ISSN: 2834-4928

Research Article

Journal of Electrical Electronics Engineering

AB485: A Simplified Data Transfer Protocol for Process Control and Automation

Abdalrhman Alquaary[1]* and Hasan Bayhan[2]

[1]*ALQUANIX Sakarya, Turkiye.*

[2]*Bayhanelektroteknik Sakarya, Turkiye.*

***Corresponding Author**
Abdalrhman Alquaary, ALQUANIX Sakarya, Turkiye.

Submitted: 2024, Mar 13; **Accepted:** 2024, Apr 16; **Published:** 2024, Jun 18

Citation: Alquaary, A., Bayhan, H. (2024). AB485: A Simplified Data Transfer Protocol for Process Control and Automation. *J Electrical Electron Eng, 3*(3), 01-07.

Abstract

In industrial environments, achieving effective communication between electronic devices poses a multifaceted challenge that hinges on the seamless integration of both hardware and software protocols. This paper introduces AB485, a purpose-built data transfer protocol designed primarily for microcontroller-based systems, offering a streamlined solution to the complexities of software protocol implementation. We delve into AB485's design principles and features, underscoring its minimal hardware and software requirements. Notably, AB485 stands out for its ease of implementation, enabling engineers or system integrators to seamlessly integrate it into their projects. We also explore potential applications where AB485's efficiency and low overhead make substantial contributions to ensuring reliable data transmission. By presenting AB485, our research contributes to the field of data transfer protocols, offering an accessible and efficient solution for resource-constrained applications, while addressing the intricate interplay of hardware and software in data communication.

Keywords: AB485, Serial Communications, Data Transfer, Software Protocol, RS-485 Interface

1. Introduction

As civilization advanced, communication became the cornerstone of human progress. Similarly, in the world of industries and devices, effective communication is the linchpin for building complex systems and powering modern technologies. While hardware protocols serve as reliable means of communication, they often come with limitations, including reduced flexibility, higher costs, and complex implementation. In contrast, software protocols offer adaptability, customization, and compatibility across diverse hardware platforms, making them the preferred choice in many applications.

Amidst the myriad of complex protocols, AB485 stands out as a streamlined and efficient solution. It simplifies the art of device communication by employing just two primary functions: one for reading and the other for writing. What makes AB485 particularly noteworthy is its minimum data transfer unit – a single byte. This choice stems from the increased storage capacity of modern EEPROMs, which enables more efficient and versatile data exchange.

AB485 is strategically built upon the established framework of RTU in serial communications, ensuring seamless operation across diverse hardware protocols. While AB485 maintains compatibility with various communication mediums, our preference lies in its deployment over the RS-485 interface. RS-485, known for its robust features, plays a pivotal role in enhancing the effectiveness of AB485. The RS-485 interface offers advantages such as differential signaling, enabling reliable communication over long distances by minimizing signal degradation and susceptibility to electromagnetic interference. Additionally, its multi-point capability allows multiple devices to communicate on the same bus, enhancing the scalability of the network. These features make RS-485 an ideal choice, reinforcing the decision to leverage it in conjunction with the AB485 protocol. In this paper, we will delve into the intricacies of AB485, exploring its design, implementation, and the symbiotic relationship it shares with the robust RS-485 interface, thereby fostering efficient and straightforward device-to-device communication.

1.1 Literature Overview

Serial communication is a cornerstone in industrial automation and control systems. This is exemplified in the use of TMS320C6748DSP for DSP-PC communication, highlighting its precision and real-time interfacing capabilities [1]. Similarly, the study by Auccahuasia et al. underscores the relevance of RS-232 in industrial settings, particularly through Labview software for effective data exchange [2]. The importance of low-power transceivers, as detailed in the MAX481/MAX483/MAX485/MAX487–MAX491/MAX1487 documentation, aligns with the need for efficient data transmission in industrial control [3].

The versatility of the Modbus protocol in various configurations, as analyzed by Gaitan and Zagan [4], and its implementation on single-chip microcomputers by Su Ying et al., point towards its

adaptability and efficiency [5]. Tools for response time analysis in Modbus networks, as developed by Künzel and colleagues, further enhance the reliability of serial communications [6]. The advancements in Modbus for high-speed data transmission by Yao Yuanyuan et al. indicate evolving needs in big data scenarios [7].

Xu, et al's research on C8051F020 microcontrollers demonstrates the practicality and reliability of single-chip systems in serial communication [8]. Xunwen et al.'s work on RS-485 pseudo-full-duplex communication highlights its applications in remote control and aerospace engineering [9]. The development of USB to RS485 converters caters to the need for seamless integration of RS485 devices with modern computer systems [10].

Huang's exploration of STM32 microcontrollers in serial communication showcases their utility in short-distance networks while Cao, Chen, and Li's study on UART emphasizes its wide-ranging applications despite certain limitations [12].

Expanding on the landscape of electronic device control, the prevalence of hardware serial communication protocols, including RS-232, RS485, I2C, and SPI, emerges as a crucial foundation for seamless data transfer between personal computers or mobile devices and peripheral devices [13]. Within this context, the integration of the PIC16F876A in the discussed system accentuates the pivotal role of serial communication, operating with a secure and reliable set of rules, in efficiently controlling electronic devices. This perspective complements existing research and further underscores the paramount importance of hardware protocols in shaping the landscape of contemporary electronic device control.

In light of the diverse studies in serial communication, the introduction of AB485 addresses a critical need in industrial environments where effective communication between electronic devices necessitates a harmonious integration of both hardware and software protocols. AB485, being purpose-built for microcontroller-based systems, stands out as a streamlined solution, minimizing the complexities associated with software protocol implementation. Its design principles and features emphasize efficiency with minimal hardware and software requirements, offering engineers and system integrators an accessible and user-friendly option for seamless project integration. As we delve into the potential applications of AB485, its efficiency and low overhead become paramount, contributing significantly to reliable data transmission in resource-constrained environments. In essence, the emergence of AB485 reflects the ongoing demand for tailored software protocols that not only simplify implementation but also address the intricate interplay of hardware and software in the dynamic landscape of data communication.

2. Methodology

In our research endeavor, we present the AB485 software serial protocol, a communication solution tailored to the needs of development and research entities, as well as individual users. AB485 offers a user-friendly and straightforward approach, making it particularly suitable for these audiences. The protocol's steps are intentionally designed to be concise, comprehensive, and devoid of unnecessary complexity.

AB485, with its focus on data read and write operations between master and slave devices, stands out for its simplicity and efficiency. It is positioned as an accessible and practical alternative for development and research companies, as well as individual users.

For connectivity, AB485 operates seamlessly above RS232, providing a reliable communication channel for applications requiring shorter distances. Additionally, AB485 is designed to run efficiently above RS485, offering enhanced performance over longer distances. Supporting the connection of a master device to up to 255 slave devices simultaneously, AB485 demonstrates its scalability and versatility. Furthermore, it accommodates data transfers ranging from 1 to 255 bytes per frame, thus catering to a wide array of data transfer requirements.

In AB485 software serial protocol, a deliberate 10-bit response delay has been strategically introduced as a crucial element in ensuring precise and reliable communication. This delay serves multiple key purposes within the communication framework, as illustrated in Figures 2 and 4, particularly in the "Write Answer" and "Read Answer" frames. Firstly, it plays a vital role in synchronizing the timing between transmitting and receiving devices, mitigating timing misalignments and ensuring accurate reception. Secondly, the calibrated delay contributes to effective buffering and flow control mechanisms, providing the receiving device with sufficient time to process and clear its buffer before the next data set arrives, thereby preventing buffer overflow and maintaining communication stability. Calculating the response delay will depend on the baud rate speed. So it will be:

$$10 * 1/\text{baudrate} \quad (1)$$

Within the AB485 protocol, we encounter four distinct scenarios that revolve around data frames. These scenarios encompass a thorough exploration of both transmission directions: from the master to the slave and from the slave to the master. To facilitate a comprehensive understanding, each scenario is discussed in detail, and dedicated visualizations are provided. It's worth noting that while these visualizations help illustrate the protocol's operation, Table 1 exclusively features the variables used within the data frames. This distinction serves to clarify the specific constants employed in the protocol while presenting a visual representation of each scenario for enhanced comprehension.

Table 1: Section Names in AB485 Data Frame

N	Section Name	Meaning
1	Enable	(0xAA) simple acknowledgment byte to confirm readiness for communication.
2	Slave ID	ID of the slave device. 0 < X < 255
3	Function Code	"w" (0x77) for writing data and "r" (0x72) for reading data.
4	Byte Count	The number of the bytes that will be read or written. 0 < X < 255
5	Address HH	The high and low sides of the address.
6	Address LL	0 < X < 65535
7	EOT	(0x55) is the end of the tail.
8	BCC	The control code that we calculate

To calculate the Block Check Character (BCC), you need to sum all the bytes between the "enable" and "EOT" bytes, byte by byte. After this summation, the least significant byte is extracted and used as the BCC for the data frame. The following equation provides a mathematical representation of this operation (2). In this equation, "B[k]" represents the value of the byte at position "k" within the data frame.

$$BCC = Least\ Siginificant\ Byte(\sum_{k}^{n}(B[k])) \quad for\ B = Bytes\ array, and\ k = 0\ to\ n-1 \tag{2}$$

• Read Request

The Read Request Format, designed for requesting data reads, encompasses critical components. The "Enable" byte (1 Byte) signals the device's readiness for communication, initiating the data exchange process. The "Slave ID" byte (1 Byte) identifies the target device, ensuring that the request is directed to the intended recipient, with Slave IDs ranging from 1 to 254. The "Function Code" byte (1 Byte) specifies the type of operation to be executed, typically indicating a read operation. The "Byte Count" byte (1 Byte) provides information about the number of bytes to be included in the response, aiding in understanding the payload size. The "Address HH" (2 Bytes) represents the high-order section of the memory address from which data is to be read, while "Address LL" (2 Bytes) corresponds to the low-order section of the memory address. The "EOT" byte (1 Byte) marks the conclusion of the request, and the "BCC" byte (1 Byte) is included for error checking and data integrity validation Figure 1.

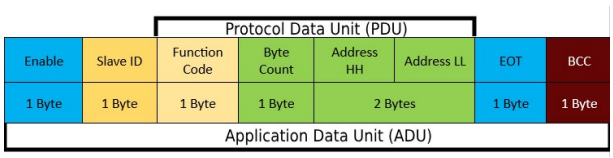

Figure 1: Read Request Data Frame Structure

• Read Answer

The Read Answer Format, used for responding to read requests, is structured with essential elements. The "Enable" byte (1 Byte) acknowledges the request and indicates readiness for further data transmission or reception. The "Slave ID" byte (1 Byte) identifies the responding device, confirming the recipient of the response, with Slave IDs ranging from 1 to 254. The "Function Code" byte (1 Byte) specifies the type of operation performed, typically confirming a read operation. The "Byte Count" byte (1 Byte) specifies the number of data bytes included in the response, aiding in understanding the payload size. The "Bytes" section (N Bytes) contains the data read from the specified memory location. The "EOT" byte (1 Byte) signifies the completion of the response, while the "BCC" byte (1 Byte) is used for error checking, ensuring data integrity Figure 2.

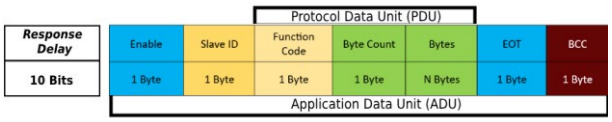

Figure 2: Read Answer Data Frame Structure

• Write Request

The Write Request Format consists of several crucial fields. The “Enable” byte (1 Byte) indicates the device’s readiness for data transmission or reception. The “Slave ID” byte (1 Byte) plays a critical role in identifying the specific target device within the network, with Slave Ids ranging from 1 to 254. The “Function Code” byte (1 Byte) specifies the type of operation to be performed, often indicating a write operation. The “Byte Count” byte (1 Byte) provides essential information about the number of bytes that will follow in the request, helping determine the request’s payload size. The “Address HH” (2 Bytes) represents the high-order part of the memory address where data is to be written, while “Address LL” (2 Bytes) corresponds to the low-order portion of the memory address. The “Bytes” section (N Bytes) contains the actual data to be written to the specified memory location. The “EOT” byte (1 Byte) marks the conclusion of the data transfer process. Finally, the “BCC” byte (1 Byte) is the Block Check Character, calculated for error checking and ensuring data integrity Figure 3.

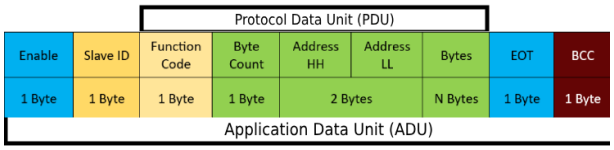

Figure 3: Read Answer Data Frame Structure

• Write Answer

The Write Answer Format, used for acknowledging write

requests, comprises essential fields. The “Enable” byte (1 Byte) acknowledges the request and signals readiness for further data transmission or reception. The “Slave ID” byte (1 Byte) identifies the responding device, confirming the recipient of the acknowledgment, with Slave Ids ranging from 1 to 254. The “Function Code” byte (1 Byte) specifies the type of operation performed, typically confirming a write operation. The “EOT” byte (1 Byte) signifies the completion of the response. Lastly, the “BCC” byte (1 Byte) serves as the Block Check Character for error checking and data integrity validation Figure 4.

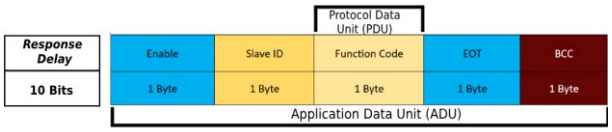

Figure 4: Read Answer Data Frame Structure

3. Experimental Results

To assess the practical effectiveness of the AB485 protocol, our research incorporates a comprehensive experimental approach. We will execute the protocol using a computer as the master device and a PIC chip as the slave device, both in computer-based simulations and oscilloscope-based hardware tests. These dual methodologies ensure a thorough evaluation of the protocol’s performance and its compatibility with real-world applications. The following examples showcase the outcomes of these experiments, shedding light on the protocol’s capabilities and reliability in different settings.

• Hardware

We constructed a slave device in the BayhanElektroteknik Lab using the DSPIC33MC64 as the main microcontroller Figure 5. To store data, we integrated an EEPROM chip. For data transmission, we employed the MAX485 chip to send data from the slave device. To receive data on the computer’s COM port, we utilized an RS485 to TTL converter. Additionally, we developed a Python program capable of interpreting and retrieving data using the AB485 protocol.

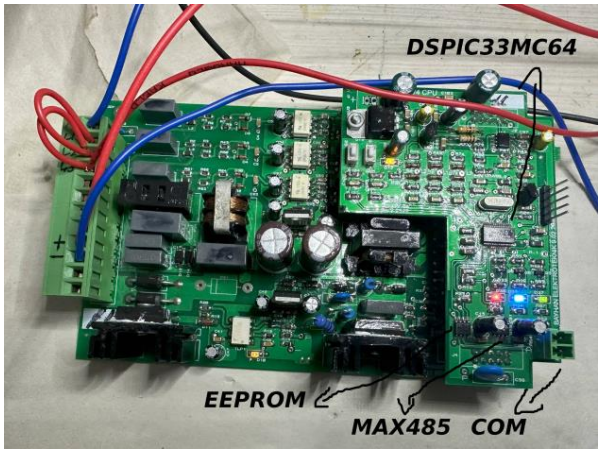

Figure 5: BETCOM Device Connected to Controlling System

• Software

We configured the computer as the master device and designed a user-friendly GUI software using PyQt5 Figure 6. This software facilitated the transmission of requests and retrieval of responses from the slave devices. After thorough testing, we achieved a high level of efficiency in serial communication, ensuring seamless and reliable data exchange between the master and slave devices. It's important to note that this program was developed in our lab and is not currently open source. However, we utilized it extensively to rigorously test the communication between the chip and the PC. Detailed examples and outcomes of these tests will be elucidated next, shedding light on the practical applications and performance of our developed software.

Moreover, the use of software serial protocols such as AB485 provides developers with the ability to create programs that simulate processes inside the chips and memories more easily and efficiently. This capability not only enhances the development and testing phases but also contributes to a more streamlined and effective exploration of the intricate functionalities within electronic devices

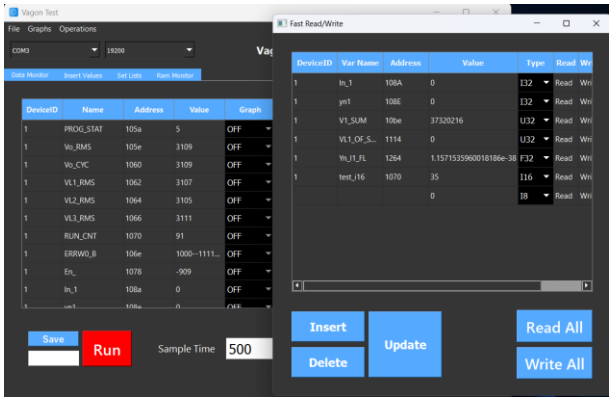

Figure 6: AB485 Communication Program

In these visual examples, we’ve illustrated the AB485 protocol’s frames in distinct scenarios, allowing us to gain a deeper understanding of how the protocol operates. By interpreting these signals, we can discern the intricacies of the protocol’s functionality in various situations. These visual representations serve as valuable tools for enhancing our comprehension of the AB485 protocol and its real-world applications.

The visual representation includes four distinct frames, each serving a specific purpose within the protocol. The “Read Request” and “Read Answer” frames exemplify the request-response cycle associated with read operations. In the “Read Request” frame, the master initiates communication, indicating its intent to obtain data from a specific address in the slave device. Conversely, the “Read Answer” frame serves as the response from the slave to the master, providing essential information, including the data received.

Similarly, the “Write Request” and "Write Answer" frames symbolize the exchange for write operations. The "Write Request" frame signifies the master's transmission of data to the slave, encapsulating both the address and the data to be written. The corresponding "Write Answer" frame indicates the acknowledgment or response from the slave device, confirming the successful execution of the write operation.

To enhance the clarity of the visual representation, annotations have been thoughtfully incorporated into the graphs. "DF" (Data Frame) and "Clock" annotations have been strategically placed to distinguish and clarify the components of the signals. Additionally, the titles of each graph provide further contextual information, including addresses and details regarding the data received or transmitted.

Moreover, it is important to note that data transmission in the AB485 protocol occurs over RS485, a communication standard that excels in long-distance communication. This feature extends the protocol's utility to scenarios where data needs to be reliably transmitted over extended distances. It is particularly well-suited for applications that require secure and error-resistant communication over substantial distances, making it an ideal choice for industries where the need for long-distance communication is paramount. Industries such as manufacturing, industrial automation, agriculture, and utilities can greatly benefit from the AB485 protocol's ability to ensure dependable data transfer over extended network lengths.

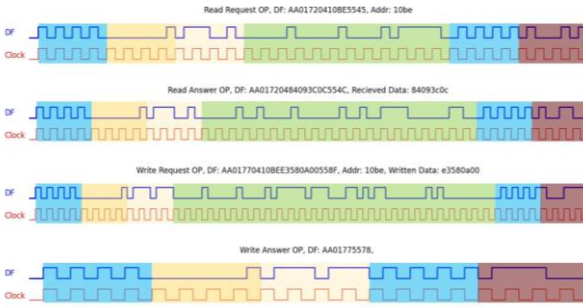

Figure 7: AB485 Communication Program

• Read Request Data Frame

In Figure 8., we present a captured data frame representing the "Read Request" within our data communication system. This figure includes a visual depiction of the data frame itself, showcasing the specific content and structure of the request. The oscilloscope-captured data frames were executed using address other than 10BE.

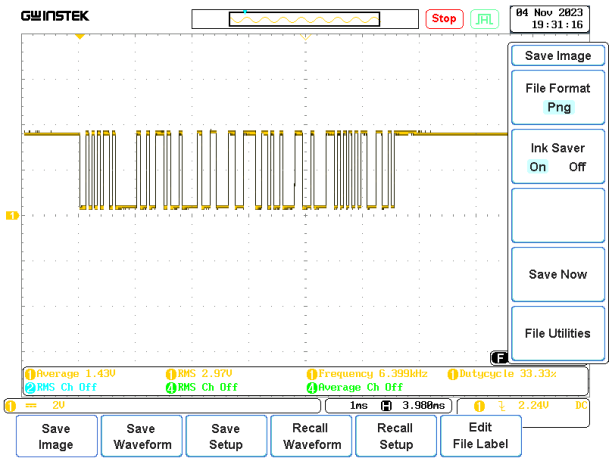

Figure 8: Read Request Data Frame Signals

• Read Answer Data Frame

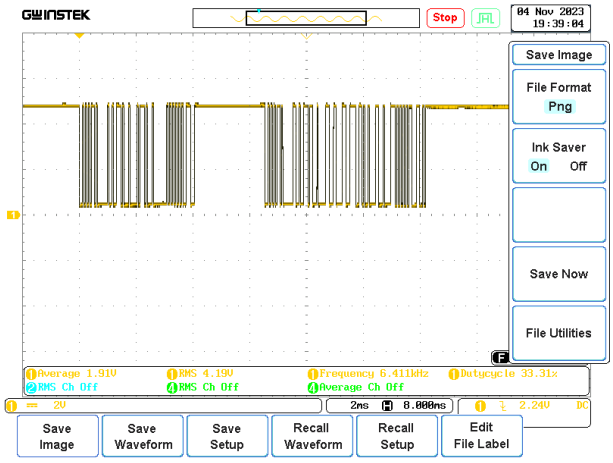

Figure 9: Read Answer Data Frame Signals

Figure 9. illustrates the "Read Answer" data frame captured from our oscilloscope. This figure represents the response data frame generated in response to a "Read Request." The automatic transmission feature of the MAX485 chip is evident once again, displaying both the request and response data frames due to the chip's direct forwarding of transmitted data to the receive line. These figures collectively offer a comprehensive view of the bidirectional data communication process in our system, highlighting the operation of the MAX485 chip and the interactions between request and response data frames.

• Write Request Data Frame

Figure 10. illustrates the "Write Request" data frame captured from our oscilloscope. This figure, similar to Figure 8., provides a visual representation of the requested data frame.

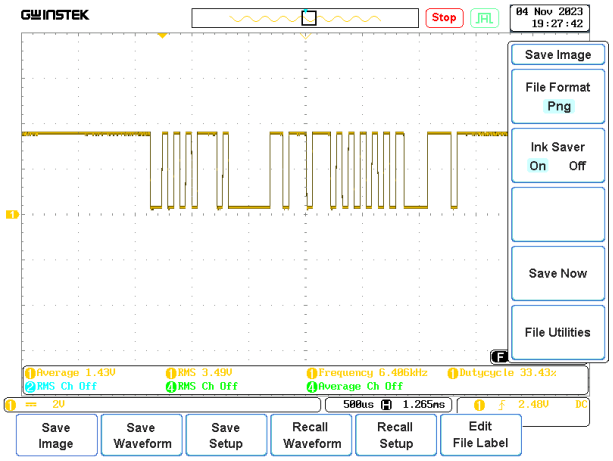

Figure 10: Write Request Data Frame Signals

• **Write Answer Data Frame**

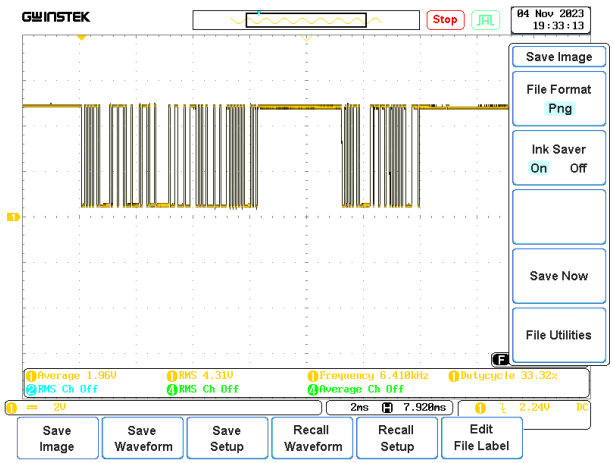

Figure 11: Write Answer Data Frame Signals

In Figure 11, we present a captured data frame representing the "Write Answer" within our data communication system. This figure showcases the response data frame generated in response to a "Write Request." As in the previous figures, the simultaneous presence of the request and response data frames is a result of the MAX485 chip's automatic transmission mechanism.

Modbus RTU remains a widely used protocol in various industries, offering reliability in many applications [13]. However, our protocol stands out as more suitable for specific implementations, as highlighted in Table 2. While both protocols have additional properties, our focus centers on the key distinctions that make our protocol more appropriate in certain aspects.

Table 2: Comparison between AB485 and Modbus RTU/ASCII

N	Property	AB485	Modbus RTU/Ascii
1	Functions	2 Function	More than 10
2	Least Data Unit	1 Byte	1 bit
3	Error Check	BCC – 1 Byte	CRC – 2 Bytes
4	Response Delay	10 bits	3.5 Bytes
5	Complexity	Easier to implement	Harder to implement

• **Functions**

AB485's simplicity in offering two main functions is a positive aspect, streamlining applications where a straightforward approach is preferred and reducing unnecessary overhead. Conversely, Modbus RTU/ASCII's array of more than 10 functions provides versatility but may introduce complexity, particularly in scenarios favoring simplicity.

• **Least Data Unit**

AB485's use of a 1-byte data unit strikes a balance, offering a granularity suitable for various applications without unnecessary complexity. In contrast, Modbus RTU/ASCII operates at the bit level, potentially introducing challenges in data handling and processing due to its finer granularity.

• **Error Check**

AB485's utilization of a 1-byte Block Check Character (BCC) for error checking is resource-efficient and effective for ensuring data integrity. On the other hand, Modbus RTU/ASCII's more robust 2-byte Cyclic Redundancy Check (CRC) enhances error detection but adds overhead to the communication process.

• **Response Delay**

AB485's relatively low response delay of 10 bits contributes to faster communication between devices. In contrast, Modbus RTU/ASCII incurs a higher response delay of 3.5 bytes, potentially impacting real-time responsiveness, a critical factor in certain applications.

• **Complexity**

AB485's ease of implementation is a positive aspect, offering a user-friendly approach that accelerates development and reduces the likelihood of errors. In contrast, Modbus RTU/ASCII, while versatile, comes with increased complexity, which may pose challenges in projects where simplicity is a priority.

The choice between AB485 and Modbus RTU/ASCII hinges on specific application requirements. AB485's simplicity and efficiency make it favorable for streamlined communication with fewer functions, while Modbus RTU/ASCII's versatility may be more suitable for complex systems requiring a broader range of functionalities despite the associated increase in complexity.

4. Conclusion

The introduction of AB485 signifies a substantial leap in addressing the intricate challenges of effective communication in industrial environments, emphasizing the integration of both hardware and software protocols. Designed specifically for microcontroller-based systems, AB485 stands out for its minimal hardware and software requirements, offering an efficient and accessible solution for engineers and system integrators. Its ease of implementation allows seamless integration into diverse projects, showcasing adaptability across applications. Notably, AB485's efficiency and low overhead contribute significantly to reliable data transmission, making it a valuable asset in resource-constrained environments. An additional noteworthy application of AB485 lies in its compatibility with chips that use internal/external EEPROM, presenting a versatile alternative to traditional Programmable Logic Controllers (PLCs) in industrial

settings. By presenting AB485, our research makes a noteworthy contribution to the field of data transfer protocols, offering a streamlined solution that balances accessibility and efficiency, particularly in the complex interplay of hardware and software in data communication within industrial settings.

Future Studies

We aspire to enhance the capabilities of our protocol by doubling the byte count to 2 bytes, thereby extending the potential data range to 65,536. This modification aims to elevate the protocol's versatility and accommodate a broader spectrum of data-intensive applications, ensuring adaptability to evolving industry needs. Additionally, we plan to enhance the security of data transfer by expanding the BCC to 2 bytes, reinforcing the robustness and reliability of the communication protocol. Furthermore, our future endeavors include the development of an IP/TCP version, necessitating adjustments to certain aspects of the data frame to align with the requirements of this networking protocol. These advancements represent our commitment to continual improvement and innovation in the realm of data transfer protocols.

Acknowledgments

We extend my heartfelt gratitude to BayhanElektroteknik lab for their exceptional craftsmanship in crafting all the hardware devices used in this project. Their expertise and dedication greatly contributed to the success of the hardware implementation. Additionally, I would like to acknowledge ALQUANIX lab for their support in developing the master software, which played a pivotal role in the overall functionality of the project. It's important to note that this project was accomplished without external funding, highlighting the commitment and collaborative spirit of all involved parties.

Refernces

1. Wu, X., Mei, Y. S., Yu, J. Q., Yu, T. P., & Li, J. X. (2013). A Design of UART Serial Communication between the TMS320C6748 DSP and PC. *Applied Mechanics and Materials, 380,* 3657-3660.
2. Auccahuasia, W., Urbanob, K., Ovallec, C., Felipped, M., Pachecoe, O., Bejarf, M., & Ruizj, M. (2021). Application of serial communication in industrial automation processes.
3. Maxim Integrated. (2023). MAX481/MAX483/MAX485/MAX487–MAX491/MAX1487 Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers.
4. Găitan, V. G., & Zagan, I. (2022). Modbus protocol performance analysis in a variable configuration of the physical Fieldbus architecture. *IEEE Access, 10*, 123942-123955.
5. Ying, S., Jin, H., Wu, X., Chunjie, J., Wei, G., & Ping, W. (2018). Research on Modbus Bus Protocol Implementation Technology Based on Single Chip Microcomputer. In *2018 3rd International Conference on Information Systems Engineering (ICISE)* (pp. 127-131). IEEE.
6. Künzel, G., Ribeiro, M. A. C., & Pereira, C. E. (2014, July). A tool for response time and schedulability analysis in modbus serial communications. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)* (pp. 446-451). IEEE.
7. Yuanyuan, Y., & Meng, C. (2021). The design of adaptive communication frame supporting high-speed transmission based on ModBus protocol. *Procedia Computer Science, 183,* 551-556.
8. Xu, L. I. N. G., Chen, Z. H. E. N., & Zhang, S. L. (2010). Research of serial communication system based on C8051F020 singlechip. In *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)* (Vol. 8, pp. V8-404). IEEE.
9. S. Xunwen, W. Shaoping, Z. Dongmei, Z. Qisherr. (2010). "RS-485 Serial Port Pseudo-full-duplex Communication Research and Application." IEEE Xplore.
10. Rishabh. (2023). USB to RS485 Converter.
11. Jian, H. (2017). Research of Serial Communication Based on STM32. In *7th International Conference on Education, Management, Information and Computer Science (ICEMC 2017)* (pp. 191-193). Atlantis Press.
12. Cao, L., Chen, J., & Li, J. (2023). Working principle and application analysis of UART. In *2023 IEEE 2nd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA)* (pp. 255-259). IEEE.
13. ModbusTool. (2023). Modbus.
14. Pwint, H. N. Y., Kywe, T., & Aung, T. T. E. (2019). Pc And Pic Based Electronic Devices Controller Using Serial Communication. *International Journal of All Research Writings, 2*(3), 129-133.