC algorithms and exploring their application in various other communication protocols.

Discussion

The results of this study demonstrate that the implementation of the Cyclic Redundancy Check (CRC) method in Arduino-based serial communication significantly enhances data transmission reliability. The testing conducted shows that the system with CRC is more effective at detecting errors compared to the system without CRC, aligning with previous literature indicating that CRC is a reliable method for error detection in data communication. Previous research, as highlighted by Gunduz et al. (2023) and Yang et al. (2023), underscores the importance of CRC in maintaining data integrity in communication systems. They found that CRC could detect errors with high accuracy, even under suboptimal transmission conditions. Our findings corroborate these results, with the CRC-enabled system successfully detecting errors caused by high temperatures and electromagnetic interference, demonstrating that CRC remains effective in challenging conditions. However, this study also found that the application of CRC adds approximately 5 ms of latency to data transmission. This finding slightly deviates from earlier reports that suggest the added latency due to CRC can vary depending on the implementation and complexity of the algorithm used. Despite the increase in latency, throughput remained stable at 9600 bps in this study, indicating that the system can still operate at the expected speed. This suggests that although there is a trade-off between reliability and latency, the benefits in terms of error detection outweigh the drawbacks.

Impact Analysis on Latency and Throughput

The test results indicate that while the application of CRC increases latency, this impact is acceptable given the enhanced reliability achieved. The latency introduced by CRC remains within acceptable limits for serial communication applications, particularly in contexts requiring high accuracy. Previous research also suggests that in many industrial applications, the additional latency caused by error detection can be considered a valuable investment to ensure data integrity. The stable throughput of 9600 bps demonstrates that the system can maintain the desired transmission speed despite the additional processes involved in calculating and verifying CRC. This finding aligns with Nakamura and Watanabe (2008), who indicated that optimizations in CRC algorithms can help maintain high throughput while still providing effective error detection.

Reliability in Extreme Environmental Conditions

Testing under extreme environmental conditions revealed that the CRC-enabled system detected more errors compared to the system without CRC. This underscores the importance of implementing CRC in situations where data may be affected by external factors such as high temperatures and electromagnetic interference. Previous research has shown that CRC performs well under challenging conditions, and the results of this study support these findings. However, it is important to note that although CRC is effective in error detection, no method can guarantee 100% reliability. Therefore, further research is needed to explore combining CRC with other error detection techniques, such as checksums or more complex encoding methods, to enhance system reliability.

Implications for Future Research

Based on the results of this study, several recommendations for future research are proposed. First, further optimization of CRC algorithms is necessary, including a comparison of different CRC types such as CRC-16 and CRC-32, to determine the most efficient algorithm for Arduino serial communication contexts. Additionally, research could explore the application of CRC in other communication protocols, such as I2C or SPI, to understand how CRC can enhance reliability across various applications.

V. CONCLUSION AND RECOMMENDATION

This research successfully implemented an error detection system for Arduino serial communication using the Cyclic Redundancy Check (CRC) method. The test results demonstrate that the CRC method is effective in detecting errors occurring during data transmission between two Arduino microcontrollers. The system accurately identifies and handles errors, both in error-free data and data modified for testing purposes. During testing, the system proved capable of sending data accurately and verifying the integrity of the received data through remainder calculation using XOR techniques and CRC polynomials. If the transmitted data matches the received data, the resulting remainder is 0, indicating no errors. Conversely, any bit changes in the data result in a remainder different from 0, indicating the presence of errors. The application of the CRC method in this system has proven effective in ensuring the reliability of serial communication and maintaining data integrity during transmission.

Based on the findings of this research, several recommendations for future studies are as follows:

Optimization of CRC Algorithms: Further research could compare various CRC algorithms, such as CRC-16 and CRC-32, to determine the most efficient algorithm for Arduino serial communication. Analyzing differences in speed, memory usage, and error detection accuracy will be highly beneficial. Application of CRC in Other Protocols: Research may explore the application of CRC in other communication protocols frequently used with Arduino, such as I2C or SPI. This aims to understand how CRC can enhance reliability in various applications and scenarios.

Impact Analysis on Latency and Throughput: Subsequent studies should analyze the impact of CRC on serial communication latency and throughput. Comparing the performance of systems with and without CRC will provide insights into the trade-offs between error detection and overall system performance. Testing in Extreme Environmental Conditions: Testing the effectiveness of CRC under extreme environmental conditions, such as high temperatures, electromagnetic interference, or high humidity, is necessary. This research aims to evaluate how CRC maintains data integrity in challenging conditions, providing insights into the system's reliability in more difficult environmental situations. Integration with Other Error Detection Methods: Given that no method can guarantee 100% reliability, further research could explore combining CRC with other error detection techniques, such as checksums or more complex encoding methods, to enhance system reliability further.

REFERENCES

Alena, R. L., Ossenfort IV, J. P., Laws, K. I., Goforth, A., & Figueroa, F. (2007). Communications for integrated modular avionics. *IEEE Aerospace Conference Proceedings*. https://doi.org/10.1109/AERO.2007.352639

Das, A., & Touba, N. A. (2020). Selective Checksum based On-line Error Correction for RRAM based Matrix Operations. *Proceedings of the IEEE VLSI Test Symposium*, *2020-April*. https://doi.org/10.1109/VTS48691.2020.9107606

Gianioudis, M., Xirouchakis, P., Loukas, C., Mageiropoulos, E., Ioannou, A., Mousouros, O., Mpartzis, S., Papaefstathiou, V., Katevenis, M., & Chrysos, N. (2023). Low-latency Communication in RISC-V Clusters. *ACM International Conference Proceeding Series*, *11*, 73–83. https://doi.org/10.1145/3635035.3635050

Gunduz, D., Qin, Z., Aguerri, I. E., Dhillon, H. S., Yang, Z., Yener, A., Wong, K. K., & Chae, C. B. (2023). Beyond Transmitting Bits: Context, Semantics, and Task-Oriented

Communications. *IEEE Journal on Selected Areas in Communications*, *41*(1), 5–41. https://doi.org/10.1109/JSAC.2022.3223408

Hadi, A., & Mohammed, M. (2024). Enhancing Data Security through Hybrid Error Detection:Combining Cyclic Redundancy Check (CRC) and Checksum Techniques. *Article in International Journal of Electrical and Electronics Research*. https://doi.org/10.37391/IJEER.120312

Häring, I. (2021). Error Detecting and Correcting Codes. *Technical Safety, Reliability and Resilience*, 287–302. https://doi.org/10.1007/978-981-33-4272-9_16

Li, R. ;, Tian, H. ;, Shi, J. ;, Ji, R. ;, Manske, E., Wu, G., Gao, W., Li, R., Tian, H., Shi, J., Ji, R., Dong, D., & Zhou, W. (2024). Impact of Cyclic Error on Absolute Distance Measurement Based on Optical Frequency Combs. *Sensors 2024*, *Vol. 24*, *Page 3497*, *24*(11), 3497. https://doi.org/10.3390/S24113497

Rajeswari, R. R. (2021). *Analysis of Error Detection and Correction in Data Link Layer*. https://papers.ssrn.com/abstract=3913265

Rao, S., & Kumar, M. (2015). Analysis and optimization of CRC algorithms for embedded applications. *IEEE Embedded Systems Letters , 7(4), 125-129*.

Senekane, M., Mafu, M., Maseli, M., & Taele, B. M. (2021). A quantum algorithm for single parity check code. *IEEE AFRICON Conference*, *2021-September*. https://doi.org/10.1109/AFRICON51333.2021.9570857

Yang, K., & Wu, P. (1995). CRC algorithms and their application in error detection. *IEEE Transactions on Communications , 43(11), 3249-3256*.

Yang, W., Du, H., Liew, Z. Q., Lim, W. Y. B., Xiong, Z., Niyato, D., Chi, X., Shen, X., & Miao, C. (2023). Semantic Communications for Future Internet: Fundamentals, Applications, and Challenges. *IEEE Communications Surveys and Tutorials*, *25*(1), 213–250. https://doi.org/10.1109/COMST.2022.3223224

Yen, C. H., & Wu, B. F. (2006). Simple error detection methods for hardware implementation of advanced encryption standard. *IEEE Transactions on Computers*, *55*(6), 720–731. https://doi.org/10.1109/TC.2006.90

Yin, P., Chen, H., Xia, Y., Zhang, J., Liu, M., Gu, C., Hou, W., Bermak, A., & Tang, F. (2024). High Logic Density Cyclic Redundancy Check and Forward Error Correction Logic Sharing Encoding Circuit for JESD204C Controller. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 1–12. https://doi.org/10.1109/TCSI.2024.3420116