Bulletin of the *Transilvania* University of Braşov • Vol. 10 (59) No. 2 - 2017
Series I: Engineering Sciences

DESIGN AND IMPLEMENTATION OF A CAN BUS PROTOCOL USING ATMEGA MICROCONTROLLERS

Cristian IOSIF[1]

Abstract: *The Controller Area Network (CAN) is a vehicle bus standard, which allows various electronic components such as microcontrollers, sensors, actuators to communicate with each other without a host computer and up to 1Mb/s. Is a message based protocol designed specifically for automotive but it is also used in areas such as aerospace, maritime and industrial automation. This paper is aimed at the design and implementation of CAN protocol using ATmega microcontrollers.*

Key words: *Controller Area Network, bus, communication, automotive, hardware.*

1. Introduction

The Controller Area Network architecture was developed in the 1980's at Robert Bosch GmbH for automotive applications to enable a robust serial communication.

The goal was to obtain a more reliable, safe and fuel efficient automobile and decrease the weight and complexity of the wiring harness [1], [4].

The CAN bus is a multi-master serial bus which connects minimum two nodes or Electronic Control Units (ECU) in order to communicate. The complexity of a node varies from a simple I/O device to an embedded computer with a CAN interface. The physical bus is made from two wires which connects all nodes [1].

2. The CAN Standard

The International Standardization Organization (ISO) defined serial communications bus.

ISO-11898:2003 describes how the information are passed between network's devices and conforms in terms of layers to the Open Systems Interconnection (OSI). In Figure 1, there are shown the architecture layers, which establish the communication link to an upper level application protocol like CANopen™ [2], [5].

The ISO-11898:2003 standard, with the standard 11-bit or 2048 identifiers is providing transfer rates from 125kbps to 1Mbps. This standard was improved later with the extended 29-bit or 537 millions identifiers [2].

[1] Student Master "Electronics Systems and Integrated Communications", *Transilvania* University of Braşov.

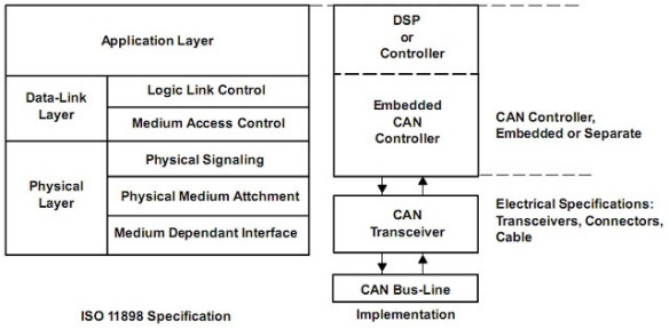

Fig. 1. *The layered ISO 11898 architecture*

2.1. Standard CAN

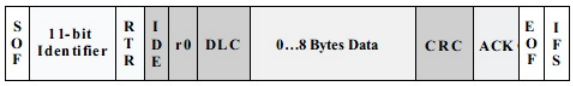

Fig. 2. *Standard CAN: 11-bit identifier*

The meaning of the bit fields are (Figure 2):

- SOF - the single dominant start of frame bit marks the start of the message and synchronizes the devices on the bus after being idle.
- Identifier - sets the message priority, the lower the binary value, the higher the priority.
- RTR - the single remote transmission request bit is dominant when data is required from another device.
- IDE - the dominant single identifier extension bit means is transmitted a CAN identifier with no extension.
- r0 - reserved bit.
- DLC - the 4-bit data length code contains the number of bytes of data being transmitted.
- Data - can be transmitted up to 64 bits of data.
- CRC - the 16-bit cyclic redundancy check contains the checksum of the preceding application data for error detection.
- ACK - every device receiving an accurate message, overwrites this recessive bit in the original message with a dominate bit, indicating that a message without errors has been sent.
- EOF - the 7-bit end of frame marks the end of a CAN message.
- IFS - the 7-bit interframe space contains the time required by the controller to move a correctly received frame to its proper position in a message buffer area [2].

2.2. Extended CAN

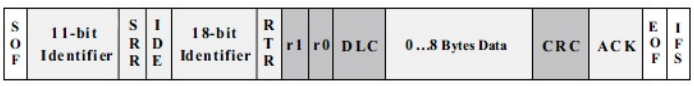

Fig. 3. *Extended CAN: 29-bit identifier*

The extended CAN is similar with the standard CAN with the addition of the following bit fields (Figure 3):

- SRR - the substitute remote request bit replace the RTR bit in the standard message location.
- IDE - a recessive bit in the identifier extension indicates that more identifier bits will follow;
- r1 - an additional reserve bit [2].

3. The CAN Features

Why the CAN bus became so popular over the years in so many industries? Because is a mature standard with excellent error handling, high speeds of data transfer, real time communications and with noise immunity in an electrically noisy environment.

3.1. Message Based Protocol

The CAN bus is a message based protocol which means that every device will receive every transferred message, acknowledge if the message was properly received and it is the decision of every device to keep the message for processing or to discard it.

One advantage of the protocol is that additional devices can be added to the current system without reprogramming all other devices to recognize the new device [6].

3.2. Carrier Sense Multiple Access with Collision Detection

Carrier Sense means that each device of the network will check the bus to verify if there is some data transfer in progress or not and data is transferred only if the bus is free for data transfer. Multiple Access means that every device of the network has an equal priority for data transferring.

Collision Detection means that if two devices of the network are transmitting data in the same time, the devices detects the data collision and bit wise arbitration is used to specify the priority of the message. Thus, the message with higher priority is transmitted over the network [6].

3.3. Arbitration

Since any device may transfer data when the CAN bus is free, two or more devices may begin the transfer in the same time.

This is the moment when the arbitration intervenes. Bit wise arbitration is the process in which the winning device continues to transfer the message. This is a feature that makes the CAN bus very attractive in a real-time control environment.

In the case of allocation of the messages priority, industry groups agreed on the significance on certain messages. As an example, the message 0010 have a higher priority over the message 0011, because the first message has the lowest binary identifier [6].

3.4. Opposite Logic

One of the CAN fundamentals characteristic is the opposite logic state between the bus, the driver input and receiver output. Normally, a logic high is associated with a one and

logic low is associated with a zero. But, on the CAN bus, logic zero is the dominant and logic one is recessive [3].

4. The Proposed Concept

In this proposed system, the CAN bus is used for faster communication between devices. The protocol has two wires, named CAN High and CAN Low. Different devices can be connected to high speed or low speed bus, depending upon the requirement.

This system intent is to implement the message's transmission over the CAN bus.

4.1. SPI

The Serial Peripheral Interface bus was developed with the sole purpose to provide a full-duplex synchronous serial communication between master and slave devices and is used for communications with sensors, real time clocks, analogue to digital converters and so on.

As seen in figure 4, standard SPI master communicate with slave devices using the serial clock (SCK), master out slave in (MOSI), master in slave out (MISO) and slave select (SS) lines. The SS signal must have an unique line, while the other can be shared by the slaves over the same line [7].

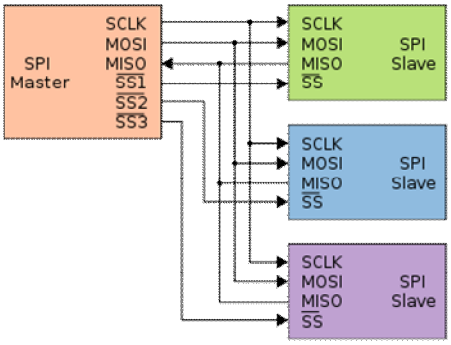

Fig. 4. *Serial Peripheral Interface*

4.2. CAN Controller

Microchip Technology's MCP2515 is a stand-alone Controller Area Network controller that implements version 2.0B of the CAN Specification. Is capable of transmitting and receiving both standard and extended data and remote frames.

The controller has two acceptance masks and six acceptance filters used to filter unwanted messages, which reduce the host microcontroller overhead. It interfaces with microcontrollers through a Serial Peripheral Interface (SPI). It was developed to simplify applications requiring an interface with a CAN bus. The controller consists of three main blocks:

- the CAN module, which contains the CAN protocol engine, masks, filters, transmit and receive buffers;
- the control logic and registers that are used to configure the controller and its operation;
- the SPI protocol block.

4.3. ATMega 328p

Is a low power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, it achieves throughputs of 1 MIPS/MHz making possible to optimize power consumption versus processing speed.

5. The Prototype and Discussions on Results

The parts which are usually connected by the CAN bus are sensors, actuators and other control devices. These devices are not connected directly to the CAN bus, but through a host microprocessor and a CAN controller.

An Electronic Control Unit is also called a node and includes a host microprocessor, a CAN controller and a CAN transceiver:

- the host microprocessor decides which messages are transmitted and received and manages the signals from sensors and actuators;
- the CAN controller stored the received bits until the message is complete and then stores this message;
- the CAN transceiver converts the transmit-bit received by the CAN controller into a signal which is sent thought the bus.

The block diagram for the CAN bus prototype studied in this paper is shown in Figure 5.

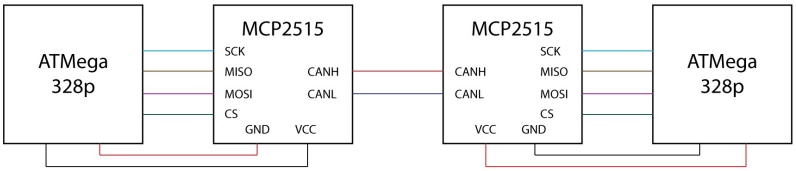

Fig. 5. *The Block Diagram*

One microprocessor is considered the receiving ECU and the other one is attached to a Electronic Brake Control Module.

The CAN bus have a baud rate of 500 kbps and the messages are transmitted with a delay of 1 second.

```
CAN BUS Initialized Successfully!
Data                        Status
-------------------------------------------
00 10 00 11 00 40 07 00     Data Sent Successfully!
00 10 00 11 00 40 07 00     Data Sent Successfully!
00 10 00 11 00 41 07 00     Data Sent Successfully!
00 10 00 11 00 40 07 00     Data Sent Successfully!
00 10 00 11 00 41 07 00     Data Sent Successfully!
00 10 00 11 00 41 07 00     Data Sent Successfully!
00 10 00 11 00 40 07 00     Data Sent Successfully!
00 10 00 11 00 40 07 00     Data Sent Successfully!
00 10 00 11 00 40 07 00     Data Sent Successfully!
00 10 00 11 00 41 07 00     Data Sent Successfully!
00 10 00 11 00 41 07 00     Data Sent Successfully!
00 10 00 11 00 41 07 00     Data Sent Successfully!
00 10 00 11 00 41 07 00     Data Sent Successfully!
00 10 00 11 00 40 07 00     Data Sent Successfully!
00 10 00 11 00 40 07 00     Data Sent Successfully!
00 10 00 11 00 40 07 00     Data Sent Successfully!
00 10 00 11 00 40 07 00     Data Sent Successfully!
00 10 00 11 00 40 07 00     Data Sent Successfully!
00 10 00 11 00 40 07 00     Data Sent Successfully!
00 10 00 11 00 40 07 00     Data Sent Successfully!
```

Fig. 6a. *The sending ECU*

```
CAN BUS Initialized Successfully!
Type          ID    DLC    Data
-------------------------------------------
Standard CAN:  119    8     00 10 00 11 00 40 07 00
Standard CAN:  119    8     00 10 00 11 00 40 07 00
Standard CAN:  119    8     00 10 00 11 00 41 07 00
Standard CAN:  119    8     00 10 00 11 00 40 07 00
Standard CAN:  119    8     00 10 00 11 00 41 07 00
Standard CAN:  119    8     00 10 00 11 00 41 07 00
Standard CAN:  119    8     00 10 00 11 00 40 07 00
Standard CAN:  119    8     00 10 00 11 00 40 07 00
Standard CAN:  119    8     00 10 00 11 00 40 07 00
Standard CAN:  119    8     00 10 00 11 00 41 07 00
Standard CAN:  119    8     00 10 00 11 00 41 07 00
Standard CAN:  119    8     00 10 00 11 00 41 07 00
Standard CAN:  119    8     00 10 00 11 00 40 07 00
Standard CAN:  119    8     00 10 00 11 00 40 07 00
Standard CAN:  119    8     00 10 00 11 00 40 07 00
Standard CAN:  119    8     00 10 00 11 00 40 07 00
Standard CAN:  119    8     00 10 00 11 00 40 07 00
Standard CAN:  119    8     00 10 00 11 00 40 07 00
Standard CAN:  119    8     00 10 00 11 00 40 07 00
Standard CAN:  119    8     00 10 00 11 00 40 07 00
```

Fig. 6b. *The receiving ECU*

In Figure 6a, it can be seen that the CAN bus was initialized successfully and then it starts to send messages to the receiving ECU.

There are 2 fields present:

- data, which is the message currently sent through the bus;
- status, which is the status of the message sent or not through the bus.

The tests were performed in real-time and pressing the pedal brake and monitoring the messages transmitted over the CAN bus was simulated.

As it can be seen, the low nibble of the third byte is changing the value when the pedal brake is pressed; it starts with the value 40 when the pedal is not pressed, and changing to 41 when the pedal is pressed.

Figure 6b shows that the CAN initialized successfully and we have 4 fields of data:

- type, which shows that the message is a 11-bit Standard CAN message;
- ID, the Identifier of the sending ECU;
- DLC, the number of bytes sent through the bus;
- data, which is the message sent through the bus.

In can be seen in both Figure 6a and Figure 6b that the messages sent through the CAN bus are corresponding and transmitted without errors.

6. Conclusions

This paper provided a brief study about the implementation of the CAN bus in the automotive industry.

From the above results, it can conclude that simulation of brake pedal which is pressed or not was monitored and transmitted with success over the CAN bus.

The main goal of this project was to develop a prototype system of a CAN bus which can monitor various parameters of a vehicle. This project can be extended in the future to monitor other parameters such as rpm's, temperatures and so on.

References

1. Bosch, R.: *CAN Specification Version 2.0.* September 1991.
2. Corrigan, S.: *Introduction to the Controller Area Network.* In: Texas Instruments Application Report, SLOA101A, August 2002 - Revised July 2008.
3. Di Natale, M., Zeng, H., Guisto, P., Ghosal, A.: *Understanding and Using the Controller Area Network Communication Protocol*. Springer Science & Business Media, 2012.
4. Johansson, K.H., Tongren, M., Nielsen, L.: *Vehicle Application of Controller Area Network*. In: Proc. of the Handbook of Networked and Embedded Control Systems, VI, 2005, p. 741-76.
5. Pazul, K.: *Controller Area Network (CAN) Basics.* In: Microchip Technology Inc, 2002.
6. Navet, N., Perrault, H.: *CAN in Automotive Applications: A Look Forward*. In: 13th International CAN Conference, Hambach Castle, 2012.
7. *** *Stand-Alone CAN Controller with SPI interface.* Microchip Technology Inc, 2007.