September 5, 1997

NMEA 2000 & the Controller Area Network (CAN)

Frank Cassidy, Chairman NMEA Standards Committee

Preview:
An Introduction to NMEA 2000 in the May/June 1997 issue of Marine Electronics provided an overview of the network and describes NMEA 2000 as containing "…the requirements for the implementation of a serial-data communications network to interconnect marine electronic equipment onboard vessels. The network operates in a Carrier Sense/Multiple Access/Collision Arbitration mode, is multi-master and self-configuring, there is no central network controller."

To provide structure to the development of the NMEA 2000 standard it is being defined in layers according to the International Standards Organization Open Systems Interconnect (ISO/OSI) model. Of the seven layers defined by the model, the NMEA 2000 standard will specify the following:

- **Layer 1. The Physical Layer** - For the electrical and mechanical definitions of the network
- **Layer 2. Data Link Layer**, with Media Access Control (MAC) and Logical Link Control (LLC) sub-layers - For network access protocol, timing, error detection, and network management
- **Layer 3. The Network Layer** - For interconnecting multiple network segments
- **Layer 7. The Application Layer** - For the actual definition of the data transmitted on the network

This series of articles in Marine Electronics will provide an increasing level of detail about the NMEA 2000 standard. Because it is fundamental to the operation of the network and impacts all of the other layers, this second article will describe the Controller Area Network (CAN), the basis for the Media Access Control portion of the Data Link Layer. It is not possible to define any of the other layers until the power that is provided by CAN is understood, so let's start there.

Multi-Master
NMEA 2000 is a multi-master serial data network wired as a bus, with all inputs and outputs of all devices connected together.

What does multi-master mean? What we really want for NMEA 2000 is a network architecture that is not dependent on a central network controller. How do we get this on

a single channel with all devices connected together? To get this we have to distribute the control equally among all of the network devices.

This means that each device must be programmed with a set of rules, or protocol, that will: allow each device its fair share of network bandwidth for sending its messages, allow it to identify and acknowledge messages, provide the capability to detect errors and retransmit data or shut down automatically if necessary. By rigidly following this set of rules each device is its own master of the network and is only dependent on the other devices to follow the same rules.

The CAN Solution

In the mid-eighties Robert Bosch GmbH and Intel collaborated on the specification and development of integrated circuits (IC) for a device that would provide a serial communications protocol to support distributed real-time control applications. The objective was to provide a robust solution for automotive applications that include high-speed networking as well low-cost wire multiplexing. CAN was originally intended for real-time engine and transmission control, anti-skid breaking systems, and to replace the wiring of body components.

The current CAN specification is Version 2.0 (1991, Robert Bosch GmbH) and is an open document and freely available on the Internet, it is also documented in ISO-11898.

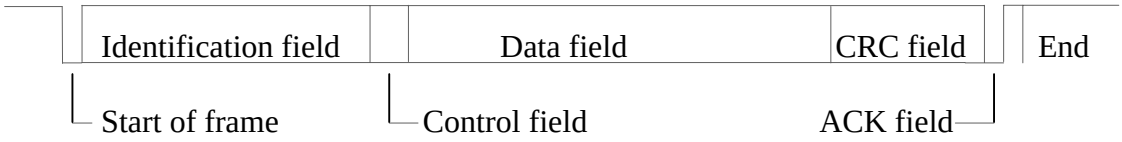
While automotive application dominate, and have the potential to drive prices down with high production volumes, CAN has exploded into other applications in all areas of industrial control. CAN is supported by virtually every major integrated circuit manufacturer and devices are available as standalone CAN controllers, and embedded with microcontrollers such as the 8051, 68HC05, and 87C196 families. Manufacturers that include CAN in their standard product line include:

- Hitachi
- Intel
- Mitsubishi
- Motorola
- National Semiconductor
- NEC
- Phillips-Signetics
- SGS-Thompson
- Siemens
- Texas Instruments

The utility of CAN, and the power it adds to a microcontroller-based system, may in some way be thought of as analogous to the standard Universal Asynchronous Receiver/Transmitter (UART), and it may soon be as commonplace.

What is CAN?
CAN is a serial communications protocol with high data security implemented by standard integrated circuits from numerous suppliers. It provides a transfer protocol for serial communications that includes all bit timing, frame formatting, message identification, data transmission, acknowledgement, and error checking.

The serial data frame used by CAN has an identification field and from zero to eight data bytes. In addition, as shown below, the frame contains start of frame and end of frame bits, frame control bits, error detection fields, acknowledgement bits, plus a number of fixed format and reserved bits. All device on the bus are synchronized by the leading edge of the frame start and are resynchronized by the edges of the data bits in the frame. In CAN it is required that all devices sample the same bit in the frame at the same time.



CAN defines four types of frames:
- Data Frame, used to transmit data
- Remote Frame, used to request data
- Error Frame, transmitted when errors are detected
- Overload Frame, transmitted to delay additional Data or Remote frames

The frame type is indicated by the settings of certain bits in the frame and each frame is used for specific purposes as implied above. Error frames and Overload frames are transmitted automatically by CAN when conditions warrant, but the use and content of Data and Remote frames is under the control of the designer.

The format of the CAN frame is rigidly defined, and except for the way certain fields are used, the ability to vary the length of the data field, and to program the bit rate (practical bit rates are within the range of 5 Kbps to 1 Mbps), there is no freedom for modification.

A notable exception to this statement is that CAN is available in two formats. The Standard Format (Part A of the CAN specification) utilizes an 11-bit Identification field, and the Extended Format (Part B) extends the 11-bits with 18 additional bits for a total Identification field length of 29 bits. Control bits are used to indicate which format is being transmitted such that it is possible for both formats to be used on the same bus without serious interference. NMEA 2000 is defined using only the 29-bit identifier, but with some limitations, non-NMEA 2000 frames with 11-bit identifiers may exist on the same network.

So much for the mechanics of CAN, the real utility of CAN is described in the next section.

CAN Features

CAN produces the bit pattern of the serial data as described above, this is necessary, but is also the easy part. The real utility of CAN is contained in the powerful suite of features that manage the communications on the bus. CAN automatically handles the entire data transfer function through the following steps:

- Bus access
- Priority-based bus contention resolution
- Data transmission
- Error detection
- Automatic re-transmission of failed messages
- Message delivery acknowledgement
- Automatic shutdown of failed nodes

Like Ethernet, CAN operates in the Carrier-Sense/Multiple Access mode. What this means is that when there is a frame to be sent CAN listens on the bus (carrier-sense), and if the bus is not busy data transmission may proceed. It is multiple access because if every device follows the rules there can be many devices sharing the same bus. The difference between CAN and the Ethernet approach occurs when two devices simultaneously determine that the bus is not busy, and both start to transmit. This leads to bus contention. With Ethernet there is a data collision and both devices stop transmitting and try again later, valuable time on the bus is lost during the collision and the net bandwidth is reduced. CAN handles bus contention in a way that prevents loss of bus bandwidth. When bus contention occurs CAN arbitrates bus access on a bit-by-bit basis, the device with the highest priority prevails and continues to transmit data, the device with lower priority tries again later.

To support bus arbitration CAN requires that the physical layer of the network transmit bits that are either Dominant (Logic “0” as it turns out) or Recessive (Logic “1”). Simply, this means that the bus connections operate as Wire-AND circuits where a single device transmitting a “0” will dominate over all other devices transmitting “1”s. CAN performs this bit-by-bit arbitration using only the bits in the Identification field; this means that the network design must require the Identification field be unique for each frame, and that the importance, or priority, of the message be implied by the numerical value of the Identification field.

Data Delivery

The “useable” portions of the CAN frame are the Identification and Data fields, all of the other bits have special purposes (e.g., error detection, acknowledgement) and are not available to the designer. While it is called an “identification” field, “arbitration” field might have been a better name. These 29 bits (11-bits for the Standard format) could be used to transmit data, and for that matter some of the “data” bits could be used to “identify” the data or the transmitting node. The point is that CAN specifies the field format rigidly, but there is considerable flexibility in how these fields are used.

However the bits are utilized, within a CAN frame there is a limited number of bits for transferring data (8-bytes if we stick to the defined Data fields, and only a few more if we steal some of the Identification field bits). There will certainly be data that exceeds the limits of a single CAN frame. The actual use of these fields, and provisions for sending multi-frame messages, are tasks performed by the other part of the Data Link Layer, the Logical Link Control (LLC) sub-layer, and will be covered in a future article.

Error detection & Confinement

The powerful error detection techniques provided by CAN offer a data delivery capability with less than one chance in 20 billion that an error will not be detected. All nodes perform error detection and any node detecting an error generates an Error frame, Error frames may interrupt Data frames. Once an error is flagged the transmission is aborted and the data retransmitted automatically by CAN.

There are five levels of error detection applied to every frame transmitted:

- Bit test: The CAN device that is transmitting tests the bus for each bit as it is transmitted.
- Cyclic Redundancy Check (CRC): A 15-bit CRC sequence transmitted in the frame is calculated by the transmitter and checked by the receiver.
- Encoding test: CAN encodes the transmitted bits using bit-stuffing where if more than five bits of the same polarity are to be transmitted the sixth-bit is automatically reversed (and restored by the receiver). Besides being part of the error checking this technique assures an adequate number of bit edges to maintain synchronization on the bus.
- Frame check: Specified bits within CAN frames are fixed format bits with defined polarities, CAN receivers automatically check for these formats.
- Acknowledgement: Special acknowledgement bits follows the CRC field in the CAN frame that are to be set by receiving devices if the CRC is correctly received. The transmitter tests to see if these bits are set.

Errors detected during transmission and during reception are counted by the CAN device and kept track of. If the transmit or receive error count reaches a set value the device is automatically designated as Error Passive and certain restrictions are placed on its operation. As future frames are transmitted or received by this device without errors the error counts are decremented and normal operation is allowed again. If the error count continues to increase to a higher set value the device is declared Bus Off and the drivers are deactivated. Bus Off devices are allowed to return to the Error Passive state after a lengthy number of inactive gaps are detected on be bus, which will vary with bus loading, and then must recover from the Error Passive state in the usual way.

CAN Devices

A sampling of CAN devices from various manufacturers is provided in Table 1. Some devices are stand-alone CAN controllers but most are embedded in 8- or 16-bit microcontrollers along with other system features. The list of devices is a moving target, with an upward trend, and it is difficult to compile a listing without leaving out devices.

Beyond the microcontroller related configuration, and the availability of both Part A and Part B CAN devices, ICs differ in other important respects so it important to know the device and the application before making a selection. The CAN specification provides for message filtering by the CAN device to reduce the interrupts to a connected processor. Not an insignificant issue in some systems, a processor connected to a bus operating at 25% capacity at 500 Kbps could receive an interrupt every millisecond. Filtering is applied only to the Identification field, and optional programmable filter masks allow groups of identifiers to be accepted. This, plus other features (e.g., sleep modes, amount of onboard RAM) vary between devices.

Table 1 - CAN Devices

Hitachi	HCAN-1		National	COP888
Intel	AN82527		National	COP87L84
Intel	AN87C196		National	COP840
Mitsubishi	M37630		National	COP984
Motorola	MC68HC05		Philips	P82C150
Motorola	MC68HC705		Philips	PCA82C200
Motorola	MC68HC08		Philips	PCA82C250
Motorola	MPC500		Philips	PCA82C251
Motorola	MC68376		Philips	XA-C3 (SJA1000)
NEC	72005		SGS-Thompson	ST10F167
National	COP884		Siemens	SAB-C515 (8051)
National	COP888		Siemens	SAB-C167
National	COP684/688/689		Texas Instruments	TMS370x08D55
National	COP889			

To be continued …

Subsequent articles will describe the use of the CAN Identification field and Data fields, along with actual data descriptions and information on the physical layer. Stay tuned, better yet send comments and participate in the process …. the address is nmea@coastalnet.com.