Implementation of CAN Protocol

Pooja Patel
Department of Electronics and Communication Engineering
Institute of Technology, Nirma University
Ahmedabad, India
22bec094@nirmauni.ac.in

Mudra Shah
Department of Electronics and Communication Engineering
Institute of Technology, Nirma University
Ahmedabad, India
22bec075@nirmauni.ac.in

***Abstract*—The Controller Area Network (CAN) protocol is widely used in embedded systems for robust and reliable communication. This paper presents an in-depth implementation of the CAN protocol using two Arduino Uno boards. The system allows for serial data transmission from one Arduino, which is received and displayed on an LCD screen at the receiver end. The hardware setup includes MCP2515 CAN controllers and MCP2551 transceivers, which facilitate CAN communication. The system is designed to ensure reliable message transmission with minimal delay while maintaining data integrity.**

***Index Terms*—CAN Protocol, Arduino Uno, MCP2515, Serial Communication, Embedded Systems**

I. Introduction

The CAN protocol has been a strong, popular communication standard, especially in embedded systems as well as automotive systems. The CAN protocol was developed by Robert Bosch GmbH during the 1980s and was created to provide a reliable, real-time data exchange among electronic control units (ECUs) in applications where efficiency, fault tolerance, and minimized latency are critical. This assignment is a special one and is centered on the deployment of the CAN protocol, delving into its architecture, operation, and real-world application in a given use case. The relevance of CAN comes from its capability to enable hassle-free communication in distributed systems, which makes it a foundation for contemporary automotive, industrial automation, and IoT infrastructures.

Compared to other communication protocols, CAN has distinctive characteristics. While the Universal Asynchronous Receiver/Transmitter (UART) is a point-to-point protocol that lacks scalability, CAN facilitates a bus-based, multi-master topology enabling several nodes to exchange information using a single shared medium. In contrast to Ethernet, with high bandwidth but at increased complexity and cost, CAN focuses on simplicity, determinism, and cost-effectiveness but at reduced data rates (normally up to 1 Mbps). Likewise, although protocols such as SPI and I2C work for short-distance, intra-device communication, they do not possess the noise immunity and range abilities of CAN, which can run hundreds of meters on the correct termination. Additionally, CAN's inherent error detection features such as CRC and bit stuffing give it a degree of reliability over many lightweight protocols and thus make it more suited for applications where safety is an absolute priority. This report seeks to explore the actual application of CAN, its merits, and challenge issues faced, basing conclusions on its relative advantages over competing communication standards.

II. Literature Review

In embedded systems, the Controller Area Network (CAN) protocol is widely used, especially in applications related to industrial automation and automobiles. It makes it possible for several microcontrollers to communicate reliably with one another, allowing for effective real-time data exchange.

Nahas et al. (2009) studied resource-constrained embedded systems using CAN and proposed methods to minimize message-length variations caused by bit stuffing. They evaluated software-based solutions to improve timing predictability, highlighting the impact of message variations on real-time applications[4]. This research is relevant to our implementation, as our system also relies on predictable message transmission to ensure reliable data display on an LCD.

Similarly, Bozdal et al. (2018) surveyed CAN bus vulnerabilities, analyzing attacks and potential solutions. Their research emphasized that although CAN is robust against electrical noise, it lacks encryption and authentication features[6]. While our implementation does not focus on security enhancements, understanding these vulnerabilities is crucial for future improvements.

Lotto et al. (2024) provided an extensive survey of CAN authentication protocols, identifying security gaps and the need for cryptographic authentication methods[5]. The lack of authentication in CAN makes it susceptible to unauthorized data injection, which could compromise system reliability. Future iterations of our implementation could integrate security mechanisms to prevent such attacks.

III. Drawbacks of currently available technology

While the CAN protocol is highly regarded for its reliability and efficiency in embedded systems, it is not without its limitations. These drawbacks stem from its design constraints and the evolving demands of modern applications:

1. Limited Bandwidth: The default CAN protocol has a maximum data rate of 1 Mbps (CAN 2.0), which is much less than contemporary options such as Ethernet (up to 1 Gbps or higher) or FlexRay (up to 10 Mbps). This limits its application

in high-speed data transfer applications like sophisticated multimedia systems or extensive sensor networks.

2. Payload Size Limitation: CAN frames support an 8-byte (64-bit) maximum payload per message in the traditional CAN specification. Such a small data size requires fragmentation for larger data transfers, causing higher latency and complexity in firmware updates or transmission of high-resolution sensor data.

3. Scalability Issues: While CAN can handle a multi-node bus topology, its performance under heavy bus loads suffers as more nodes are added because of bus arbitration and the possibility of collisions. The protocol's priority arbitration (based on message IDs) can also result in lower-priority messages taking an infinite wait under heavy bus loading, which is called "priority inversion.

4. Obsolescence with Emerging Needs: The emergence of technologies such as autonomous cars and Industry 4.0 has brought into focus the limitations of CAN in contrast to newer standards such as CAN FD (Flexible Data-rate, an extension of CAN) or Time-Triggered Ethernet. CAN FD solves some problems (e.g., increased data rates and increased payloads), but it is not yet widely adopted, and existing CAN systems are still limited.

5. No Native Support for Time Synchronization: In contrast to protocols like FlexRay or Time-Sensitive Networking (TSN), CAN doesn't have support for exact time synchronization among nodes, which is a disadvantage when coordinated action or strict timing is needed in certain applications.

IV. Components Used

The hardware setup includes various components essential for CAN communication:

A. Arduino Uno

Arduino Uno is a microcontroller board based on the ATmega328P processor. It provides an easy-to-use development platform with multiple I/O pins, SPI communication support, and serial interface capabilities. The Arduino Uno serves as the processing unit for both the transmitter and receiver nodes.

B. MCP2515 CAN Controller

The MCP2515 supports the CAN V2.0B technical standard, with communication rate 1Mb/s. The MCP2515 is a CAN standalone controller that interfaces with the Arduino via SPI. It manages CAN message transmission and reception, providing filtering mechanisms to handle multiple message types. It has 120ohm terminal resistance. The MCP2515 has an integrated 8MHz crystal oscillator for stable communication and is configured to operate at a baud rate of 250 kbps. The MCP2515 uses high-speed CAN transceiver TJA1050. The module integrates seamlessly with the Arduino and allows easy expansion of CAN networks.

C. 16x2 LCD Display

The 16x2 LCD module is used at the receiver end to display the transmitted messages in real time. Unlike I2C-based LCDs, this display operates using a parallel communication interface, requiring multiple digital I/O pins from the Arduino. The LCD is interfaced using RS, EN, D4, D5, D6, and D7 pins, which are connected directly to the Arduino for data transmission. While this setup requires more wiring than an I2C-based display, it offers better control over display timing and refresh rates.

D. Power Supply and Connections

Each Arduino is powered via a 5V DC supply, while the MCP2515 module derives power from the Arduino's onboard voltage regulator. Proper grounding is maintained across all components to minimize interference and ensure reliable communication.

V. Block Diagram

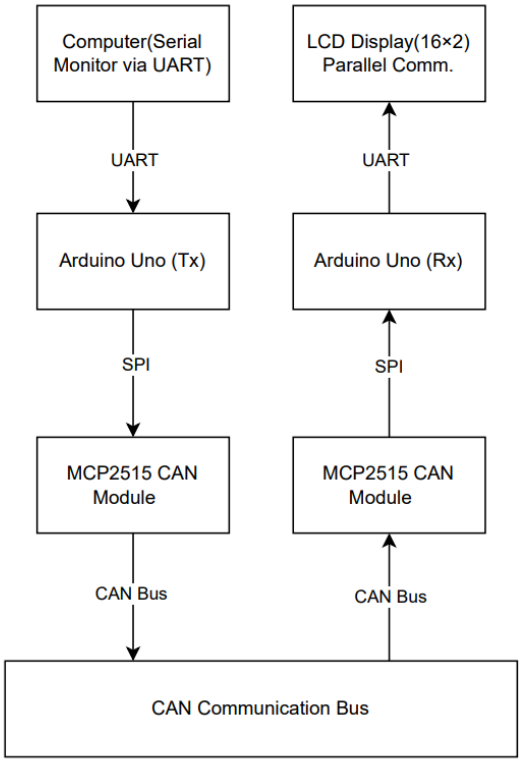

Fig. 1. Block Diagram of CAN Protocol Implementation

VI. Methodology

In our implementation of the CAN Protocol on two Arduino Uno boards, we utilize several communication protocols to ensure data transmission efficiently. At the transmitting end, we utilize UART communication between the computer and the Arduino Uno to allow the Arduino to get data input through the serial monitor. The Arduino Uno and the MCP2515 CAN controller module exchange information through the

SPI protocol to provide transparent data transfer from the microcontroller to the CAN transceiver and vice versa. The actual transmission of data from one MCP2515 module to the other happens through the CAN bus, which provides effective and reliable data transfer. Equally, in the receiving direction, SPI communication takes place between the Arduino Uno and its MCP2515 module to collect the CAN data. In order to supply power to the receiver-side Arduino Uno, we use UART communication, which supplies the power via the serial interface. We also include an LCD 16×2 display on the receiver side to display the received data. The LCD is interfaced with parallel communication to the Arduino Uno so that there is direct control over the display for optimal data presentation.

VII. Results in terms of Hardware Implementation

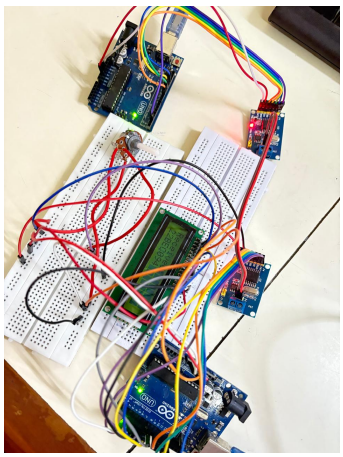

Fig. 2. CAN protocol implementation on Hardware

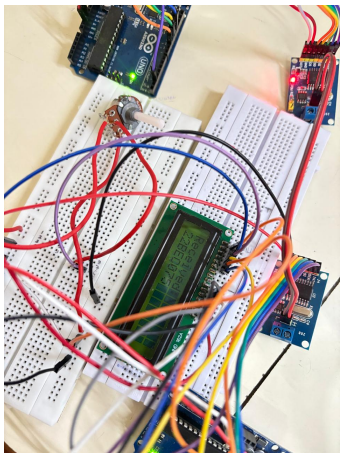

Fig. 3. CAN protocol implementation on Hardware

VIII. Conclusion

This paper presented the implementation of the CAN protocol using two Arduino Uno boards and MCP2515 CAN modules for reliable communication. The system utilized UART for serial input, SPI for data exchange between Arduino and MCP2515, and parallel communication for LCD display on the receiver side. The results demonstrated efficient and error-free data transmission over the CAN bus, validating its robustness in embedded applications. While the implementation performed well, future improvements could include security enhancements, error detection mechanisms, and scalability for multi-node networks.

IX. Acknowledgment

The authors would like to thank the Department of Electronics and Communication Engineering, Nirma University, for providing the necessary resources and support for this work.

References

[1] R. Bosch, “CAN Specification Version 2.0,” Bosch GmbH, 1991.
[2] Microchip Technology Inc., “MCP2515 Stand-Alone CAN Controller with SPI Interface,” Datasheet, 2007.
[3] Arduino, “Arduino Uno Rev3 Documentation,” [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno.
[4] Nahas, M., Pont, M. J., & Short, M. (2009). ”Reducing message-length variations in resource-constrained embedded systems implemented using the Controller Area Network (CAN) protocol.” Journal of Systems Architecture, 55, 344–354.
[5] Lotto, A., Marchiori, F., Brighente, A., & Conti, M. (2024). ”A Survey and Comparative Analysis of Security Properties of CAN Authentication Protocols.” IEEE Communications Surveys & Tutorials.
[6] Bozdal, M., Samie, M., & Jennions, I. (2018). ”A Survey on CAN Bus Protocol: Attacks, Challenges, and Potential Solutions.” IEEE Conference on Vehicular Electronics and Safety.

X. Appendix: Arduino UNO Code

Fig. 4. Arduino UNO code at Transmitter side

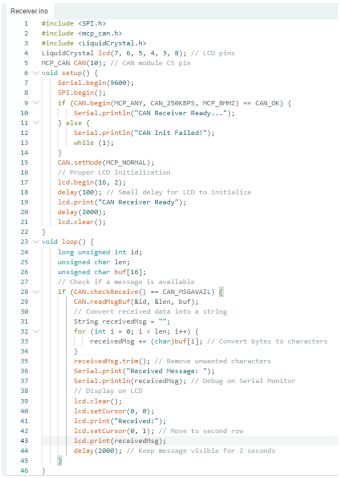

Fig. 5. Arduino UNO code at Receiver side