Controller area network

Controller–area network (**CAN** or **CAN-bus**) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other within a vehicle without a host computer.

CAN is a message based protocol, designed specifically for automotive applications but now also used in other areas such as industrial automation and medical equipment.

Development of the CAN-bus started originally in 1983 at Robert Bosch GmbH.[1] The protocol was officially released in 1986 at the Society of Automotive Engineers (SAE) congress in Detroit, Michigan. The first CAN controller chips, produced by Intel and Philips, came on the market in 1987. Bosch published the CAN 2.0 specification in 1991.

CAN is one of five protocols used in the OBD-II vehicle diagnostics standard. The OBD standard is mandatory for all cars and light trucks sold in the United States since 1996, and the EOBD standard, mandatory for all petrol vehicles sold in the European Union since 2001 and all diesel vehicles since 2004.[2]

Applications

Automotive

A modern automobile may have as many as 70 electronic control units (ECU) for various subsystems[3] . Typically the biggest processor is the engine control unit, which is also referred to as "ECU" in the context of automobiles; others are used for transmission, airbags, antilock braking, cruise control, audio systems, windows, doors, mirror adjustment, etc. Some of these form independent subsystems, but communications among others are essential. A subsystem may need to control actuators or receive feedback from sensors. The CAN standard was devised to fill this need.

The CAN bus may be used in vehicles to connect engine control unit and transmission, or (on a different bus) to connect the door locks, climate control, seat control, etc. Today the CAN bus is also used as a fieldbus in general automation environments, primarily due to the low cost of some CAN Controllers and processors.

Bosch holds patents on the technology, and manufacturers of CAN-compatible microprocessors pay license fees to Bosch, which are normally passed on to the customer in the price of the chip. Manufacturers of products with custom ASICs or FPGAs containing CAN-compatible modules may need to pay a fee for the *CAN Protocol License* [4].

Technology

CAN is a multi-master broadcast serial bus standard for connecting electronic control units (ECUs).

Each node is able to send and receive messages, but not simultaneously. A message consists primarily of an ID — usually chosen to identify the message-type or sender — and up to eight data bytes. It is transmitted serially onto the bus. This signal pattern is encoded in NRZ and is sensed by all nodes.

The devices that are connected by a CAN network are typically sensors, actuators, and other control devices. These devices are not connected directly to the bus, but through a host processor and a CAN controller.

If the bus is free, any node may begin to transmit. If two or more nodes begin sending messages at the same time, the message with the more dominant ID (which has more dominant bits, i.e., zeroes) will overwrite other nodes' less dominant IDs, so that eventually (after this arbitration on the ID) only the dominant message remains and is received by all nodes.

Each node requires a

- **host processor**
  - The host processor decides what received messages mean and which messages it wants to transmit itself.

- Sensors, actuators and control devices can be connected to the host processor.
- **CAN controller** (hardware with a synchronous clock).
  - *Receiving*: the CAN controller stores received bits serially from the bus until an entire message is available, which can then be fetched by the host processor (usually after the CAN controller has triggered an interrupt).
  - *Sending*: the host processor stores its transmit messages to a CAN controller, which transmits the bits serially onto the bus.
- **Transceiver** (possibly integrated into the CAN controller)
  - *Receiving*: it adapts signal levels from the bus to levels that the CAN controller expects and has protective circuitry that protects the CAN controller.
  - *Sending*: it converts the transmit-bit signal received from the CAN controller into a signal that is sent onto the bus.

Bit rates up to 1 Mbit/s are possible at network lengths below 40 m. Decreasing the bit rate allows longer network distances (e.g., 500 m at 125 kbit/s).

The CAN data link layer protocol is standardized in ISO 11898-1 (2003). This standard describes mainly the data link layer — composed of the logical link control (LLC) sublayer and the media access control (MAC) sublayer — and some aspects of the physical layer of the OSI reference model. All the other protocol layers are the network designer's choice.

Data transmission

CAN features an automatic 'arbitration free' transmission. A CAN message that is transmitted with highest priority will 'win' the arbitration, and the node transmitting the lower priority message will sense this and back off and wait.

This is achieved by CAN transmitting data through a binary model of "dominant" bits and "recessive" bits where dominant is a logical 0 and recessive is a logical 1. This means open collector, or 'wired or' physical implementation of the bus (but since dominant is 0 this is sometimes referred to as wired-AND). If one node transmits a dominant bit and another node transmits a recessive bit then the dominant bit "wins" (a logical AND between the two).

Truth tables for dominant/recessive and logical AND

Bus state with two nodes transmitting

	dominant	**recessive**
dominant	dominant	dominant
recessive	dominant	recessive

Logical AND

	0	**1**
0	0	0
1	0	1

So, if you are transmitting a recessive bit, and someone sends a dominant bit, you see a dominant bit, and you know there was a collision. (All other collisions are invisible.) A dominant bit is asserted by creating a voltage across the wires while a recessive bit is simply not asserted on the bus. If any node sets a voltage difference, all nodes will see it. Thus there is no delay to the higher priority messages, and the node transmitting the lower priority message automatically attempts to re-transmit 6 bit clocks after the end of the dominant message.

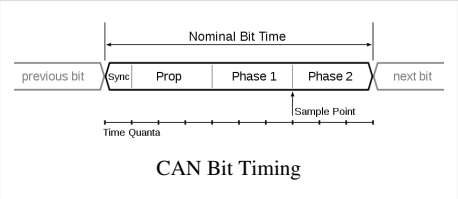
When used with a differential bus, a Carrier Sense Multiple Access/Bitwise Arbitration (CSMA/BA) scheme is often implemented: if two or more devices start transmitting at the same time, there is a priority based arbitration scheme to decide which one will be granted permission to continue transmitting. The CAN solution to this is prioritised arbitration (and for the dominant message delay free), making CAN very suitable for real time prioritised communications systems.

During arbitration, each transmitting node monitors the bus state and compares the received bit with the transmitted bit. If a dominant bit is received when a recessive bit is transmitted then the node stops transmitting (i.e., it lost arbitration). Arbitration is performed during the transmission of the identifier field. Each node starting to transmit at the same time sends an ID with dominant as binary 0, starting from the high bit. As soon as their ID is a larger number (lower priority) they'll be sending 1 (recessive) and see 0 (dominant), so they back off. At the end of ID transmission, all nodes but one have backed off, and the highest priority message gets through unimpeded.

For example, consider an 11-bit ID CAN network, with two nodes with ID's of 15 (binary representation, 00000001111) and 16 (binary representation 00000010000). If these two nodes transmit at the same time, each will transmit the first 6 zeros of their ID with no arbitration decision being made. When the 7th bit is transmitted, the node with the ID of 16 transmit a 1 (recessive) for its ID, and the node with the ID of 15 transmits a 0 (dominant) for its ID. When this happens, the node with the ID of 16 will realize that it lost its arbitration, and allow the node with ID of 15 to continue its transmission. This ensures that the node with the lower bit value will always win the arbitration.

Bit timing

Each node in a CAN network has its own clock, and no clock is sent during data transmission. Synchronization is done by dividing each bit of the frame into a number of segments: Synchronization, Propagation, Phase 1 and Phase 2. The Length of each *phase* segment can be adjusted based on network and node conditions. The sample point falls between Phase Buffer Segment 1 and Phase Buffer Segment 2, which helps facilitate continuous synchronization. Continuous synchronization in turn enables the receiver to be able to properly read the messages.



CAN Bit Timing

Layers

Based on levels of abstraction, the structure of the CAN protocol can be described in terms of the following layers:

- Application Layer
- Object Layer
  - Message Filtering
  - Message and Status Handling
- Transfer Layer

  The Transfer Layer represents the kernel of the CAN protocol. It presents messages received to the object layer and accepts messages to be transmitted from the object layer. The transfer layer is responsible for bit timing and synchronization, message framing, arbitration, acknowledgment, error detection and signaling, and fault confinement. It performs:
  - Fault Confinement
  - Error Detection
  - Message Validation
  - Acknowledgment
  - Arbitration
  - Message Framing
  - Transfer Rate and Timing
  - Information Routing

- Physical Layer
  The physical layer defines how the signals are actually transmitted. Tasks include:
  - Signal Level and Bit Representation
  - Transmission Medium

Frames

A CAN network can be configured to work with two different message (or "frame") formats: the standard or base frame format (or CAN 2.0 A), and the extended frame format (or CAN 2.0 B). The only difference between the two formats is that the “CAN base frame” supports a length of 11 bits for the identifier, and the “CAN extended frame” supports a length of 29 bits for the identifier, made up of the 11-bit identifier (“base identifier”) and an 18-bit extension (“identifier extension”). The distinction between CAN base frame format and CAN extended frame format is made by using the IDE bit, which is transmitted as dominant in case of an 11-bit frame, and transmitted as recessive in case of a 29-bit frame. CAN controllers that support extended frame format messages are also able to send and receive messages in CAN base frame format. All frames begin with a start-of-frame (SOF) bit that denotes the start of the frame transmission.

CAN has four frame types:

- Data frame: a frame containing node data for transmission
- Remote frame: a frame requesting the transmission of a specific identifier
- Error frame: a frame transmitted by any node detecting an error
- Overload frame: a frame to inject a delay between data and/or remote frame

Data frame

The data frame is the only frame for actual data transmission. There are two message formats:

- Base frame format: with 11 identifier bits
- Extended frame format: with 29 identifier bits

The CAN standard requires the implementation must accept the base frame format and may accept the extended frame format, but must tolerate the extended frame format.

Base frame format

The frame format is as follows:

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier	11	A (unique) identifier for the data
Remote transmission request (RTR)	1	Dominant (0) (see Remote Frame below)
Identifier extension bit (IDE)	1	Must be dominant (0)Optional
Reserved bit (r0)	1	Reserved bit (it must be set to dominant (0), but accepted as either dominant or recessive)
Data length code (DLC)*	4	Number of bytes of data (0-8 bytes)
Data field	0-8 bytes	Data to be transmitted (length dictated by DLC field)
CRC	15	Cyclic Redundancy Check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)

End-of-frame (EOF)	7	Must be recessive (1)

One restriction placed on the identifier is that the first seven bits cannot be all recessive bits. (I.e., the 16 identifiers 1111111xxxx are invalid.)

Extended frame format

The frame format is as follows:

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier A	11	First part of the (unique) identifier for the data
Substitute remote request (SRR)	1	Must be recessive (1)Optional
Identifier extension bit (IDE)	1	Must be recessive (1)Optional
Identifier B	18	Second part of the (unique) identifier for the data
Remote transmission request (RTR)	1	Must be dominant (0)
Reserved bits (r0, r1)	2	Reserved bits (it must be set dominant (0), but accepted as either dominant or recessive)
Data length code (DLC)*	4	Number of bytes of data (0-8 bytes)
Data field	0-8 bytes	Data to be transmitted (length dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

The two identifier fields (A & B) combine to form a 29-bit identifier.

* It is physically possible for a value between 9-15 to be transmitted in the 4-bit DLC, although the data is still limited to 8 bytes. Certain controllers allow the transmission and/or reception of a DLC greater than 8, but the actual data length is always limited to 8 bytes.

Remote frame

- Generally data transmission is performed on an autonomous basis with the data source node (e.g. a sensor) sending out a Data Frame. It is also possible, however, for a destination node to request the data from the source by sending a Remote Frame.
- There are 2 differences between a Data Frame and a Remote Frame. Firstly the RTR-bit is transmitted as a dominant bit in the Data Frame and secondly in the Remote Frame there is no Data Field.

i.e.

RTR = 0 ; DOMINANT in data frame

RTR = 1 ; RECESSIVE in remote frame

In the very unlikely event of a Data Frame and a Remote Frame with the same identifier being transmitted at the same time, the Data Frame wins arbitration due to the dominant RTR bit following the identifier. In this way, the node that transmitted the Remote Frame receives the desired data immediately.

Error frame

Error frame consists of two different fields

The first field is given by the superposition of ERROR FLAGS contributed from different stations. The following second field is the ERROR DELIMITER.

There are two types of error flags

Active Error Flag

: Transmitted by a node detecting an error on the network that is in error state "error active".

Passive Error Flag

: Transmitted by a node detecting an active error frame on the network that is in error state "error passive".

Overload frame

The overload frame contains the two bit fields Overload Flag and Overload Delimiter. There are two kinds of overload conditions that can lead to the transmission of an overload flag:

1. The internal conditions of a receiver, which requires a delay of the next data frame or remote frame.
2. Detection of a dominant bit during intermission.

The start of an overload frame due to case 1 is only allowed to be started at the first bit time of an expected intermission, whereas overload frames due to case 2 start one bit after detecting the dominant bit. Overload Flag consists of six dominant bits. The overall form corresponds to that of the active error flag. The overload flag's form destroys the fixed form of the intermission field. As a consequence, all other stations also detect an overload condition and on their part start transmission of an overload flag. Overload Delimiter consists of eight recessive bits. The overload delimiter is of the same form as the error delimiter.

Interframe spacing

Data frames and remote frames are separated from preceding frames by a bit field called interframe space. Overload frames and error frames are not preceded by an interframe space and multiple overload frames are not separated by an interframe space. Interframe space contains the bit fields intermission and bus idle and, for error passive stations, which have been transmitter of the previous message, suspend transmission.

Bit stuffing

In CAN frames, a bit of opposite polarity is inserted after five consecutive bits of the same polarity. This practice is called bit stuffing, and is due to the "Non Return to Zero" (NRZ) coding adopted. The "stuffed" data frames are destuffed by the receiver. Since bit stuffing is used, six consecutive bits of the same type (111111 or 000000) are considered an error.

Bit stuffing implies that sent data frames could be larger than one would expect by simply enumerating the bits shown in the tables above.

Standards

There are several CAN physical layer standards:

- **ISO 11898-1**: CAN Data Link Layer and Physical Signalling
- **ISO 11898-2**: CAN High-Speed Medium Access Unit
- **ISO 11898-3**: CAN Low-Speed, Fault-Tolerant, Medium-Dependent Interface
- **ISO 11898-4**: CAN Time-Triggered Communication
- **ISO 11898-5**: CAN High-Speed Medium Access Unit with Low-Power Mode
- **ISO 11992-1**: CAN fault-tolerant for truck/trailer communication
- **ISO 11783-2**: 250 kbit/s, Agricultural Standard
- **SAE J1939-11**: 250 kbit/s, Shielded Twisted Pair (STP)
- **SAE J1939-15**: 250 kbit/s, UnShielded Twisted Pair (UTP) (reduced layer)
- **SAE J2411**: Single-wire CAN (SWC)

ISO 11898-2 uses a two-wire balanced signaling scheme. It is the most used physical layer in car powertrain applications and industrial control networks.

ISO 11898-4 standard defines the time-triggered communication on CAN (TTCAN). It is based on the CAN data link layer protocol providing a system clock for the scheduling of messages.

SAE J1939 standard uses a two-wire twisted pair, -11 has a shield around the pair while -15 does not. SAE 1939 is widely used in agricultural & construction equipment.

ISO 11783-2 uses four unshielded twisted wires; two for CAN and two for terminating bias circuit (TBC) power and ground. This bus is used on agricultural tractors. This bus is intended to provide interconnectivity with any implementation adhering to the standard.

Higher layer implementations

As the CAN standard does not include tasks of application layer protocols, such as flow control, device addressing, and transportation of data blocks larger than one message, many implementations of higher layer protocols were created. Among these are **DeviceNet**, **CANopen**, **SDS (Smart Distributed System)**, **CANaerospace**, **J1939**, SmartCraft, **NMEA 2000**, **CAN Kingdom**, **SafetyBUS p**, **EnergyBus** and **MilCAN**.

An ARINC technical working group develops the ARINC 825 standard with special requirements for the aviation industry.

Security

CAN is a low-level protocol, and does not support any security features intrinsically. Applications are expected to deploy their own security mechanisms, e.g., to authenticate each other. Failure to do so may result in various sorts of attacks, if the opponent manages to insert messages on the bus.

See also

- FlexCAN - An alternative implementation.
- CANopen - A higher-layer protocol.
- Local Interconnect Network - A low cost alternative.
- FlexRay - A possible future direction
- GMLAN - A customized version by GM
- Socketcan - a set of open source CAN drivers and a networking stack contributed by Volkswagen Research to the Linux kernel.
- OSEK

- List of network buses

External links

- Bosch specification [5] (old document — slightly ambiguous/unclear in some points, superseded by the standard ISO 11898 [6])
- Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised [7]
- Pinouts for common CAN-bus connectors [8]
- Independent discussion platform CANLIST [9]

References

[1] "CAN History" (http://www.can-cia.de/index.php?id=161). CAN in Automation. .

[2] *Building Adapter for Vehicle On-board Diagnostic* (http://www.obddiag.net/adapter.html), obddiag.net, accessed 2009-09-09

[3] Comparison of Event-Triggered and Time-Triggered Concepts with Regard to Distributed Control Systems A. Albert, Robert Bosch GmbH Embedded World, 2004, Nürnberg

[4] http://www.semiconductors.bosch.de/en/20/can/2-license.asp

[5] http://www.semiconductors.bosch.de/pdf/can2spec.pdf

[6] http://www.iso.org/iso/search.htm?qt=Controller+Area+Network&searchSubmit=Search&sort=rel&type=simple&published=true

[7] http://www.springerlink.com/content/8n32720737877071/

[8] http://www.interfacebus.com/Can_Bus_Connector_Pinout.html

[9] http://www.canlist.org/

Article Sources and Contributors

Controller area network *Source*: http://en.wikipedia.org/w/index.php?oldid=369893298 *Contributors*: AGlossop, Accounting4Taste, Aechols, Agl, Ahoerstemeier, Ajn1, Akadruid, Albramallo, Alex.muller, Alexandreag, Allen Moore, Anapaulasag, Andre maier, Andy Dingley, Armistej, Attilios, Automotive joe, Axlq, Benburleson, Betsytimmer, Biker Biker, Bobblewik, Bsilverthorn, CAkira, Calltech, CanisRufus, Cburnett, Chrisbolt, Colin Marquardt, D rock naut, Deeptrivia, DerHexer, Dmorr, Dpotop, Drunken Pirate, Egil, Electron9, Eliyahu S, Finlay McWalter, Fleminra, Ft1, Fxhomie, Gabriel Kielland, Gejigeji, Gregmelson, Guy Harris, GyroMagician, Harriv, Heron, Hidayat ullah, Hondrej, Hopp, IamNotU, IanOsgood, Idonra, Invernizzi.l, Iviney, JacobBramley, Jidan, Jni, JohnCD, Johnny.cespedes, JonHarder, Jredders, JuergenKlueser, Katharineamy, Kdub432, Kingdon, KnowledgeOfSelf, Kramer-Wolf, Kris iyer, Lakshmank85, Langerwolfgang, MC MasterChef, MStock, Marcelosr, MaxSem, Mdem, Meestaplu, Meise, Miceduan, Michael Hardy, MisterTS, Mpeg4codec, MrOllie, Nasa-verve, Natevw, Niteowlneils, Nixdorf, Notyetinspace, Patpou77, Pentawing, Philippe, Phosphoricx, Pingveno, Ponchobonjo, Qxz, Radartooth, Ramtam, Reply123, Robert K S, Robin48gx, Rocketmagnet, RolfBly, Ronz, Rvoorhees, SGBailey, Sam8, Selket, Sertrel, Shaddack, Skomorokh, SlackerMom, Slgrandson, Stannered, Stobor827, Suruena, TFolsom, TastyPoutine, Thaiio, TheAllSeeingEye, Thebigandroid, Thunderbird2, TinyMark, Tpikonen, Trailprice, Treekids, Triplestop, Tyler, Versageek, Wdfarmer, Wikipedia tang, Wleizero, Wolfch, Woohookitty, Zer0431, Zigger, ²¹², 328 anonymous edits

Image Sources, Licenses and Contributors

Image:CAN Bit Timing2.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:CAN_Bit_Timing2.svg *License*: unknown *Contributors*: User:Stannered

License