IMPLEMENTING THE CONTROLLER AREA NETWORK (CAN) PROTOCOL FOR MULTIPLEX SYSTEM

Zarina Mohd. Noh[1], Muhammad Nasir Ibrahim[2], Muzalifah Mohd. Said[1], Norhidayah Mohamad Yatim[1], Rostam Affendi Hamzah[1]

[1]Faculty of Electronics & Computer Engineering, Universiti Teknikal Malaysia Melaka (UTeM), 76100 Durian Tunggal, Melaka

[2]Faculty of Electrical Engineering, Universiti Teknologi Malaysia (UTM), 81310 Skudai, Johor

zarina.noh@utem.edu.my

Abstract

Controller Area Network (CAN) protocol can be implemented in networking a system by the application of PIC18F458 microcontroller and MCP2551 transceiver. For successful transmission, design specification concerning the CAN node identifier assignment, message reception pattern and time synchronization must be determined accordingly. The CAN network developed in this article performs a simple application of transmitting a one-byte data through a three-node network. This eye-opener CAN system realization may act as a platform for researchers who are interested in the application of CAN protocol in networking a multiplex system. The work can also be used for analyzing purposes concerning the CAN usage in domestic embedded system application networking.

Keywords: *communication protocol, Controller Area Network (CAN), networking, synchronization.*

I. INTRODUCTION

Previously developed for In-Vehicle Network (IVN) protocol, CAN application had been extended to factory automation, industrial machine control, lifts and escalators, building automation, medical equipment and devices, and non-industrial control and equipment [1]. It is a technology that is guaranteed well into the future as large numbers of semiconductor manufacturers such as Philips, Motorola, Microchip, and National Semiconductors are now producing CAN devices due to its widespread use [2].

As a low cost networking alternative, CAN is a powerful tool for general-purpose sensor/actuator bus system for distributed real time control which can be utilized with any microcontroller-based system [1]. It allows the application of multi-master architecture on a broadcast shared bus, operating over a simple twisted pair wire imposing the simplicity of the network cabling [2][3]. With CAN Version 2.0A, a single network (theoretically) may consist of up to 2032 nodes while CAN Version 2.0B may consists of more that 500 million nodes in a single network [4]. Specifically concern on layer 2 (data link layer) and part of layer 1 (physical layer) in the ISO/OSI (International Standardization Organization/Open System Interconnection) model, this networking technology imposes some advantages in its arbitration technique, message framing and filtering, error confinement, and also in its bit representation [4]. All of those suggest the use of CAN protocol as a reliable embedded system networking solution.

In the communication point of view, CAN have its own specialties and strengths. Among those is the absence of indication of the originating or destination addresses for each of its messages. Instead, an identifier is embedded in the message and each controller in the network is responsible in determining the receipt of messages by deciphering the headers of the messages. The advantages of such scheme are one can add a controller to the

network without stopping the operation and the network allows multi-casting capabilities [5].

As an event-triggered protocol, signal transmission on the bus only happened if there is any messages need to be sent by any nodes in the CAN network [6]. This is another specialties of CAN protocol which ensures efficient bandwidth utilization through the bus network.

In accessing the network, CAN employs the Carrier Sense, Multiple Access with Collision Avoidance (CSMA/CA) mechanism in providing the arbitration access to the bus [7]. The bits in the identifier of CAN messages which are categories into either dominant (logic zero) or recessive (logic one), plays an important role in handling the bit collision happened during simultaneous transmission by two or more nodes. The CAN network is structured such that the dominant bit stays on the bus while the recessive bit lost its arbitration. Hence, the highest priority identifier will always win the arbitration and successfully control the bus. This offers message prioritization and access conflict resolution in the network, which is important for a dependable system.

The CAN protocol also provides a high-level of error detection and correction [2]. The built-in error detection of the CAN controller together with the error signaling ensures that the information transmitted is correct and consistent. When a CAN controller detects an error (either bit errors or message errors), an error frame is immediately transmitted in the network. The message (in error) is then cancelled at all nodes and the correct message is then retransmitted. A continuous retransmission of messages (in error) caused the faulty node to fall to a mode where it will not disturb the traffic on the bus. This is called as 'bus-off' state and no messages can be received or transmitted until the host microcontroller reset the node. The automatic handling of error ensures the reliability of CAN protocol.

The following section in this article discusses on the CAN protocol design issues for a network, the hardware implementation related, the network system operation and recommendation for future research work.

II. CAN IMPLEMENTATION

To implement the CAN protocol in a network, the network must consists of a group of CAN nodes, connected to each other via the CAN-bus. For a node to be regarded as a CAN node, it must comprises of its own controller, CAN protocol handler and the CAN line interface. Fig. 1 illustrates a CAN node for a system, disregard to the type of controller used in the networking system.

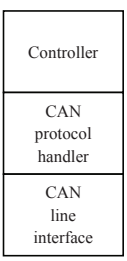

Fig. 1. CAN node

In Fig. 1, the controller in the CAN node acts as the main processing unit of the node, which performs functions and operations as needed. The CAN protocol handler will configure the node, such that its configuration specification is as intended for the networking purposes. The CAN line interface convert the data (receives from the CAN network or transmits by the node's controller) into the form that can be understood by the CAN network and the intended node. The data can be sent up to 8-bytes length in a single CAN message through the CAN network [4].

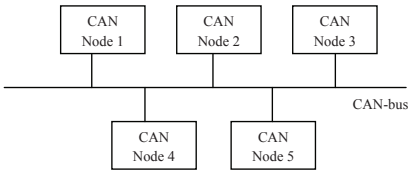

Fig. 2. CAN network

Each of the CAN node connected together to form a network via the CAN-bus as illustrated in Fig. 2. In this article, the CAN system hardware is implemented by the use of PIC18F458 as the controller for the CAN node in the network, as the chip is readily integrated with the CAN protocol handler. The chosen chip proves to save space as well as programming hassles as most of the CAN system specification are made to be easily used by the CAN protocol users. The CAN line interface which is also known as the CAN transceiver is realized by the use of MCP2551 chip. It is chosen as the MCP2551 chip is compatible with the ISO-11898 which defines the CAN and its physical layer implementation [8]. Together, the PIC18F458 and MCP2551 chip make up the CAN node. The CAN-bus in Fig. 2 can be developed by the usage of a simple twisted pair wire, connected to each of the CAN nodes in the network.

Fig. 3 illustrates the CAN hardware implementation used for the purpose of this research article.

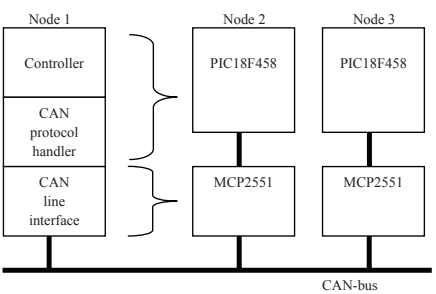

Fig. 3. CAN hardware implementation

III. DESIGN ISSUES

In ensuring successful data transmission in the system, the CAN protocol handler (also known as CAN module) of PIC18F458 is fully accessed in designing its application operation. Among the issues that need to be taken into consideration in the designing phase are the node identifier assignment, the message reception specification for each node, and also the time synchronization between the nodes in the network.

As each of the node is differentiated by its unique identifier, a higher priority node will be given the bit value (as its identifier) that will win arbitration most of the time in the network. In this case, whenever there are two or more nodes tries to transmit messages through the network at the same time, the higher priority node will always wins the network bus and the lower priority node will need to wait for the next turn. In CAN network, '0' binary bit is regarded to have a higher priority than the '1' binary bit value.

In this article, the CAN network is designed such that the identifiers are assigned randomly as the network is assumed to be constructed in an ad-hoc manner. The identifier is chosen from the combination of 29-bit value as the network deals with an extended CAN data frame format (CAN version 2.0B). The identifier value for each of the node in the CAN network developed is as shown in Table 1. It can be seen that the Node 2 has higher priority than Node 1 and Node 3. In case where transmission occurred at the same time from all the nodes, Node 2 will always win the arbitration.

Table 1. Node identifier value

	Identifier
Node 1	12111
Node 2	3
Node 3	113

Because all nodes in a CAN network will receive all the messages transmitted over the CAN-bus, the receiving buffer in a node should be restricted to receive only

the message of its interest. Fig. 4 shows the block diagram of a CAN node receiving buffer offered by the CAN module of PIC18F458. Different types of CAN protocol handler might have different diagram of receiving buffer block, but all of them can be specified to receive only messages of its interest.

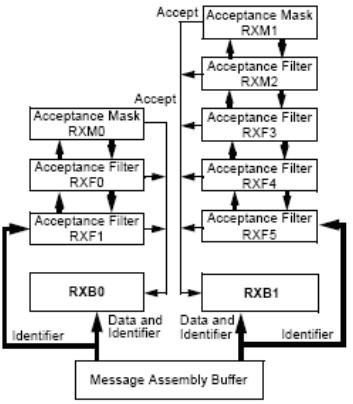

Fig. 4. Receive buffer block diagram [9]

As illustrated in Fig. 4, any messages detected on the CAN-bus will be checked for errors and loaded into the Message Assembly Buffer. It is done by manipulating the mask and acceptance filter bit value in each node. In PIC18F458, 6 acceptance filters are available for this purpose (RXF0-RXF5). The received message is matched against the filters to see if it should be received and stored in one of the receiving buffers (RXB0 or RXB1) of the node. Otherwise, the received message will be discarded. Table 2 shows the specification of the Acceptance Filter RXF1 for each of the nodes in the CAN network developed.

Table 2. Acceptance Filter RXF1 specification

Receiving Node	Acceptance Filter value
Node 1	113
Node 2	12111
Node 3	3

In this article, the designed network will make use only one of the 6 acceptance filter in its specification. Only messages originated from the specified identifier value (node) will be meaningful to the respective receiving node.

Another issue that needs to be considered in the designing phase of the CAN network is the time synchronization of the system. Practically, the oscillators used in a node and time transmission in the other node may differ in each other due to different clock speed and device propagation time delay. The receiving node needs to synchronize its clock to the transmitting node, so that the transmitted data may be recovered. In PIC18F458, the synchronization of the incoming data is done by the Digital Phase Lock Loop (DPLL) which also provides the nominal bit timing for the transmitted data [9]. The nominal bit time is divided into 4 non-overlapping time segments (represented by multiple of time quanta or TQ) as shown in Fig. 5.

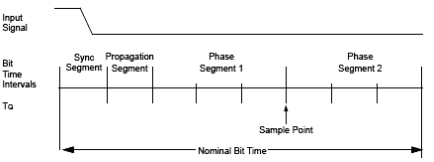

Fig. 5. Nominal bit time division [9]

T_Q is a fixed unit derived from the oscillator period while the nominal bit time is needed to indicate the nominal bit rate of the network. The nominal bit rate specifies the number of bit transmitted per second for an ideal system where no synchronization is needed in the network [9]. By specifying the nominal bit rate needed for a CAN network, the oscillator value that should be used by a CAN node in the network can be obtained. In this article, the nominal bit timing is specified according to the requirement needed for successful CAN network synchronization as shown in Table 3 [7].

Table 3. Nominal bit timing specification

	t_Q
SyncSeg	1
PropSeg	1
PhaseSeg1	3
PhaseSeg2	3

IV. SYSTEM SPECIFICATION & OPERATION

A. Hardware Realization

The functional block diagram of the developed CAN network is as shown in Fig. 6. Each node has 4 LEDs connected to Port C (I/O port) of PIC18F458 which displays the data received from the CAN network. These LEDs served as the output of each node. Once the 'Reset Switch' in the first node is activated, the Node 1 will start sending the data to the CAN network. The network operation is as shown in Fig. 9. This simple configuration can be further extended to an application specific system to fully utilize the advantages of CAN protocol.

B. CAN Module Specification

The utilization of the CAN module in handling the design issues as mentioned in previous section is done by using the mikroC software. The software supports C-language for PIC implementation, which also has the CAN related functions in its library definition. Among those defined CAN functions available are CANInitialize(…), CANSetOperationMode(…), CANSetMask(…), CANSetFilter(…), CANWrite(…) and CANRead(…) [10]. Some of these functions application are as shown in Fig. 7 for the module initialization steps.

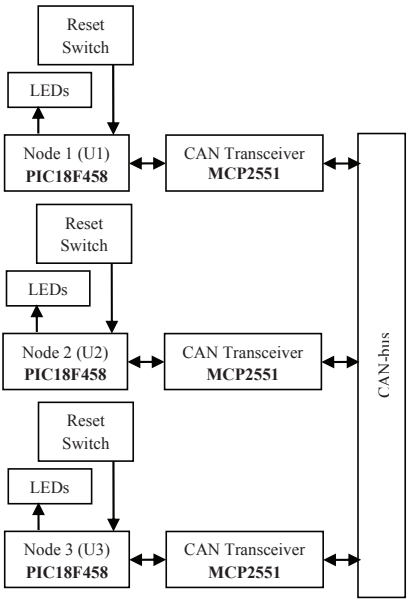

Fig. 6. CAN network

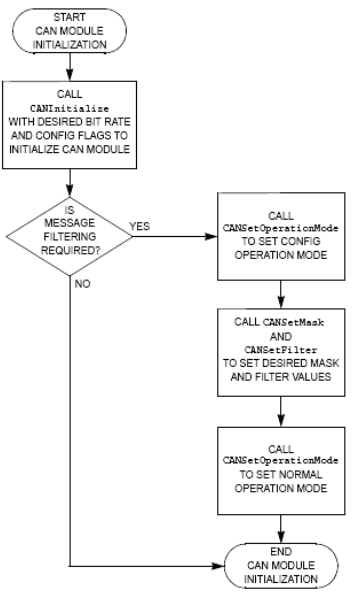

Fig. 7. CAN module initialization steps [10]

In configuring a node to be one of the nodes in a CAN network, the CAN module in the node needs to be initialized. Fig. 7 shows the flowchart of the steps taken in performing the CAN module initialization for a CAN node. It is at this stage that the CAN protocol is specified to either uses which CAN data format or

how the bit timing partitioning is done in the network. Part of the program for CAN module initialization is as shown in Fig. 8.

```
// Initializing CAN related bits and module
Can_Init_Flags = 0;
Can_Send_Flags = 0;
Can_Rcv_Flags = 0;
RxTx_Data[0] = 1;    // R[0] = 1

Can_Init_Flags = CAN_CONFIG_SAMPLE_THRICE &
CAN_CONFIG_PHSEG2_PRG_ON &
CAN_CONFIG_XTD_MSG &
CAN_CONFIG_DBL_BUFFER_ON &
CAN_CONFIG_VALID_XTD_MSG;

Can_Send_Flags = CAN_TX_PRIORITY_0 &
CAN_TX_XTD_FRAME &
CAN_TX_NO_RTR_FRAME;

CANInitialize(1,3,3,3,1,Can_Init_Flags);
CANSetOperationMode(CAN_MODE_CONFIG,0xFF);
CANSetMask(CAN_MASK_B1,-1,CAN_CONFIG_XTD_MSG);
CANSetMask(CAN_MASK_B2,-1,CAN_CONFIG_XTD_MSG);
CANSetFilter(CAN_FILTER_B1_F1,113,CAN_CONFIG_XTD_MSG);
```

Fig. 8. Software coding in C-language (CAN module)

Once the CAN module in the protocol handler had been initialized, the node can starts transmit or receive messages through the CAN network after assigning its node identifier value. The remaining program may differ according to the specific application performed by the CAN node in the network.

C. Network Operation

The communication in the CAN network developed is initiated by Node 1, which transmit one byte of data with value 1 (R[0] = 1). The network is configured to use the extended frame format, where the three nodes are specified to transmit, receive, accept and decipher the message in terms of CAN Version 2.0B format.

Fig. 9 shows the sequence of operation performed by the CAN network in this article. The application starts with Node 1 sending the message which consists of data R[0] to the CAN network. Node 2 and node 3, upon receiving the message through the network, received the transmitted message and compares the message identifier with their Acceptance Filter specification. As Node 3 acceptance filter had been set to receive only messages transmitted from Node 2, message received from Node 1 will be discarded by Node 3 following its acceptance filter. (Refer to the design specification in Table 2). Node 2 will receive this message and displays the data R[0] via Port C. Node 2 will then transmit the message received from Node 1, to the CAN network. Upon receiving message from the network, Node 3 will display the data R[0] through Port C. (Node 1, as it also receives the message transmitted in the network, discards the message transmits by Node 2 as it only interested in the message transmitted by Node 3). The data R[0] will be increased by 1 before the updated message were transmitted again over the network by Node 3. The transmitted message will then be received by Node 1 (Node 2 discard the message received from Node 3) and the update data R[0] (which had been increased by 1) will be displayed in Port C.

This operation will continue until the message transmitted through the network is received in error.

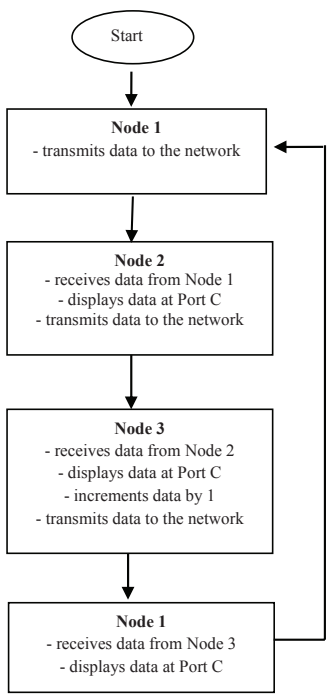

Fig. 9. CAN network operation

The simple CAN system had been successfully implemented, performing simple application as mentioned in the previous section to imitate communication in between the nodes through the CAN network. In implementing the system for any embedded system application, the data byte sent (R[0]) may hold any value of data which is meaningful to be transmitted around the network.

As the network only focuses on the transmission of data frame (extended format), further research work can be done in manipulating the transmission of error frame in demonstrating the error confinement specified by the CAN module. Extended work can also be done in manipulating the acceptance filter specification in the node, such that the intended node may receives messages from more than one node. In this case, the identifier assigned for each node may play an important role for the prioritization purposes. The CAN network in this article can also be expanded to more than three nodes to see the effect of synchronization performed in the network. The value of the time quanta (TQ) might need some adjustment in compensating the delay imposed by the bigger network. Research can also be done for successful application specific purposes utilizing the CAN protocol advantages in the network.
As such, this networking protocol can be fully accessed in the development of a distributed embedded system networking.

ACKNOWLEDGMENT

The authors would like to thank Universiti Teknikal Malaysia Melaka (UTeM) and Universiti Teknologi Malaysia (UTM) for great facilities and continuous support in completing this research. This work is highly related to the authors' previous work in 'The Development of a Dependable Distributed Embedded System'.

REFERENCES

[1] Cenesiz, N., and Esin, M. (2004). "Controller Area Network (CAN) for Computer Integrated Manufacturing Systems". Journal of Intelligent Manufacturing. Vol. 15(4), 481-489.

[2] Farsi, M., Ratcliff, K., and Barbosa, M. (1999). "An Overview of Controller Area Network". Computing & Control Engineering Journal. Vol. 10(3), 113-120.

[3] ARTIST FP5 Consortium (2005). "Networks. In: LNCS 3436. Embedded Systems Design". (316-337). Heidelberg: Springer Berlin.

[4] Paret, D. (2007). "Multiplexed Networks for Embedded Systems". England: John-Wiley Sons.

[5] Varsekalis, D. H-, and Levine, W. S. (2005). "Handbook of Networked & Embedded Control Systems". Boston: Birkhauser.

[6] Kinoshima, T., Kobayashi, K., Zakaria, N. A., Kimura, M., Matsumoto, N., and Yoshida, N. (2007). "Communication Model Exploration for Distributed Embedded Systems and System Level Interpretations". In: LNCS 4809. Emerging Directions in Embedded and Ubiquitous Computing. (355-364) Heidelberg: Springer Berlin.

[7] Microchip Technology Inc. (1999). "AN713 Controller Area Network (CAN) Basics". [Application Note]. USA: Microchip Technology Inc.

[8] Microchip Technology Inc. (2002). "AN228 A CAN Physical Layer Discussion". [Application Note]. USA: Microchip Technology Inc.

[9] Microchip Technology Inc. (2004). "PIC18FXX8 Data Sheet (DS41159)". [Datasheet] USA: Microchip Technology Inc.

[10] Microchip Technology Inc. (2002). "PIC18C CAN Routines in 'C'. [Application Note]". USA: Microchip Technology Inc.