Proceedings of the Fifteenth Annual Early Career Technical Conference
The University of Alabama, Birmingham ECTC 2015
November 7 – 8, 2015 - Birmingham, Alabama USA

A BEGINNER'S GUIDE TO CONTROLLER AREA NETWORK BUS ACCESS IN MODERN VEHICLES

Timothy D. Fisher, Dr. Kevin S. McFall
Kennesaw State University
Kennesaw, Georgia, United States of America

ABSTRACT

A controller area network (CAN) is a communication system designed so that a microcontroller governing multiple systems (nodes) on the network can function effectively. This network can be accessed, interpreted, and manipulated by an external computer. The purpose of this paper is to provide a basic working knowledge of CAN architecture and protocols, as well as how to connect to and decode a vehicular controller area network.

INTRODUCTION

CAN protocol was introduced by the Society of Automotive Engineers (SAE) in February of 1986 [1] as a multiplexed system for sending messages between devices in an automobile. The first production vehicle implementing the system was the BMW 8 Series line, starting in 1989. In 1992, CAN 2.0 was published with provisions for devices with both 11-bit and 29-bit identifiers (often referred to as CAN 2.0A and CAN 2.0B, respectively). CAN 2.0 remains the foundation of present day CAN architecture. The most recent update to the CAN protocol, published in 2012, is CAN FD 1.0, or CAN with Flexible Data-Rate. [2] This new format allows variation in both the size of the data package and in the bit rate. CAN FD 1.0 is fully back-compatible with CAN 2.0.

CAN 2.0 is also used in the on-board diagnostics (OBD-II) system, which allows an external device to communicate with an automobile and assess its functionality. Starting in 1996, the OBD-II system was made mandatory in many types of vehicles throughout the world, and specifically those sold in the United States and the European Union (which uses a marginally different standard called EOBD).

Though CAN FD 1.0 was the last major update to the CAN protocol, the technology continues both to improve and to provide a basis for other improvements. In 2007, the New Jersey State Police integrated the pursuit light package in its patrol cars into the CAN bus in order to reduce extraneous wiring in the cockpit, and to integrate more subsystems so that maintenance time could be reduced. [3] In 2014, General Motors introduced a wireless connection system called ViCAN with the goals of making it possible to connect devices to the network without physically wiring them into the bus, and to further reduce the amount of wiring needed in the vehicle by removing the need for a common wire pair connecting every device. [4] Earlier this year, a new safety system that automatically dims headlights for oncoming traffic, checks for short circuits in vehicle wiring that can cause fires and rapid battery drain, detects flammable fumes near the engine, and monitors engine temperature was introduced at a conference at Hindustan University. [5] Most recently, Kennesaw State University began research into autonomously controlling a 2012 Kia Optima via

manipulation of the CAN bus. This is the project that generated the information presented in this paper.

BACKGROUND

The CAN bus itself is really very simple. It consists of two wires connected to every CAN device in the vehicle. Instead of one wire carrying a signal and the other functioning as ground, one carries high voltage and one carries low voltage. A signal on the low voltage line is designated as a logical 1 (recessive), and a signal on the high voltage line is designated as a logical 0 (dominant).

An individual message that a CAN node sends is called a frame. Every frame begins with an identification number unique to each node. The ID numbers also set message priority on the CAN bus, with lower ID numbers taking precedence over higher numbers. For instance, a high-priority system such as the Engine Control Unit would have a very low ID number, and low-priority systems such as power locks would have a higher ID number. This precedence is established through the following protocol, known as arbitration.

When one or more CAN nodes transmit a logical 0, all nodes receive a logical 0. When one or more CAN nodes transmit a logical 1 and at least one other node transmits a logical 0, all nodes receive a logical 0, including the ones transmitting a logical 1. When a node transmits a 1 and sees a 0 during identification, this tells the node that there is a higher priority message being transmitted by a different node, and it will wait until it receives an End-of-Frame signal before reattempting transmission.

Priority on the CAN bus is indicated by a node's identification number, as noted above. A lower identification number will have a lower binary value, and it will win arbitration over nodes with higher ID numbers. If a node completes its ID transmission without interruption, it will transmit the rest of its message.

Take, for instance, a conflict between two hypothetical nodes with IDs of 47 and 11, respectively, which starts on the same clock cycle. The CAN bus arbitration for their 11-bit ID transmission is displayed in Table 1. ID Arbitration.

Table 1. ID Arbitration

Bit:	10	9	8	7	6	5	4	3	2	1	0
Node 11	0	0	0	0	0	0	0	1	0	1	1
Node 47	0	0	0	0	0	1	Not transmitting				
CAN	0	0	0	0	0	0	0	1	0	1	1

Table 2. Can Base Frame Format

0	0	0	0	0	0	0	0	1	0	1	1	0	0
0	0	0	0	1	0	1	1	1	0	0	1	0	0
1	1	0	1	0	1	0	0	1	0	1	0	0	0
1	1	1	1	1	1	1	1	1	1	Inter-frame space			

Table 3. CAN Extended Frame Format

0	0	0	0	0	0	0	0	1	0	1	1	1	1
1	0	0	0	0	1	1	0	0	1	1	1	0	0
0	1	0	1	0	0	0	0	0	0	1	0	1	1
1	0	0	1	0	0	1	1	0	1	0	1	0	0
1	0	1	0	0	0	1	1	1	1	1	1	1	1
1	1	Inter-frame space											

ID
IDE
DLC
DATA
CRC
ACK
EOF
Misc.

Figure 1. Legend for Shading in Tables 2 and 3

A standard frame can have between 55 and 131 data bits in it, depending on whether it uses a base frame with 11 ID bits or an extended frame with 29 ID bits. How much DATA is included in the frame (between 1 and 8 bytes, or 8 and 64 bits) also has an effect. Aside from the ID and DATA bits, the frames are mostly similar.

The first bit of a frame is always a dominant 0 to cut through the recessive background and alert

other nodes that a frame is beginning. The next 11 bits are the node ID.

The subsequent bit changes depending on whether the base frame format (BFF, seen in Table 2) or extended frame format (EFF, seen in Table 3) is used. The shading key for Table 2 and Table 3 can be found in Figure 1. In BFF, this bit after the node ID is the remote transmission request (RTR) bit, which is dominant for data frames and recessive for remote requests. Normally nodes transmit information as a matter of routine, but if a node wishes to query a different node for information, it will set the RTR bit to 1 as an alert to other nodes that it is requesting information. In EFF, this bit instead becomes the substitute remote request (SRR), and will always be recessive.

After the RTR/SRR bit comes the identifier extension bit (IDE). This will be dominant if using BFF, and recessive if using EFF. If the IDE bit is recessive, the next 18 bits will be the remainder of the ID, and the 19th will be the RTR. In BFF, the IDE will be followed by 1 dominant bit, and in EFF, the RTR will be followed by 2 dominant bits.

From this point, BFF and EFF are formatted the same way. The next 4 bits are the data length code (DLC), which describe how many bytes the following DATA field will be (up to 8). The DATA field contains the message that the rest of the frame acts as an envelope for, ensuring its integrity and that it goes to the correct destination. The contents of the DATA frame will change depending on which node it was generated by or addressed to (e.g. lock all doors, activate right turn signal, set tachometer to 4500 RPM).

Following this is a 15 bit cyclic redundancy check (CRC)[1] to test data integrity, and 1 recessive CRC delimiter bit. The frame then drops to a recessive acknowledgement (ACK) bit, during which any other node which has been paying attention will transmit a dominant bit if it has read the current frame as valid. A recessive ACK delimiter bit follows this, followed by a total of 7 recessive end-of-frame (EOF) bits to indicate that the frame is complete and that another node may begin transmitting.

[1] See Appendix for computation

While a normal frame will typically contain between 55 and 131 useful bits, it may occasionally contain more due to an error-checking method known as “bit stuffing.” Six sequential bits of the same type (dominant or recessive) will be read as an error by any receiving nodes, so when this occurs naturally, an opposing bit will be inserted between the fifth and sixth repeated bit. This does not occur with the CRC delimiter, either ACK bit, or the end-of-frame bits, all of which are of fixed size and value.

None of this information is strictly necessary in order to interpret the CAN bus using the methods described below, but it can help to understand why the network behaves as it does.

PROCEDURE

The most reliable way to get started with the CAN system is to dismantle part of a car and hook up to the CAN bus that way. However, sometimes an expendable vehicle may not be available. In such cases, there are commercial CAN simulators available that are useful for training, troubleshooting, and to some degree, debugging. [6]

The easiest way to access the CAN bus in a modern vehicle is via the On-Board Diagnostics port (OBD-II) under the driver’s side dashboard. This port is installed in most modern vehicles as a way of directly communicating with the microcontroller running the network, and includes both the high and low (i.e. CAN-H, CAN-L) terminals needed to access the CAN network.

However, on occasion this method may not work. During the course of the research that led to this paper, it was discovered that the OBD-II port of the project’s Kia Optima was unresponsive. In such a situation, the next course of action is to find a CAN-connected device whose wire harness can be used to read the CAN bus. Qualifying devices will vary from vehicle to vehicle, but the instrument cluster (speedometer, odometer, etc.) will typically have a CAN line if it is not mechanically actuated.

Once the CAN bus has been located, it will be necessary to connect it to a computer. A PEAK-

System OBD-II to D-sub adapter was used with accompanying software for this research, but any compatible combination of adapters and software would function adequately. After a physical connection has been established, the bit rate must be synchronized. A modern computer will often have a clock speed of several GHz, whereas a CAN microcontroller will typically have a transmission rate in the kHz to MHz range.

Upon accessing the CAN bus and synchronizing the bitrate with the computer, decoding the bus is as simple as manipulating equipment in the vehicle and recording the changes in the data values for whatever addresses are present on the bus. This process of experimentation is necessary because the DATA field is so densely packed with information that it has to be reduced to machine code in order to fit inside a standard CAN frame. The formatting and meaning of this code will vary from node to node, so the only way to interpret it without proprietary information or software from the manufacturer is via trial and error.

RESULTS

Connection to the car was made via the CAN ports on pins 31 and 32 of the instrument cluster harness at a bit rate of 100 kHz. The general state of the CAN bus immediately after ignition can be seen below in Table 4.

Table 4. Vehicle After Ignition

ID	DATA							
100h	31	1C	00	00	08	20	00	00
101h	80	40	11	00	00	00	00	00
104h	05	6E	00	00	00	00	00	00
10Ch	00	00	00	12	00	00	00	00
10Dh	16	83	C8	20	06	00	00	00
400h	01	02	00	00	00	00	FF	FF
401h	0F	02	00	00	00	00	FF	FF
40Fh	00	02	00	00	00	00	FF	FF
501h	00	03	48	00	00	00	00	00

The first observation that was made when the car had finished its power-on procedures was that bytes 2 and 3 in word 501h fluctuated slightly when the vehicle was completely idle and parked. It is hypothesized that these two bytes, and possibly byte 1, are tachometer readings. Converting 0348 from hexadecimal to decimal yields 840, which is believable for an RPM reading from an engine recently started. Further research will be made when a splitter has been fashioned that will allow the computer and the instrument cluster to read from the cluster harness at the same time. This will confirm whether the raw number corresponds to the RPM.

When the brake pedal was pressed, byte 4 of word 10Dh changed from 00 to 02, setting bit 3 of the word high. This is likely the bit that controls the brake lights, or the sensor indicating that the brakes are engaged. Bytes 1 and 2 of the same word appear to keep track of the current state of the power system and the motor, respectively.

Word 104h seems to deal mainly with safety features such as locks and flashers. Word 1 changes states depending on the state of the driver's side back door (open or closed, locked or unlocked), and word 2 performs the same function for both front doors. No data was collected on word 3, but logically, it might relate to the passenger side back door. Word 4 controls the hazard lights, and likely the turn signals.

CONCLUSION

Preliminary results on this project are encouraging. By mapping some of the more easily-accessible CAN codes, less important data generated by normal operation of devices in the car can be filtered out, and more useful codes such as those for cruise control or ABS can be more easily isolated. The 2012 Kia Optima is lacking in what would traditionally be considered drive-by-wire systems, but it may yet become unmanned via more unconventional means.

APPENDIX

A CRC check is performed on the data in every CAN frame to ensure data integrity. This is accomplished by adding a number of 0s to the end of the DATA field equal to the size of the CRC field (15 bits) and performing an XOR operation on the DATA field with a predetermined divisor (for CAN applications, 0x4599), aligning with the leftmost 1 and working to the right until the remainder is 0. Each progression aligns the first 1 in the divisor with the first 1 in the dividend. The process for the CRC field in Table 2 and Table 3 is shown below in Table 5.

Table 5. CRC Calculations

0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	1	1	0	0	1	1	0	0	1							
0	0	1	1	0	1	1	1	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0

0	0	1	1	0	1	1	1	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0
	0	1	0	0	0	1	0	1	1	0	0	1	1	0	0	1						
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	0	0	0	0

0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	0	0	0	0
		0	1	0	0	0	1	0	1	1	0	0	1	1	0	0	1					
0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	1	1	1	0	0	0	0	0

0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	1	1	1	0	0	0	0	0
				0	1	0	0	0	1	0	1	1	0	0	1	1	0	0	1			
0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	0	1	0	1	0	0	0

Since the remainder of the DATA field is 0, the process is now complete, and the result becomes the CRC field seen in Table 2 and Table 3.

REFERENCES

[1] Embedded Systems Academy, "Almost 25 Years of CAN," [Online]. Available: http://www.canopen.us/history.htm. [Accessed 12 August 2015].

[2] Robert Bosch GmbH, "CAN with Flexible Data-Rate," in *13th International CAN Conference*, Neustadt an der Haardt, Rhineland-Palatinate, Germany, 2012.

[3] A. Marino and J. Schmalzel, "Controller Area Network for In-Vehicle Law Enforcement Applications," in *IEEE Sensors Applications Symposium*, San Diego, California, USA, 2007.

[4] M. Laifenfeld and T. Philosof, "Wireless Controller Area Network For In-Vehicle Communication," in *IEEE 28-th Convention of Electrical and Electronics Engineers in Israel*, Hertzelia, Israel, 2014.

[5] K. Kalaiyarasu and C. Carthikeyan, "Design of an Automotive Safety System using Controller Area Network," in *International Conference on Robotics, Automation, Control and Embedded Systems*, Kelambakam, Chennai, India, 2015.

[6] H. Guo, J. J. Ang and Y. Wu, "Extracting Controller Area Network Data for Reliable Car Communications," IEEE, Singapore, 2009.