WIKIPEDIA
The Free Encyclopedia

CAN bus

A **Controller Area Network** (**CAN bus**) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other's applications without a host computer. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles to save on copper, but it can also be used in many other contexts. For each device, the data in a frame is transmitted serially but in such a way that if more than one device transmits at the same time, the highest priority device can continue while the others back off. Frames are received by all devices, including by the transmitting device.

History

Development of the CAN bus started in 1983 at Robert Bosch GmbH.[1] The protocol was officially released in 1986 at the Society of Automotive Engineers (SAE) conference in Detroit, Michigan. The first CAN controller chips were introduced by Intel in 1987, and shortly thereafter by Philips.[1] Released in 1991, the Mercedes-Benz W140 was the first production vehicle to feature a CAN-based multiplex wiring system.[2][3]

Bosch published several versions of the CAN specification. The latest is CAN 2.0, published in 1991. This specification has two parts. Part A is for the standard format with an 11-bit identifier, and part B is for the extended format with a 29-bit identifier. A CAN device that uses 11-bit identifiers is commonly called CAN 2.0A, and a CAN device that uses 29-bit identifiers is commonly called CAN 2.0B. These standards are freely available from Bosch along with other specifications and white papers.[4]

In 1993, the International Organization for Standardization (ISO) released CAN standard ISO 11898, which was later restructured into two parts: ISO 11898-1 which covers the data link layer, and ISO 11898-2 which covers the CAN physical layer for high-speed CAN. ISO 11898-3 was released later and covers the CAN physical layer for low-speed, fault-tolerant CAN. The physical layer standards ISO 11898-2 and ISO 11898-3 are not part of the Bosch CAN 2.0 specification.

In 2012, Bosch released CAN FD 1.0, or CAN with Flexible Data-Rate. This specification uses a different frame format that allows a different data length as well as optionally switching to a faster bit rate after the arbitration is decided. CAN FD is compatible with existing CAN 2.0 networks so new CAN FD devices can coexist on the same network with existing CAN devices. As of 2018, Bosch was active in extending CAN standards.

The CAN bus is one of five protocols used in the on-board diagnostics (OBD)-II vehicle diagnostics standard. The OBD-II standard has been mandatory for all cars and light trucks sold in the United States since 1996. The EOBD standard has been mandatory for all petrol vehicles sold in the European Union since 2001 and all diesel vehicles since 2004.[5]

Applications

- Passenger vehicles, trucks, buses (combustion vehicles and electric vehicles)
- Agricultural equipment
- Electronic equipment for aviation and navigation
- Industrial automation and mechanical control
- Elevators, escalators
- Building automation
- Medical instruments and equipment
- Pedelecs
- Model railways/railroads
- Ships and other maritime applications
- Lighting control systems
- 3D Printers
- Robotics/Automation

Automotive

The modern automobile may have as many as 70 electronic control units (ECU) for various subsystems.[6] Traditionally, the biggest processor is the engine control unit. Others are used for Autonomous Driving, Advanced Driver Assistance System (ADAS), transmission, airbags, antilock braking/ABS, cruise control, electric power steering, audio systems, power windows, doors, mirror adjustment, battery and recharging systems for hybrid/electric cars, etc. Some of these form independent subsystems, but communication among others is essential. A subsystem may need to control actuators or receive feedback from sensors. The CAN standard was devised to fill this need. One key advantage is that interconnection

between different vehicle systems can allow a wide range of safety, economy and convenience features to be implemented using software alone - functionality which would add cost and complexity if such features were *hard wired* using traditional automotive electrics. Examples include:

- Auto start/stop: Various sensor inputs from around the vehicle (speed sensors, steering angle, air conditioning on/off, engine temperature) are collated via the CAN bus to determine whether the engine can be shut down when stationary for improved fuel economy and emissions.
- Electric park brakes: The *hill hold* functionality takes input from the vehicle's tilt sensor (also used by the burglar alarm) and the road speed sensors (also used by the ABS, engine control and traction control) via the CAN bus to determine if the vehicle is stopped on an incline. Similarly, inputs from seat belt sensors (part of the airbag controls) are fed from the CAN bus to determine if the seat belts are fastened, so that the parking brake will automatically release upon moving off.
- Parking assist systems: when the driver engages reverse gear, the transmission control unit can send a signal via the CAN bus to activate both the parking sensor system and the door control module for the passenger side door mirror to tilt downward to show the position of the curb. The CAN bus also takes inputs from the rain sensor to trigger the rear windscreen wiper when reversing.
- Auto lane assist/collision avoidance systems: The inputs from the parking sensors are also used by the CAN bus to feed outside proximity data to driver assist systems such as Lane Departure warning, and more recently, these signals travel through the CAN bus to actuate brake by wire in active collision avoidance systems.
- Auto brake wiping: Input is taken from the rain sensor (used primarily for the automatic windscreen wipers) via the CAN bus to the ABS module to initiate an imperceptible application of the brakes while driving to clear moisture from the brake rotors. Some high-performance Audi and BMW models incorporate this feature.
- Sensors can be placed at the most suitable place, and their data used by several ECUs. For example, outdoor temperature sensors (traditionally placed in the front) can be placed in the outside mirrors, avoiding heating by the engine, and data used by the engine, the climate control, and the driver display.

In recent years, the LIN bus (Local Interconnect Network) standard has been introduced to complement CAN for non-critical subsystems such as air-conditioning and infotainment, where data transmission speed and reliability are less critical.

Other

- The CAN bus protocol has been used on the Shimano DI2 electronic gear shift system for road bicycles since 2009, and is also used by the Ansmann and BionX systems in their direct drive motor.
- 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.
- Manufacturers including NISMO aim to use CAN bus data to recreate real-life racing laps in the videogame *Gran Turismo 6* using the game's GPS Data Logger function, which would then allow players to race against real laps.[7]
- Johns Hopkins University's Applied Physics Laboratory's Modular Prosthetic Limb (MPL) uses a local CAN bus to facilitate communication between servos and microcontrollers in the prosthetic arm.
- Teams in the FIRST Robotics Competition widely use CAN bus to communicate between the roboRIO and other robot control modules.
- The CueScript teleprompter range uses CAN bus protocol over coaxial cable, to connect its CSSC – Desktop Scroll Control to the main unit
- The CAN bus protocol is widely implemented due to its fault tolerance in electrically noisy environments such as model railroad sensor feedback systems by major commercial Digital Command Control system manufacturers and various open-source digital model railroad control projects.
- Shearwater Research have implemented the protocol as DiveCAN[8] to use integrating their dive computers into Diving rebreathers from various manufacturers.

Architecture

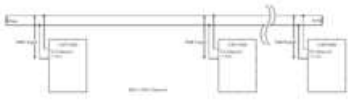
Physical organization

CAN is a multi-master serial bus standard for connecting electronic control units (ECUs) also known as nodes (automotive electronics is a major application domain). Two or more nodes are required on the CAN network to communicate. A node may interface to devices from simple digital logic e.g. PLD, via FPGA up to an embedded computer running extensive software. Such a computer may also be a gateway allowing a general-purpose computer (like a laptop) to communicate over a USB or Ethernet port to the devices on a CAN network.

All nodes are connected to each other through a physically conventional *two-wire bus*. The wires are a twisted pair with a 120 Ω (nominal) characteristic impedance.

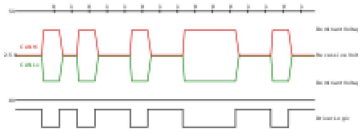
This bus uses differential wired-AND signals. Two signals, CAN high (CANH) and CAN low (CANL) are either driven to a "dominant" state with CANH > CANL, or not driven and pulled by passive resistors to a "recessive" state with CANH ≤ CANL. A 0 data bit encodes a dominant state, while a 1 data bit encodes a recessive state, supporting a wired-AND convention, which gives nodes with lower ID numbers priority on the bus.

ISO 11898-2, also called high-speed CAN (bit speeds up to 1 Mbit/s on CAN, 5 Mbit/s on CAN-FD), uses a linear bus terminated at each end with 120 Ω resistors.



High-speed CAN network. ISO 11898-2.

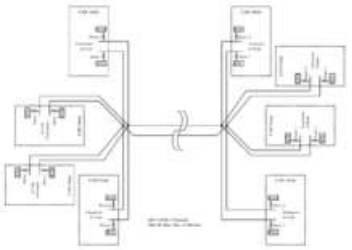
High-speed CAN signaling drives the CANH wire towards 3.5 V and the CANL wire towards 1.5 V when any device is transmitting a dominant (0), while if no device is transmitting a dominant, the terminating resistors passively return the two wires to the recessive (1) state with a nominal differential voltage of 0 V. (Receivers consider any differential voltage of less than 0.5 V to be recessive.) The dominant differential voltage is a nominal 2 V. The dominant common mode voltage (CANH+CANL)/2 must be within 1.5 to 3.5 V of common, while the recessive common mode voltage must be within ±12 V of common.



High-speed CAN signaling. ISO 11898-2.

ISO 11898-3, also called low-speed or fault-tolerant CAN (up to 125 kbit/s), uses a linear bus, star bus or multiple star buses connected by a linear bus and is terminated at each node by a fraction of the overall termination resistance. The overall termination resistance should be close to, but not less than, 100 Ω.

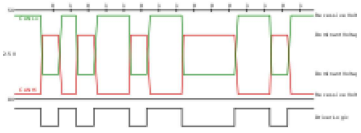
Low-speed fault-tolerant CAN signaling operates similarly to high-speed CAN, but with larger voltage swings. The dominant state is transmitted by driving CANH towards the device power supply voltage (5 V or 3.3 V), and CANL towards 0 V when transmitting a dominant (0), while the termination resistors pull the bus to a recessive state with CANH at 0 V and CANL at 5 V. This allows a simpler receiver that just considers the sign of CANH−CANL. Both wires must be able to handle −27 to +40 V without damage.



Low-speed fault-tolerant CAN network. ISO 11898-3.

Electrical properties

With both high-speed and low-speed CAN, the speed of the transition is faster when a recessive to dominant transition occurs since the CAN wires are being actively driven. The speed of the dominant to recessive transition depends primarily on the length of the CAN network and the capacitance of the wire used.



Low-speed CAN signaling. ISO 11898-3.

High-speed CAN is usually used in automotive and industrial applications where the bus runs from one end of the environment to the other. Fault-tolerant CAN is often used where groups of nodes need to be connected together.

The specifications require the bus be kept within a minimum and maximum common mode bus voltage but do not define how to keep the bus within this range.

The CAN bus must be terminated. The termination resistors are needed to suppress reflections as well as return the bus to its recessive or idle state.

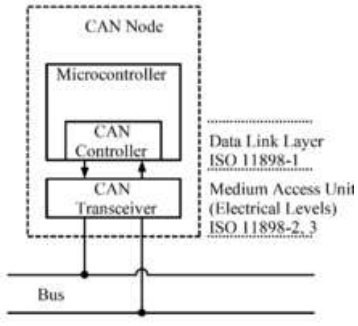
High-speed CAN uses a 120 Ω resistor at each end of a linear bus. Low-speed CAN uses resistors at each node. Other types of terminations may be used such as the Terminating Bias Circuit defined in ISO11783.[9]

A terminating bias circuit provides power and ground in addition to the CAN signaling on a four-wire cable. This provides automatic electrical bias and termination at each end of each bus segment. An ISO11783 network is designed for hot plug-in and removal of bus segments and ECUs.

Nodes

Each node requires a

- Central processing unit, microprocessor, or host processor
  - The host processor decides what the received messages mean and what messages it wants to transmit.
  - Sensors, actuators and control devices can be connected to the host processor.
- CAN controller- often an integral part of the microcontroller
  - Receiving: the CAN controller stores the received serial bits from the bus until an entire message is available, which can then be fetched by the host processor (usually by the CAN controller triggering an interrupt).



CAN bus node

- Sending: the host processor sends the transmit message(s) to a CAN controller, which transmits the bits serially onto the bus when the bus is free.

- Transceiver Defined by ISO 11898-2/3 Medium Access Unit [MAU] standards
  - Receiving: it converts the data stream from CAN bus levels to levels that the CAN controller uses. It usually has protective circuitry to protect the CAN controller.
  - Transmitting: it converts the data stream from the CAN controller to CAN bus levels.

Each node is able to send and receive messages, but not simultaneously. A message or Frame consists primarily of the ID (identifier), which represents the priority of the message, and up to eight data bytes. A CRC, acknowledge slot [ACK] and other overhead are also part of the message. The improved CAN FD extends the length of the data section to up to 64 bytes per frame. The message is transmitted serially onto the bus using a non-return-to-zero (NRZ) format and may be received by all nodes.

The devices that are connected by a CAN network are typically sensors, actuators, and other control devices. These devices are connected to the bus through a host processor, a CAN controller, and a CAN transceiver.

Data transmission

CAN data transmission uses a lossless bitwise arbitration method of contention resolution. This arbitration method requires all nodes on the CAN network to be synchronized to sample every bit on the CAN network at the same time. This is why some call CAN synchronous. Unfortunately the term synchronous is imprecise since the data is transmitted in an asynchronous format, namely without a clock signal.

The CAN specifications use the terms *dominant* bits and *recessive* bits, where dominant is a logical 0 (actively driven to a voltage by the transmitter) and recessive is a logical 1 (passively returned to a voltage by a resistor). The idle state is represented by the recessive level (Logical 1). If one node transmits a dominant bit and another node transmits a recessive bit then there is a collision and the dominant bit wins. This means there is no delay to the higher-priority message, and the node transmitting the lower-priority message automatically attempts to re-transmit six-bit clocks after the end of the dominant message. This makes CAN very suitable as a real-time prioritized communications system.

The exact voltages for a logical 0 or 1 depend on the physical layer used, but the basic principle of CAN requires that each node listen to the data on the CAN network including the transmitting node(s) itself (themselves). If a logical 1 is transmitted by all transmitting nodes at the same time, then a logical 1 is seen by all of the nodes, including both the transmitting node(s) and receiving node(s). If a logical 0 is transmitted by all transmitting node(s) at the same time, then a logical 0 is seen by all nodes. If a logical 0 is being transmitted by one or more nodes, and a logical 1 is being transmitted by one or more nodes, then a logical 0 is seen by all nodes including the node(s) transmitting the logical 1. When a node transmits a logical 1 but sees a logical 0, it realizes that there is a contention and it quits transmitting. By using this process, any node that transmits a logical 1, when another node transmits a logical 0, loses the arbitration and drops out. A node that loses arbitration re-queues its message for later transmission and the CAN frame bit-stream continues without error until only one node is left transmitting. This means that the node that transmits the first 1 loses arbitration. Since the 11 (or 29 for CAN 2.0B) bit identifier is transmitted by all nodes at the start of the CAN frame, the node with the lowest identifier transmits more zeros at the start of the frame, and that is the node that wins the arbitration or has the highest priority.

For example, consider an 11-bit ID CAN network, with two nodes with IDs of 15 (binary representation, 00000001111) and 16 (binary representation, 00000010000). If these two nodes transmit at the same time, each will first transmit the start bit then transmit the first six zeros of their ID with no arbitration decision being made.

	Start bit	ID bits											The rest of the frame
		10	9	8	7	6	5	4	3	2	1	0	
Node 15	0	0	0	0	0	0	0	0	1	1	1	1	
Node 16	0	0	0	0	0	0	0	1	Stopped Transmitting				
CAN data	0	0	0	0	0	0	0	0	1	1	1	1	

When ID bit 4 is transmitted, the node with the ID of 16 transmits 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 knows it transmitted a 1, but sees a 0 and realizes that there is a collision and it lost arbitration. Node 16 stops transmitting which allows the node with ID of 15 to continue its transmission without any loss of data. The node with the lowest ID will always win the arbitration and therefore has the highest priority.

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 improved CAN FD standard allows increasing the bit rate after arbitration and can increase the speed of the data section by a factor of up to ten or more of the arbitration bit rate.

ID allocation

Message IDs must be unique[10] on a single CAN bus, otherwise two nodes would continue transmission beyond the end of the arbitration field (ID) causing an error.

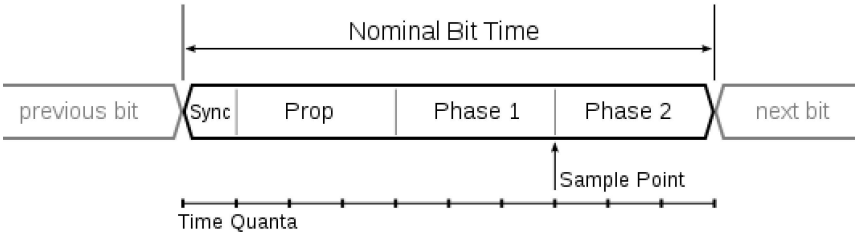
In the early 1990s, the choice of IDs for messages was done simply on the basis of identifying the type of data and the sending node; however, as the ID is also used as the message priority, this led to poor real-time performance. In those scenarios, a low CAN bus use of around 30% was commonly required to ensure that all messages would meet their deadlines. However, if IDs are instead determined based on the deadline of the message, the lower the numerical ID and hence the higher the message priority, then bus use of 70 to 80% can typically be achieved before any message deadlines are missed.[11]

Bit timing

All nodes on the CAN network must operate at the same nominal bit rate, but noise, phase shifts, oscillator tolerance and oscillator drift mean that the actual bit rate might not be the nominal bit rate.[12] Since a separate clock signal is not used, a means of synchronizing the nodes is necessary. Synchronization is important during arbitration since the nodes in arbitration must be able to see both their transmitted data and the other nodes' transmitted data at the same time. Synchronization is also important to ensure that variations in oscillator timing between nodes do not cause errors.

Synchronization starts with a hard synchronization on the first recessive to dominant transition after a period of bus idle (the start bit). Resynchronization occurs on every recessive to dominant transition during the frame. The CAN controller expects the transition to occur at a multiple of the nominal bit time. If the transition does not occur at the exact time the controller expects it, the controller adjusts the nominal bit time accordingly.

The adjustment is accomplished by dividing each bit into a number of time slices called quanta, and assigning some number of quanta to each of the four segments within the bit: synchronization, propagation, phase segment 1 and phase segment 2.



An example CAN bit timing with 10 time quanta per bit

The number of quanta the bit is divided into can vary by controller, and the number of quanta assigned to each segment can be varied depending on bit rate and network conditions.

A transition that occurs before or after it is expected causes the controller to calculate the time difference and lengthen phase segment 1 or shorten phase segment 2 by this time. This effectively adjusts the timing of the receiver to the transmitter to synchronize them. This resynchronization process is done continuously at every recessive to dominant transition to ensure the transmitter and receiver stay in sync. Continuously resynchronizing reduces errors induced by noise, and allows a receiving node that was synchronized to a node that lost arbitration to resynchronize to the node which won arbitration.

Layers

The CAN protocol, like many networking protocols, can be decomposed into the following abstraction layers:

Application layer
Object layer

- Message filtering
- Message and status handling

Transfer layer

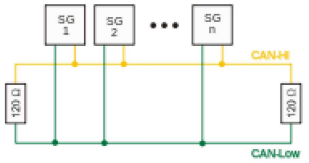
Most of the CAN standard applies to the transfer layer. The transfer layer receives messages from the physical layer and transmits those messages to the object layer. The transfer layer is responsible for bit timing and synchronization, message framing, arbitration, acknowledgment, error detection and signaling, and fault confinement. It performs:

- Fault confinement
- Error detection
- Message validation
- Acknowledgement

- Arbitration
- Message framing
- Transfer rate and timing
- Information routing

Physical layer

CAN bus (ISO 11898-1:2003) originally specified the link layer protocol with only abstract requirements for the physical layer, e.g., asserting the use of a medium with multiple-access at the bit level through the use of dominant and recessive states. The electrical aspects of the physical layer (voltage, current, number of conductors) were specified in ISO 11898-2:2003, which is now widely accepted. However, the mechanical aspects of the physical layer (connector type and number, colors, labels, pin-outs) have yet to be formally specified. As a result, an automotive ECU will typically have a particular—often custom—connector with various sorts of cables, of which two are the CAN bus lines. Nonetheless, several de facto standards for mechanical implementation have emerged, the most common being the 9-pin D-sub type male connector with the following pin-out:



CAN bus electrical sample topology with terminator resistors

- pin 2: CAN-Low (CAN−)
- pin 3: GND (ground)
- pin 7: CAN-High (CAN+)
- pin 9: CAN V+ (power)

This *de facto* mechanical standard for CAN could be implemented with the node having both male and female 9-pin D-sub connectors electrically wired to each other in parallel within the node. Bus power is fed to a node's male connector and the bus draws power from the node's female connector. This follows the electrical engineering convention that power sources are terminated at female connectors. Adoption of this standard avoids the need to fabricate custom splitters to connect two sets of bus wires to a single D connector at each node. Such nonstandard (custom) wire harnesses (splitters) that join conductors outside the node reduce bus reliability, eliminate cable interchangeability, reduce compatibility of wiring harnesses, and increase cost.


A male DE-9 connector (plug)

The absence of a complete physical layer specification (mechanical in addition to electrical) freed the CAN bus specification from the constraints and complexity of physical implementation. However, it left CAN bus implementations open to interoperability issues due to mechanical incompatibility. In order to improve interoperability, many vehicle makers have generated specifications describing a set of allowed CAN transceivers in combination with requirements on the parasitic capacitance on the line. The allowed parasitic capacitance includes both capacitors as well as ESD protection (ESD[13] against ISO 7637-3). In addition to parasitic capacitance, 12V and 24V systems do not have the same requirements in terms of line maximum voltage. Indeed, during jump start events light vehicle lines can go up to 24V while truck systems can go as high as 36V. New solutions are emerging, allowing the same component to be used for CAN as well as CAN FD (see [14]).

Noise immunity on ISO 11898-2:2003 is achieved by maintaining the differential impedance of the bus at a low level with low-value resistors (120 ohms) at each end of the bus. However, when dormant, a low-impedance bus such as CAN draws more current (and power) than other voltage-based signaling buses. On CAN bus systems, balanced line operation, where current in one signal line is exactly balanced by current in the opposite direction in the other signal provides an independent, stable 0 V reference for the receivers. Best practice determines that CAN bus balanced pair signals be carried in twisted pair wires in a shielded cable to minimize RF emission and reduce interference susceptibility in the already noisy RF environment of an automobile.

ISO 11898-2 provides some immunity to common mode voltage between transmitter and receiver by having a 0 V rail running along the bus to maintain a high degree of voltage association between the nodes. Also, in the de facto mechanical configuration mentioned above, a supply rail is included to distribute power to each of the transceiver nodes. The design provides a common supply for all the transceivers. The actual voltage to be applied by the bus and which nodes apply to it are application-specific and not formally specified. Common practice node design provides each node with transceivers that are optically isolated from their node host and derive a 5 V linearly regulated supply voltage for the transceivers from the universal supply rail provided by the bus. This usually allows operating margin on the supply rail sufficient to allow interoperability across many node types. Typical values of supply voltage on such networks are 7 to 30 V. However, the lack of a formal standard means that system designers are responsible for supply rail compatibility.

ISO 11898-2 describes the electrical implementation formed from a multi-dropped single-ended balanced line configuration with resistor termination at each end of the bus. In this configuration a dominant state is asserted by one or more transmitters switching the CAN− to supply 0 V and (simultaneously) switching CAN+ to the +5 V bus voltage thereby forming a current path through the resistors that terminate the bus. As such the terminating resistors form an essential component of the signaling system, and are included, not just to limit wave reflection at high frequency.

During a recessive state, the signal lines and resistor(s) remain in a high-impedance state with respect to both rails. Voltages on both CAN+ and CAN– tend (weakly) towards a voltage midway between the rails. A recessive state is present on the bus only when none of the transmitters on the bus is asserting a dominant state.

During a dominant state, the signal lines and resistor(s) move to a low-impedance state with respect to the rails so that current flows through the resistor. CAN+ voltage tends to +5 V and CAN– tends to 0 V.

Irrespective of signal state the signal lines are always in a low-impedance state with respect to one another by virtue of the terminating resistors at the end of the bus.

This signaling strategy differs significantly from other balanced line transmission technologies such as RS-422/3, RS-485, etc. which employ differential line drivers/ receivers and use a signaling system based on the differential mode voltage of the balanced line crossing a notional 0 V. Multiple access on such systems normally relies on the media supporting three states (active high, active low and inactive tri-state) and is dealt with in the time domain. Multiple access on CAN bus is achieved by the electrical logic of the system supporting just two states that are conceptually analogous to a 'wired AND' network.

Frames

A CAN network can be configured to work with two different message (or *frame*) formats: the standard or base frame format (described in CAN 2.0 A and CAN 2.0 B), and the extended frame format (described only by 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 or remote frame

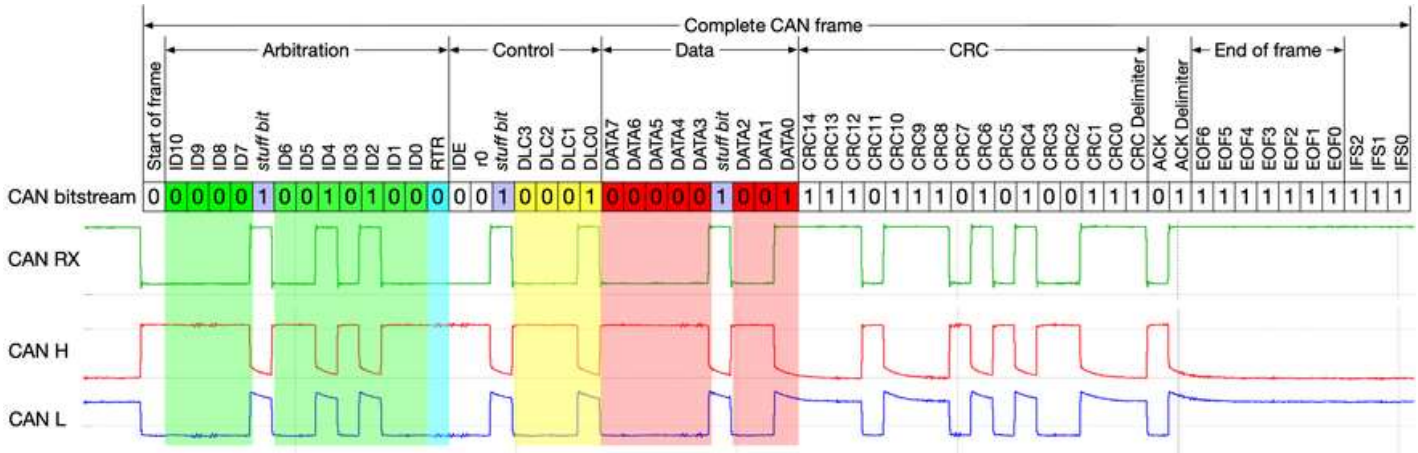
Data frame

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 that 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



A complete CAN bus frame, including stuff bits, a correct CRC, and inter-frame spacing

The frame format is as follows: The bit values are described for CAN-LO signal.

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier (green)	11	A (unique) identifier which also represents the message priority
Stuff bit	1	A bit of the opposite polarity to maintain synchronisation; see § Bit stuffing
Remote transmission request (RTR) (blue)	1	Must be dominant (0) for data frames and recessive (1) for remote request frames (see Remote Frame, below)
Identifier extension bit (IDE)	1	Must be dominant (0) for base frame format with 11-bit identifiers
Reserved bit (r0)	1	Reserved bit. Must be dominant (0), but accepted as either dominant or recessive.
Data length code (DLC) (yellow)	4	Number of bytes of data (0–8 bytes)[a]
Data field (red)	0–64 (0-8 bytes)	Data to be transmitted (length in bytes 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)
Inter-frame spacing (IFS)	3	Must be recessive (1)

a. 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 eight bytes. Certain controllers allow the transmission or reception of a DLC greater than eight, but the actual data length is always limited to eight bytes.

Extended frame format

The frame format is as follows on from here in the table below:

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier A (green)	11	First part of the (unique) identifier which also represents the message priority
Substitute remote request (SRR)	1	Must be recessive (1)
Identifier extension bit (IDE)	1	Must be recessive (1) for extended frame format with 29-bit identifiers
Identifier B (green)	18	Second part of the (unique) identifier which also represents the message priority
Remote transmission request (RTR) (blue)	1	Must be dominant (0) for data frames and recessive (1) for remote request frames (see Remote Frame, below)
Reserved bits (r1, r0)	2	Reserved bits which must be set dominant (0), but accepted as either dominant or recessive
Data length code (DLC) (yellow)	4	Number of bytes of data (0–8 bytes)[a]
Data field (red)	0–64 (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)

a. 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 eight bytes. Certain controllers allow the transmission or reception of a DLC greater than eight, but the actual data length is always limited to eight bytes.

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

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 two 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. The DLC field indicates the data length of the requested message (not the transmitted one).
  I.e.,

  RTR = 0 ; DOMINANT in data frame
  RTR = 1 ; RECESSIVE in remote frame

In the 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.

Error frame

The error frame consists of two different fields:

- The first field is given by the superposition of ERROR FLAGS (6–12 dominant/recessive bits) contributed from different stations.
- The following second field is the ERROR DELIMITER (8 recessive bits).

There are two types of error flags:

Active Error Flag
six dominant bits – Transmitted by a node detecting an error on the network that is in error state *error active*.
Passive Error Flag
six recessive bits – Transmitted by a node detecting an active error frame on the network that is in error state *error passive*.

There are two error counters in CAN:

1. Transmit error counter (TEC)
2. Receive error counter (REC)

- When TEC or REC is greater than 127 and less than 255, a Passive Error frame will be transmitted on the bus.
- When TEC and REC is less than 128, an Active Error frame will be transmitted on the bus.
- When TEC is greater than 255, then the node enters into Bus Off state, where no frames will be transmitted.

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.

ACK slot

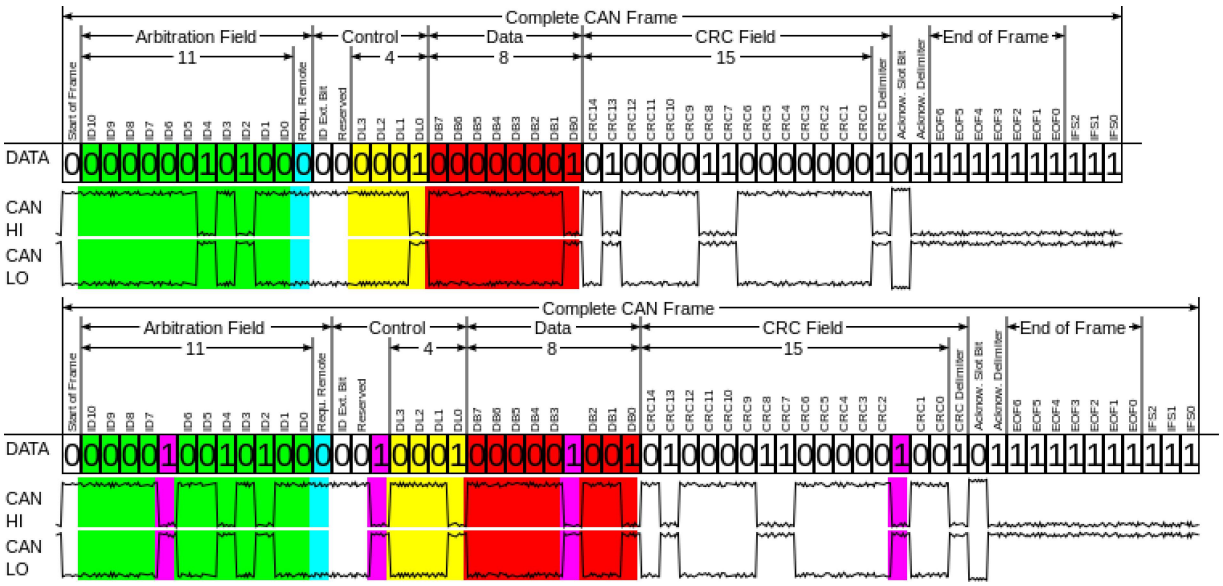
The acknowledge slot is used to acknowledge the receipt of a valid CAN frame. Each node that receives the frame, without finding an error, transmits a dominant level in the ACK slot and thus overrides the recessive level of the transmitter. If a transmitter detects a recessive level in the ACK slot, it knows that no receiver found a valid frame. A receiving node may transmit a recessive to indicate that it did not receive a valid frame, but another node that did receive a valid frame may override this with a dominant. The transmitting node cannot know that the message has been received by all of the nodes on the CAN network.

Often, the mode of operation of the device is to re-transmit unacknowledged frames over and over. This may lead to eventually entering the *error passive* state.

Interframe spacing

Data frames and remote frames are separated from preceding frames by a bit field called interframe space. Interframe space consists of at least three consecutive recessive (1) bits. Following that, if a dominant bit is detected, it will be regarded as the *Start of frame* bit of the next frame. 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 suspend transmission for error passive stations, which have been transmitter of the previous message.[15]

Bit stuffing



CAN frame before and after the addition of stuff bits (in purple). An incorrect CRC is used for bit stuffing illustration purposes.

To ensure enough transitions to maintain synchronization, a bit of opposite polarity is inserted after five consecutive bits of the same polarity. This practice is called bit stuffing, and is necessary due to the non-return-to-zero (NRZ) coding used with CAN. The stuffed data frames are destuffed by the receiver.

All fields in the frame are stuffed with the exception of the CRC delimiter, ACK field and end of frame which are a fixed size and are not stuffed. In the fields where bit stuffing is used, six consecutive bits of the same polarity (111111 or 000000) are considered an error. An active error flag can be transmitted by a node when an error has been detected. The active error flag consists of six consecutive dominant bits and violates the rule of bit stuffing.

Bit stuffing means that data frames may be larger than one would expect by simply enumerating the bits shown in the tables above. The maximum increase in size of a CAN frame (base format) after bit stuffing is in the case

11111000011110000...

which is stuffed as (stuffing bits in bold):

11111**0**00001111**0**00001...

The stuffing bit itself may be the first of the five consecutive identical bits, so in the worst case there is one stuffing bit per four original bits.

The size of a base frame is bounded by

$$8n + 44 + \left\lfloor \frac{34 + 8n - 1}{4} \right\rfloor$$

where n is the number of data bytes, a maximum of 8.

Since $8n + 44$ is the size of the frame before stuffing, in the worst case one bit will be added every four original bits after the first one (hence the −1 at the numerator) and, because of the layout of the bits of the header, only 34 out of 44 of them can be subject to bit stuffing.

frame type	before stuffing	after stuffing	stuffing bits	total frame length
base frame	$8n + 44$	$8n + 44 + \left\lfloor \frac{34 + 8n - 1}{4} \right\rfloor$	≤ 24	≤ 132
extended frame	$8n + 64$	$8n + 64 + \left\lfloor \frac{54 + 8n - 1}{4} \right\rfloor$	≤ 29	≤ 157

An undesirable side effect of the bit stuffing scheme is that a small number of bit errors in a received message may corrupt the destuffing process, causing a larger number of errors to propagate through the destuffed message. This reduces the level of protection that would otherwise be offered by the CRC against the original errors. This deficiency of the protocol has been addressed in CAN FD frames by the use of a combination of fixed stuff bits and a counter that records the number of stuff bits inserted.

CAN lower-layer standards

ISO 11898 series specifies physical and data link layer (levels 1 and 2 of the ISO/OSI model) of serial communication category called Controller Area Network that supports distributed real-time control and multiplexing for use within road vehicles.[16]

There are several CAN physical layer and other standards:

ISO 11898-1:2015 specifies the data link layer (DLL) and physical signalling of the controller area network (CAN).[17] This document describes the general architecture of CAN in terms of hierarchical layers according to the ISO reference model for open systems interconnection (OSI) established in ISO/IEC 7498-1 and provides the characteristics for setting up an interchange of digital information between modules implementing the CAN DLL with detailed specification of the logical link control (LLC) sublayer and medium access control (MAC) sublayer.

ISO 11898-2:2016 specifies the high-speed (transmission rates of up to 1 Mbit/s) medium access unit (MAU), and some medium-dependent interface (MDI) features (according to ISO 8802-3), which comprise the physical layer of the controller area network. ISO 11898-2 uses a two-wire balanced signaling scheme. It is the most used physical layer in vehicle powertrain applications and industrial control networks.

ISO 11898-3:2006 specifies low-speed, fault-tolerant, medium-dependent interface for setting up an interchange of digital information between electronic control units of road vehicles equipped with the CAN at transmission rates above 40 kbit/s up to 125 kbit/s.

ISO 11898-4:2004 specifies time-triggered communication in the CAN (TTCAN). It is applicable to setting up a time-triggered interchange of digital information between electronic control units (ECU) of road vehicles equipped with CAN, and specifies the frame synchronization entity that coordinates the operation of both logical link and media access controls in accordance with ISO 11898-1, to provide the time-triggered communication schedule.

ISO 11898-5:2007 specifies the CAN physical layer for transmission rates up to 1 Mbit/s for use within road vehicles. It describes the medium access unit functions as well as some medium-dependent interface features according to ISO 8802-2. This represents an extension of ISO 11898-2, dealing with new functionality for systems requiring low-power consumption features while there is no active bus communication.

ISO 11898-6:2013 specifies the CAN physical layer for transmission rates up to 1 Mbit/s for use within road vehicles. It describes the medium access unit functions as well as some medium-dependent interface features according to ISO 8802-2. This represents an extension of ISO 11898-2 and ISO 11898-5, specifying a selective wake-up mechanism using configurable CAN frames.

ISO 16845-1:2016 provides the methodology and abstract test suite necessary for checking the conformance of any CAN implementation of the CAN specified in ISO 11898-1.

ISO 16845-2:2018 establishes test cases and test requirements to realize a test plan verifying if the CAN transceiver with implemented selective wake-up functions conform to the specified functionalities. The kind of testing defined in ISO 16845-2:2018 is named as conformance testing.

CAN-based higher-layer protocols

As the CAN standard does not include common communication features, such as flow control, device addressing, and transportation of data blocks larger than one message, and above all, application data, many implementations of higher layer protocols were created. Several are standardized for a business area, although all can be extended by each manufacturer. For passenger cars, each manufacturer has its own standard.

CAN in Automation (CiA) is the international users' and manufacturers' organization that develops and supports CAN-based higher-layer protocols and their international standardization.[18] Among these specifications are:

Standardized approaches

- ARINC 812 or ARINC 825 (aviation industry)
- CANopen - CiA 301/302-2 and EN 50325-4 (industrial automation)
- IEC 61375-3-3 (use of CANopen in rail vehicles)
- DeviceNet (industrial automation)

- EnergyBus - CiA 454 and IEC 61851-3 (battery–charger communication)
- ISOBUS - ISO 11783 (agriculture)
- ISO-TP - ISO 15765-2 (transport protocol for automotive diagnostics)
- MilCAN (military vehicles)
- NMEA 2000 - IEC 61162-3 (marine industry)
- SAE J1939 (in-vehicle network for buses and trucks)
- SAE J2284 (in-vehicle networks for passenger cars)
- Unified Diagnostic Services (UDS) - ISO 14229 (automotive diagnostics)
- LeisureCAN - open standard for the leisure craft/vehicle industry

Other approaches

- CANaerospace - Stock (for the aviation industry)
- CAN Kingdom - Kvaser (embedded control system)
- CCP/XCP (automotive ECU calibration)
- GMLAN - General Motors (for General Motors)
- RV-C - RVIA (used for recreational vehicles)
- SafetyBUS p - Pilz (used for industrial automation)
- UAVCAN (aerospace and robotics)
- CSP (CubeSat Space Protocol)
- VSCP (Very Simple Control Protocol) a free automation protocol suitable for all sorts of automation tasks

CANopen Lift

The CANopen Special Interest Group (SIG) "Lift Control", which was founded in 2001, develops the CANopen application profile CiA 417 for lift control systems. It works on extending the features, improves technical content and ensures that the current legal standards for lift control systems are met. The first version of CiA 417 was published (available for CiA members) in summer 2003, version 2.0 in February 2010, version 2.1.0 in July 2012, version 2.2.0 in December 2015, and version 2.3.1 in February 2020.

Jörg Hellmich (ELFIN GmbH) is the chairman of this SIG and manages a wiki of the CANopen lift community (https://en.canopen-lift.org/wiki/) with content about CANopen lift.

Security

CAN is a low-level protocol and does not support any security features intrinsically. There is also no encryption in standard CAN implementations, which leaves these networks open to man-in-the-middle frame interception. In most implementations, applications are expected to deploy their own security mechanisms; e.g., to authenticate incoming commands or the presence of certain devices on the network. Failure to implement adequate security measures may result in various sorts of attacks if the opponent manages to insert messages on the bus.[19] While passwords exist for some safety-critical functions, such as modifying firmware, programming keys, or controlling antilock brake actuators, these systems are not implemented universally and have a limited number of seed/key pairs.

Development tools

When developing or troubleshooting the CAN bus, examination of hardware signals can be very important. Logic analyzers and bus analyzers are tools that collect, analyse, decode and store signals so people can view the high-speed waveforms at their leisure. There are also specialist tools as well as CAN bus monitors.

A CAN bus monitor is an analysis tool, often a combination of hardware and software, used during development of hardware that uses the CAN bus.

Typically the CAN bus monitor will listen to the traffic on the CAN bus in order to display it in a user interface. Often the CAN bus monitor offers the possibility to simulate CAN bus activity by sending CAN frames to the bus. The CAN bus monitor can therefore be used to validate expected CAN traffic from a given device or to simulate CAN traffic in order to validate the reaction from a given device connected to the CAN bus.

Licensing

Bosch holds patents on the technology, though those related to the original protocol have now expired. Manufacturers of CAN-compatible microprocessors pay license fees to Bosch for use of the CAN trademark and any of the newer patents related to CAN FD, and these 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 need to pay a fee for the CAN Protocol License if they wish to use the CAN trademark or CAN FD capabilities.[20]

See also

- Byteflight
- Car audio – Entertainment electronics in cars
- CAN bus monitor – Standard for communication between devices without host computer
- CANopen - Communication protocol for embedded systems
- CANpie – Open source device driver for CAN
- CAN FD – New implementation of CAN with a faster transmission
- can4linux – Open source Linux device driver for CAN
- FlexCAN – An alternative implementation.
- FlexRay – High-speed alternative to CAN
- List of network buses – List of single collision domain electronic communication bus systems
- Local Interconnect Network – A low cost alternative
- Modbus – Serial communications protocol mainly developed for programmable logic controllers
- MOST bus – High-speed multimedia network technology used in the automotive industry
- OBD-II PIDs – List of Parameter IDs
- OSEK – Specifications for embedded operating systems within automotives
- SAE J1939 - Communication protocol for trucks and busses
- SocketCAN – A set of open-source CAN drivers and a networking stack contributed by Volkswagen Research to the Linux kernel.

References

1. "CAN History" (http://www.can-cia.org/can-knowledge/can/can-history/). CAN in Automation.
2. "Mercedes-Benz S-Class W 140" (https://www.mercedes-benz.com/en/mercedes-benz/classic/history/mercedes-benz-s-class-w-140/). *mercedes-benz.com*. 23 February 2016. Retrieved 27 October 2017.
3. "CAN in Automation - Mercedes W140: First car with CAN" (https://can-newsletter.org/engineering/applications/160322_25th-anniversary-mercedes-w140-first-car-with-can/). *can-newsletter.org*. Retrieved 27 October 2017.
4. "Bosch Semiconductor CAN Literature" (https://web.archive.org/web/20170523083421/http://www.bosch-semiconductors.de/en/automotive_electronics/ip_modules/can_literature_2.html). Archived from the original (http://www.bosch-semiconductors.de/en/automotive_electronics/ip_modules/can_literature_2.html) on 2017-05-23. Retrieved 2017-05-31.
5. *Building Adapter for Vehicle On-board Diagnostic* (http://www.obddiag.net/adapter.html) Archived (https://web.archive.org/web/20180514092143/http://www.obddiag.net/adapter.html) 2018-05-14 at the Wayback Machine, obddiag.net, accessed 2009-09-09
6. Comparison of Event-Triggered and Time-Triggered Concepts with Regard to Distributed Control Systems A. Albert, Robert Bosch GmbH Embedded World, 2004, Nürnberg
7. "NISMO Increases GT6 GPS Data Logger Functionality and Track Count" (http://www.gtplanet.net/nismo-increases-gt6-gps-data-logger-functionality-and-track-count/). *www.gtplanet.net*. 25 October 2014.
8. "What is DiveCAN and why should I care?" (https://www.shearwater.com/monthly-blog-posts/what-is-divecan-and-why-should-i-care/). 22 March 2016.
9. "ISO11783 a Standardized Tractor – Implement Interface" (https://www.can-cia.org/fileadmin/resources/documents/proceedings/2003_fellmeth.pdf) (PDF).
10. *ISO 11898-1:2015 – Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling* (https://www.iso.org/standard/63648.html).
11. Daigmorte, Hugo; Boyer, Marc (2017), "Evaluation of admissible CAN bus load with weak synchronization mechanism" (https://dl.acm.org/citation.cfm?doid=3139258.3139261), *Proc. of the 24th Int. Conf. on Real-Time Networks and Systems (RTNS 2017)*, Grenoble, France: ACM
12. "Understanding Microchip's CAN Module Bit Timing" (http://ww1.microchip.com/downloads/en/AppNotes/00754.pdf) (PDF).
13. "ISO7637-3 diodes protection for CAN bus" (http://www.st.com/content/st_com/en/product-selector.html?querycriteria=productId=SC2185$$1=ESDCAN*).
14. "CAN bus ESD protection" (http://www.st.com/content/st_com/en/product-selector.html?querycriteria=productId=SC2185$$1=ESDCAN*).
15. "CAN BUS MESSAGE FRAMES – Overload Frame, Interframe Space" (http://rs232-rs485.blogspot.com/2009/11/can-bus-message-frames-overload.html). 18 November 2009.
16. "Controller Area Network (CAN)" (https://web.archive.org/web/20160425131652/http://vector.com/vi_controller_area_network_en.html). Vector Group. Archived from the original (https://vector.com/vi_controller_area_network_en.html) on 25 April 2016. Retrieved 25 Sep 2013.

17. "ISO 11898-1:2003 - Road vehicles -- Controller area network (CAN) -- Part 1: Data link layer and physical signalling" (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=33422). ISO.
18. CiA: International standardization (https://www.can-cia.org/groups/international-standardization/).
19. "We Drove a Car While It Was Being Hacked" (https://www.vice.com/en_us/article/ae33jk/we-drove-a-car-while-it-was-being-hacked). *www.vice.com*. Archived (https://web.archive.org/web/20191108154439/https://www.vice.com/en_us/article/ae33jk/we-drove-a-car-while-it-was-being-hacked) from the original on 8 November 2019.
20. "License Conditions CAN Protocol and CAN FD Protocol" (https://web.archive.org/web/20160316021829/http://www.bosch-semiconductors.de/media/automotive_electronics/pdf_2/ipmodules_3/can_protocol_license_1/Bosch_CAN_Protocol_License_Conditions.pdf) (PDF). Archived from the original (http://www.bosch-semiconductors.de/media/automotive_electronics/pdf_2/ipmodules_3/can_protocol_license_1/Bosch_CAN_Protocol_License_Conditions.pdf) (PDF) on 2016-03-16. Retrieved 2016-03-15.

External links

Specifications

- ISO 11898-1 Standard (2015) (http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=63648) - includes CAN and CAN-FD specifications
- Bosch CAN Specification Version 2.0 (1991, 1997) (https://web.archive.org/web/20221010170747/http://esd.cs.ucr.edu/webres/can20.pdf) - also known as *Classical CAN* and *CAN-Classic*
- Bosch CAN-FD Specification Version 1.0 (2012) (https://web.archive.org/web/20151211125301/http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can_fd_spec.pdf) - increase data rates up to 8 Mbit/s
- Bosch CAN-XL (future) (https://www.bosch-semiconductors.com/ip-modules/can-protocols/can-xl/) - increase data rates up to 20 Mbit/s
- Bosch CAN-FD-Light (future) (https://web.archive.org/web/20221022035940/https://www.bosch-semiconductors.com/media/ip_modules/pdf_2/can_fd_light/20220825_can_fd_light.pdf) - cost-optimized subset of CAN-FD

Other

- Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised (https://doi.org/10.1007%2Fs11241-007-9012-7)
- Pinouts for common CAN bus connectors (http://www.interfacebus.com/Can_Bus_Connector_Pinout.html)
- A webpage about CAN in automotive (http://marco.guardigli.it/2010/10/hacking-your-car.html)
- Controller Area Network (CAN) Schedulability Analysis with FIFO Queues (https://web.archive.org/web/20190616130249/https://www.cs.york.ac.uk/ftpdir/reports/2011/YCS/462/YCS-2011-462.pdf)
- Controller Area Network (CAN) Implementation Guide (http://www.analog.com/static/imported-files/application_notes/AN-1123.pdf)
- Freeware Bit-Timing calculator for Windows, supports a lot of microcontrollers, e.g. Atmel, STM32, Microchip, Renesas, ... (http://www.mhs-elektronik.de/userdata/downloads/BitCalc.zip) (ZIPfile)
- Free e-learning module "Introduction to CAN" (https://web.archive.org/web/20170801161111/https://elearning.vector.com/vl_can_introduction_en.html)
- ARINC-825 Tutorial (video) (http://www.mil-1553.com/tutorials-arinc) from Excalibur Systems Inc.
- Website of CiA (http://www.can-cia.org)
- CAN Newsletter Online (http://www.can-newsletter.org)
- Understanding and Using the Controller Area Network from UC Berkeley (https://web.archive.org/web/20161213185005/http://inst.cs.berkeley.edu/~ee249/fa08/Lectures/handout_canbus2.pdf)
- CAN Protocol Tutorial (https://www.kvaser.com/can-protocol-tutorial/)
- ESD protection (https://web.archive.org/web/20160807075624/http://blog.st.com/can-bus-protection-stmicroelectronics-protects-what-protects-you/) for CAN bus and CAN FD
