Controller Area Network (CAN)

EECS 461, Fall 2008*

J. A. Cook
J. S. Freudenberg

1 Introduction

Up until now, we've considered our embedded control system to be self-contained: an algorithm implemented in software resident on a single microprocessor, communicating with its environment through sensors and actuators via peripheral devices such as an analog-to-digital converter. In fact, many embedded systems are *distributed*, consisting of multiple microprocessors communicating over one or more networks to accomplish shared tasks. For example, a modern automobile may have seventy or more microprocessors communicating over several networks to manage entertainment and navigation functions, central locking mechanisms, lighting and other vehicle systems. Safety systems such as air bags employ dedicated high speed network communication, as does powertrain control for communication between, for example, the engine and transmission controllers. Figure 1 illustrates some of the networks connecting automotive embedded systems [1, 2]. Although we will consider only wired networks, "wireless" is clearly a crucial technology for everything from assisted living to national defense [4], and wireless networking is a growing area of importance to the twenty-first century automobile. Applications include toll collection, fleet vehicle management, stolen vehicle tracking, automatic collision notification and remote diagnostics. One may expect that the confluence of in-vehicle and external communication technologies will lead to new information, entertainment and safety services such as the in-vehicle display of roadway emergency warnings or even active mitigation of collisions at intersections and vehicle-to-vehicle cooperation for improvement of safety and traffic flow [3].

1.1 Open System Interconnection (OSI)

It should be obvious that if two or more microprocessors are to communicate, a standard *protocol* must exist defining how data are to be transmitted among cooperating devices. The most common protocol is TCP/IP (Transmission Control Protocol/Internet Protocol), which is used to connect hosts on the Internet. Pre-dating TCP/IP was the Open Systems Interconnection (OSI) protocol initiated in 1982 by the International Organization for Standardization (ISO 7498-1:1994(E)). The OSI protocol is sometimes referred to as the "7-layer" model because it consists of seven independent elements that describe the requirements for communication at different levels of abstraction. The seven layers are:

Application Layer: The application layer specifies how application programs access the network. Examples include email, file transfer, remote terminal access and web browsers.

Presentation Layer: The presentation layer defines things like data compression and encryption.

Session Layer: The session layer establishes, manages and terminates the connections between cooperating applications.

Transport Layer: The transport layer provides transfer of data between users and addresses issues of error control and security.

*Revised October 13, 2008.

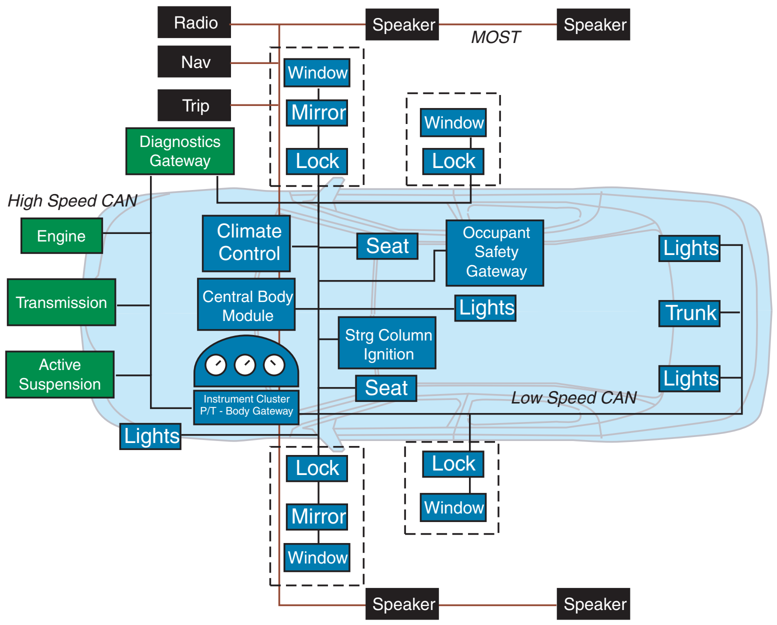

Figure 1: Typical Automotive Networks

Network Layer: The network layer performs network routing functions.

Data Link Layer: The data link layer provides synchronization and error control.

Physical Layer: The physical layer defines the physical specifications for devices on the network, including connectors, cables and electrical specifications like voltage levels.

The most commonly used network for control in automotive and manufacturing applications is the Controller Area Network, or CAN. The CAN protocol specifies rules for implementing the physical and data link layers of the OSI model in silicon to effect serial transfer of information between two or more devices.

1.2 Controller Area Network (CAN)

The Controller Area Network was developed by Robert Bosch GmbH for automotive applications in the early 1980s and publicly released in 1986. The Bosch CAN specification became an ISO standard (ISO 11898) in 1993 (CAN 2.0A), and extended in 1995 to permit longer device identifiers (CAN 2.0B) [5]. Typically, CAN interconnects a network of modules (or nodes) using two wire, twisted pair cable. Many companies implement CAN devices. In the Freescale MPC 5xx series of processors, the CAN device is called the TouCAN module; in the MPC 55xx series it's called FlexCAN. CAN is a serial, multimaster, multicast protocol, which means that when the bus is free, any node can send a message (multimaster), and all nodes may receive and act on the message (multicast). The node that initiates the message is called the transmitter; any node not sending a message is called a receiver. Messages are assigned static priorities, and a transmitting node will remain a transmitter until the bus becomes idle or until it is superseded by a node with a higher priority message through a process called arbitration. A CAN message may contain up to 8 bytes of data. A message identifier describes the data content and is used by receiving nodes to determine the destination on the network. Bit rates up to 1 Mbit/s are possible in short networks ($\leq$ 40 m). Longer network distances reduce the available bit rate (125 kbit/s at 500 m, for example). "High speed" CAN is considered to be 500 kbit/s.

1.2.1 CAN Fundamentals

The details of CAN are specified in [6]. In the following paragraphs, we will provide a brief description of how data are transmitted over CAN, how CAN messages are structured, and how transmission errors are handled. There are four types of CAN messages, or "frames:" Data Frame, Remote Frame, Error Frame and Overload Frame. The data frame is the standard CAN message, broadcasting data from the transmitter to the other nodes on the bus. A remote frame is broadcast by a transmitter to request data from a specific node. An error frame may be transmitted by any node that detects a bus error. Overload frames are used to introduce additional delay between data or remote frames. CAN 2.0A and 2.0B data frames are illustrated in Figure 2 and in Tables 1 and 2. The difference between a CAN 2.0A and a CAN 2.0B message is that CAN 2.0B supports both 11 bit (standard) and 29 bit (extended) identifiers. Standard and extended frames may exist on the same bus, and even have numerically equivalent identifiers. In this case, the standard frame will have the higher priority.

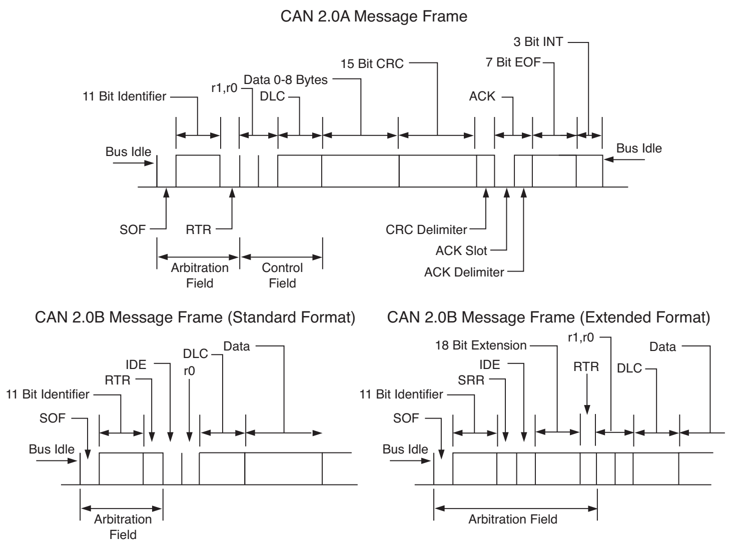

Figure 2: CAN Message Formats

Table 1: CAN 2.0A Message Frame		
Field	Length (bits)	Description
Start of Frame (SOF)	1	Must be dominant
Identifier	11	Unique identifier indicates priority
Remote Transmission Request (RTR)	1	Dominant in data frames; recessive in remote frames
Reserved	2	Must be dominant
Data Length Code (DLC)	4	Number of data bytes (0–8)
Data Field	0–8 bytes	Length determined by DLC field
Cyclic Redundancy Check (CRC)	15	
CRC Delimiter	1	Must be recessive
Acknowledge (ACK)	1	Transmitter sends recessive; receiver asserts dominant
ACK Delimiter	1	Must be recessive
End of Frame (EOF)	7	Must be recessive

Table 2: CAN 2.0B Message Frame

Field	Length (bits)	Description
Start of Frame (SOF)	1	Must be dominant
Identifier – Standard and Extended Formats	11	Unique identifier corresponds to Base ID in Extended Format
Identifier – Extended Format	29	Comprised of 11 bit Base ID and 18 bit Extended ID
Remote Transmission Request (RTR) – Standard and Extended Formats	1	Dominant in data frames; recessive in remote frames. In Standard Format, the 11 bit identifier is followed by the RTR bit.
Substitute Remote Request (SRR) – Extended Format	1	Must be recessive. SRR is transmitted in Extended Frames at the position of the RTR bit in Standard Frames. In arbitration between standard and extended frames, recessive SRR guarantees the standard message frame prevails.
IDE – Standard and Extended Frames	1	Must be recessive for Extended Format; dominant for Standard Format.
Reserved r0 – Standard Format	1	Must be dominant
Reserved r1, r0 – Extended Format	2	Must be recessive
Data Length Code (DLC)	4	Number of data bytes (0–8)
Data Field	0–8 bytes	Length determined by DLC field
Cyclic Redundancy Check (CRC)	15	
CRC Delimiter	1	Must be recessive
Acknowledge (ACK)	1	Transmitter sends recessive; receiver asserts dominant
ACK Delimiter	1	Must be recessive
End of Frame (EOF)	7	Must be recessive

1.2.2 The CAN Data Frame

The CAN data frame is composed of seven fields: Start of frame (SOF), arbitration, control, data, cyclical redundancy check (CRC), acknowledge (ACK) and end of frame (EOF). CAN message bits are referred to as “dominant” (0) or “recessive” (1). The SOF field consists of one dominant bit. All network nodes waiting to transmit synchronize with the SOF and begin transmitting at the same time. An *arbitration* scheme determines which of the nodes attempting to transmit will actually control the bus.

Arbitration

The arbitration field of the CAN message consists of an 11- or 29-bit identifier and a remote transmission (RTR) bit. The CAN arbitration scheme is called “carrier sense multiple access with collision detection” or CSMA/CD, and assures that the highest priority message is broadcast. Message priority is determined by the numerical value of the identifier in the arbitration field, with the lowest numerical value having the highest priority. Non-destructive, bit-wise arbitration resolves conflicts among competing transmitters. This means that the bus can be thought of as acting like an AND gate: If any node writes a dominant (0) bit on the bus, every node will read a dominant bit regardless of the value written by that node. Every transmitting node always reads back the bus value for each bit transmitted. If a node transmits a recessive bit and reads back a dominant bit, it immediately stops transmitting. Arbitration is illustrated in Figure 3.

The RTR bit simply distinguishes between data frames and remote frames. In data frames, the RTR bit must be dominant; in remote frames it must be recessive.

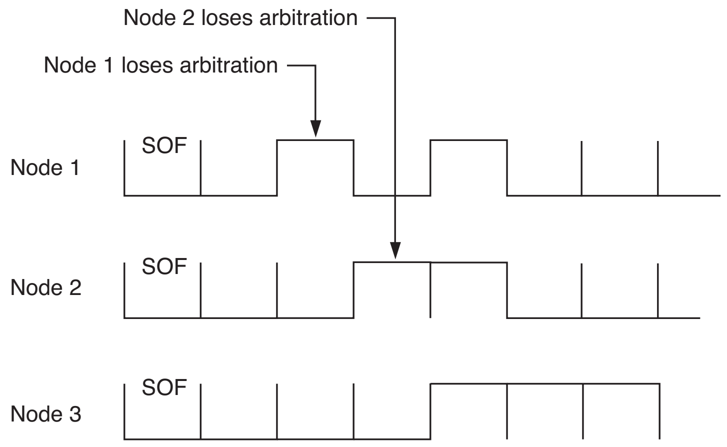

Figure 3: CAN Arbitration: Node 3 has highest, and Node 1 the lowest, priority messages.

Control and Data Fields

The control field of the data frame consists of 6 bits (of which only the lower 4 are used) that indicate the amount of data in the message. Since up to 8 bytes of data may be sent in one message, the control field may take values ranging from 000000 to 000111. The data to be transmitted are contained in the data field. The most significant bit (MSB) of a data byte is sent first.

Error Handling

CAN implements five levels of error detection. At the message level, it performs cyclic redundancy checks, frame checks and acknowledgment checks. Bit level checks consist of monitoring and stuffing.

Cyclical redundancy errors are detected using a 15 bit CRC computed by the transmitter from the message content. Each receiver accepting the message recalculates the CRC and compares it against the transmitted value. A discrepancy between the two calculations causes an error flag to be set. Frame checks that will flag an error are the detection by a receiver of an invalid bit in the CRC delimiter, ACK delimiter, EOF or 3-bit interframe space. Finally, each receiving node writes a dominant bit into the ACK slot of the message frame that is read by the transmitting node. If a message is not acknowledged (perhaps because the receiver has failed), an ACK error is flagged.

At the bit level, we have already noted that each transmitted bit is “read back” by the transmitter. If the monitored value is different than the value being sent, a bit error is detected. Additionally, bit errors are detected by stuffing: After five consecutive identical bits have been transmitted, a bit of the opposite polarity will be inserted (“stuffed”) by the transmitter into the bit stream (bits are stuffed from the SOF through the CRC field). Receivers automatically “de-stuff” the message. If any node detects six consecutive bits of the same level, a stuff error is flagged. In addition to error detection, bit stuffing assures that there are enough edges in the non-return to zero (NRZ) bit stream to maintain synchronization.

1.2.3 The CAN Error Frame

If a transmitting or receiving node detects an error, it will immediately abort the transmission and broadcast an error frame consisting of an error flag made up of six dominant bits and an error flag delimiter made up of eight recessive bits. Since this bit string violates the bit stuffing rule, all other nodes respond by transmitting error flags, too. After a sufficient number of errors are detected, a node will eventually turn itself off. Robustness, especially in manufacturing and automotive environments where CAN is prevalent, requires that the network determine whether errors are transient (due to voltage spikes, noise or some other temporary condition) or permanent failure of the node due to defective hardware. Consequently, nodes store and track the number of errors detected. A node may be in one of three modes depending on the error count: If the count in either the transmit or receive buffer of a node is greater than zero and less than 128, the node is considered “error active,” indicating that, although the node remains fully functional, at least one error has been detected. An error count between 128 and 255 puts the node in “error passive” mode. An error passive node will transmit at a slower rate by sending 8 recessive bits before transmitting again or recognizing the bus to be idle. Error counts above 255 will cause the node to enter “bus off” mode, taking itself off-line. Receive errors increment the error count by 1; transmit errors increment the count by 8. Subsequent error-free messages decrement the error count by 1. If the error count returns to zero, a node will return to normal mode. A node in the bus off condition may become error active after 128 occurrences of 11 consecutive recessive bits have been monitored. A message is considered valid by the transmitter if there is no error until the EOF. Corrupted messages are automatically retransmitted as soon as the bus is idle.

1.2.4 The CAN Remote Frame

A node that requires data from another node on the network can request a transmission by sending a Remote Frame. For example, the microprocessor controlling the central locking on your car may need to know the state of the transmission gear selector from the powertrain controller (is the car in “park?”). A remote frame is the same as a data frame, without the data field (with the RTR bit recessive).

1.2.5 Overload Frames and Interframe Space

If a CAN node receives messages faster than it can process them, then an Overload Frame will be generated to provide extra time between successive Data or Remote frames. Similar to an Error Frame, the Overload Frame has two fields: an overload flag consisting of six dominant bits, and an overload delimiter consisting of eight recessive bits. Unlike error frames, error counters are not incremented.

The Interframe Space consists of a three recessive bit Intermission and the bus idle time between Data or Remote Frames. During the intermission, no node is permitted to initiate a transmission (if a dominant bit is detected during the Intermission, an Overload Frame will be generated). The bus idle time lasts until a node has something to transmit, at which time the detection of a dominant bit on the bus signals a SOF.

1.3 Bus Loading

CAN provides a robust, simple and flexible network solution for manufacturing, automotive and many other applications. The major drawback to CAN is that message latency is non-determinant (due to the existence of Error Frames, Overload Frames and retransmissions), and latency increases with the amount of traffic on the bus. In general, bus utilization should not exceed 30% of the bus capacity to assure that low priority messages do not experience unacceptable delay. Bus utilization is defined as total bit consumption / total bits available, and is calculated as follows:

Step 1: Choose a time unit $\geq$ the slowest fixed periodic message on network (usually 1 second).

Step 2: Identify all periodic messages.

Step 3: For each of these messages approximate the total bit size of the message by adding 47 bits to the size of each data field (SOF + Arbitration + RTR + Control + CRC + Acknowledgment + EOF + Interframe Space = 1 + 11 + 1 + 6 + 16 + 2 + 7 + 3 = 47 bits).

Step 4: Calculate the message bits consumed by multiplying the message bit size by the number of transmissions performed in one time unit.

Step 5: Sum all the message bits consumed to estimate the total periodic bits consumed. Multiply this number by a safety factor of 1.1 to account for worst case traffic.

Step 6: Finally, divide the total periodic bits consumed by the total bits available (for example, 125 kbps or 500 kbps multiplied by the time unit) to arrive at the estimated bandwidth consumption percentage for the network.

Table 2: Bus Loading Example

Message	Data (bytes)	Message Size (bits)	Rate and Period	Message Bits Consumed
MsgA	0	47	10 trx/s: 100 ms	$10 \cdot 47 = 470$ bps
MsgB	5	$5 \cdot 8 + 47 = 87$	2 trx/s: 500 ms	$87 \cdot 2 = 174$ bps
MsgC	8	$8 \cdot 8 + 47 = 111$	1 trx/s: 1 s	$111 \cdot 1 = 111$ bps
...	...	...	...	...
			Total periodic bits consumed	10000 bps
Total Bits Consumed = 1.1·(Total Periodic Bits Consumed) = 11000 bps Bandwidth Consumption = 11000/125000 = 8.8%				

1.4 Time-triggered Protocols

For real-time control over a network it may be advisable to implement a communication protocol that guarantees that messages meet timing deadlines regardless of the load on the bus. One such protocol that retains the CAN data link layer protocol is "time-triggered CAN," or TTCAN (ISO 11898-4) [7]. A TTCAN message frame incorporates two types of "time windows": exclusive time windows, and arbitrating time windows. Exclusive time windows are assigned to specific messages that are transmitted periodically. Thus, exclusive window messages do not compete for bus access. Arbitrating windows are used for messages that are not time critical. Arbitrating window messages, like normal CAN messages, compete for bus access based on priority through arbitration. Time-triggered CAN requires the existence of a "master node" that periodically broadcasts its time (referred to as global time) in a reference message. For fault tolerance, there must be multiple potential master nodes on the network. If the master node fails (detected by the absence of a reference message), the other potential masters compete for the bus by arbitration, with the highest priority node becoming the new master and broadcasting reference messages. Time-triggered CAN does not re-broadcast corrupted messages, nor does it invoke Error Frames.

A competitive protocol to TTCAN is FlexRay, developed by a consortium of automotive manufacturers and suppliers [8]. The FlexRay communication frame consists of periodically triggered "static" and "dynamic" parts. The static segment is made up of identical length time slots assigned to connected nodes. Each node transmits its messages synchronously in its reserved slot. The static segment also transmits a "synch" frame to provide a global timebase for the network. Unlike CAN, there is no arbitration for the bus. The dynamic segment is essentially a "polling" mechanism wherein each node is given the opportunity to put an event-triggered or asynchronous message on the bus in priority order using a "mini-slotting" timing mechanism. That is, the slot counter increments through each identifier, but whereas in the static segment the counter increments at a periodic rate whether or not there is a message in each slot, in the dynamic segment the counter waits only a brief period for each node to request transmission. If there is no transmission request for a message with a particular slot identifier, the counter continues to increment; if a transmission request occurs, the counter idles until the transmission is complete.

For redundant fault tolerance, FlexRay nodes may be connected to two buses, or channels simultaneously.

References

[1] Nicolas Navet, Yeqiong Song, Françoise Simonot-Lion, and Cédric Wilwert, "Trends in Automotive Communication Systems," Proceedings of the IEEE, Vol. 93, NO. 6, June 2005 Page(s):1204 - 1223

[2] Gabriel Leen and Donal Heffernan, "Expanding Automotive Electronic Systems," IEEE Computer Volume 35, Issue 1, Jan. 2002 Page(s):88 - 93

[3] Jeffrey Cook, Ilya Kolmanovsky, David McNamara, Edward Nelson and K. Venkatesh Prasad, “Control, Computing and Communications: Technologies for the Twenty-first Century Model T,” Proceedings of the IEEE, February 2007.

[4] J.M. Eklund, J. Sprinkle, S. Sastry and T.R. Hansen, “Information Technology for Assisted Living at Home: building a wireless infrastructure for assisted living,”IEEE-EMBS 2005 27th Annual International Conference of the Engineering in Medicine and Biology Society, Sept. 2005 page(s): 3931- 3934.

[5] CAN in Automation Website, http://www.can-cia.org, viewed September 2006.

[6] Robert Bosch GmbH, “CAN Specification 2.0,” 1991.

[7] Holger Zeltwanger, “Time-Triggered Communication on CAN,” SAE Paper 2002-01-0437, 2002.

[8] Thomas Fuehrer, Robert Hugel, Florian Hartwich and Harald Weiler, “FlexRay - The Communication System for Future Control Systems in Vehicles,” SAE Paper 2003-01-0110, 2003.