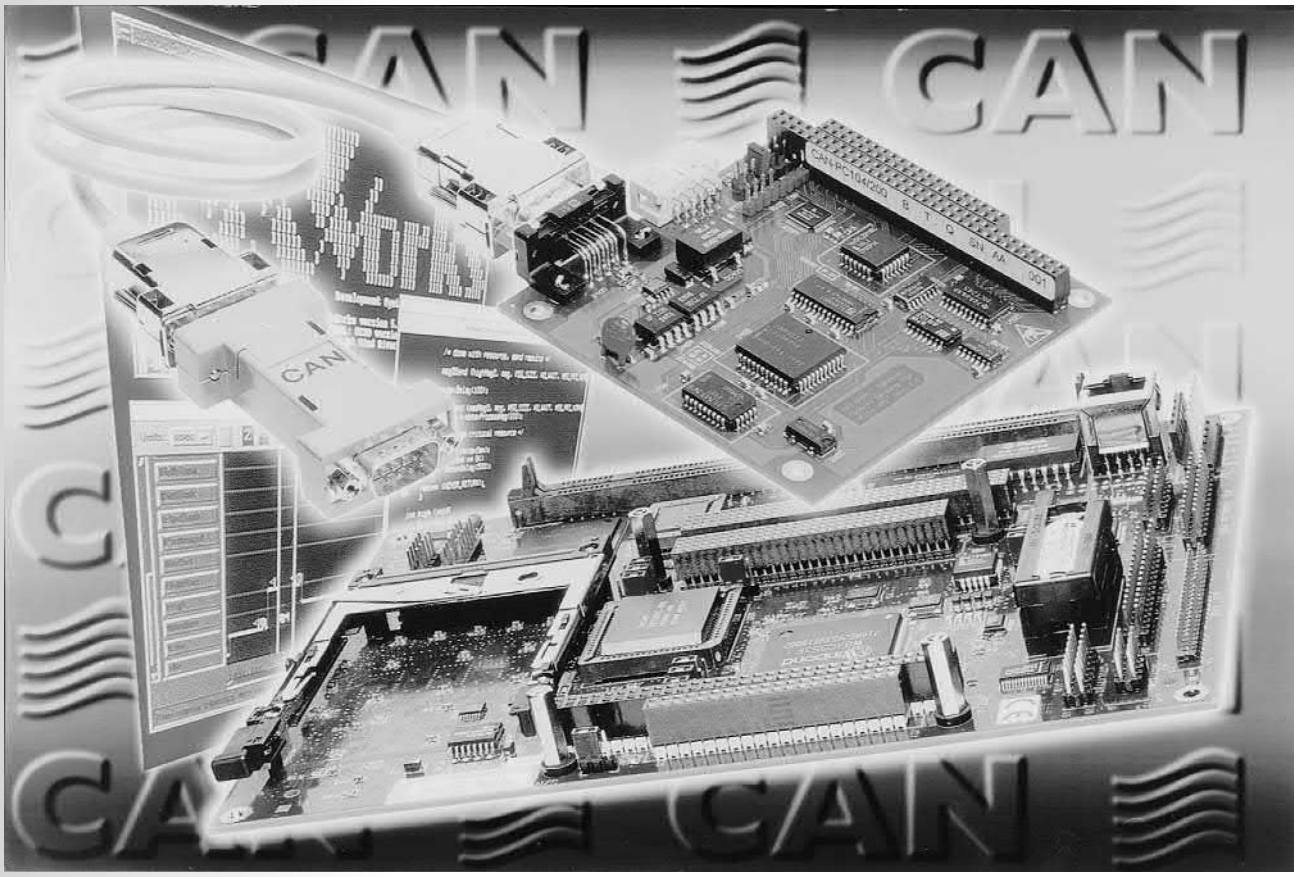
# the CAN bus

## *intelligent, decentralized data communications Part 1*

The proliferation of local communication networks in industrial systems seems inexorable: CAN bus, Profibus, LON, ASI, Interbus-S, FIP, EIB, eBus, and many more. The time has now arrived where these well-established technologies, protocol implementation in small silicon chips, dropping prices, and simple maintenance have become accessible to the smaller engineering firms and the enthusiastic amateur engineer/technician. In this short series of articles, the technologies underlying the CAN bus system will be described in a simple and practical manner. In a later part, a complete CAN bus interface for a microcontroller system will be presented.




By B vom Berg & P Groppe

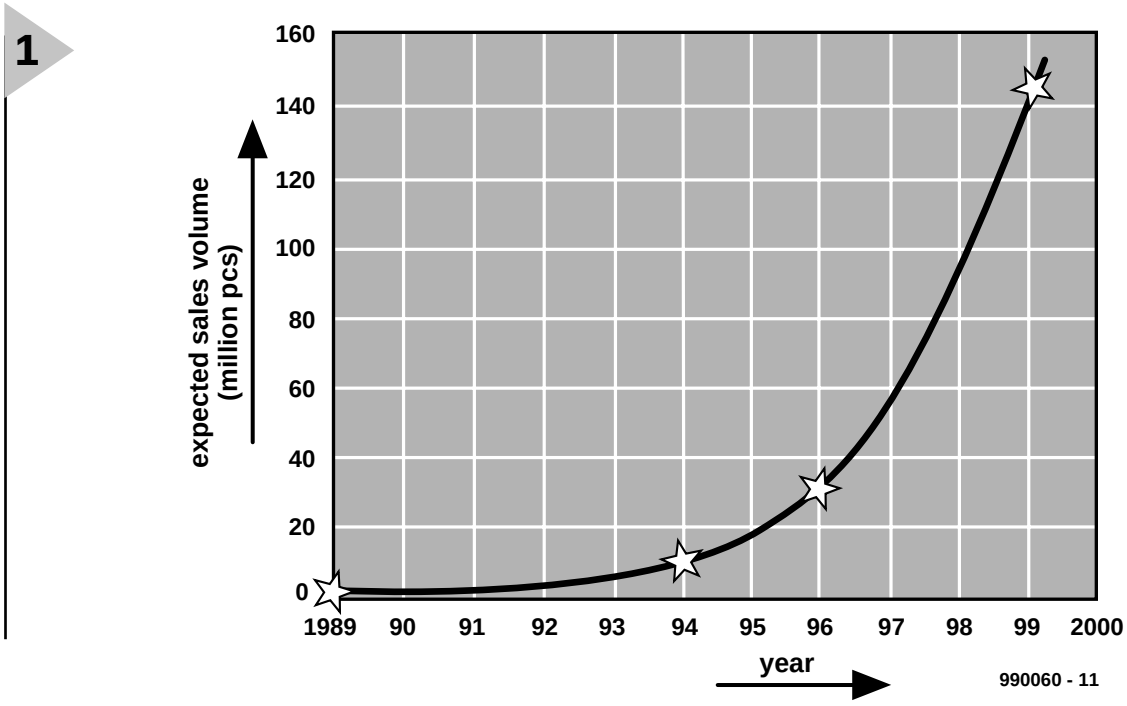



***Figure 1. Estimated world-wide sales volume of CAN ICs.***

## INTRODUCTION

This first part in the series will take a brief look at the history and standardization of the CAN bus. It will also deal with the most important characteristics of the physical layer.

In the second part, the data link layer will be explained. This part will also contain the CAN building bricks of various manufacturers in tabular form. This is followed by the introduction of a universal CAN bus interface that may be used to make a variety of microcontroller and microprocessor system suitable for use in a CAN bus network.

The third part describes the construction, programming and usage of a small CAN bus network in conjunction with a PC and a microcontroller card.

## DEVELOPMENT OF THE CAN BUS

In the early 1990s, the international car industry was faced with two problems concerned with the future development of private cars and goods vehicles. The first was concerned with the growing demand for more comfort in vehicles: electrically operated windows, seat and mirror adjustment, heated seats, electronic climate control, as well as audio-visual equipment and satellite-controlled navigation systems (GPS = Global Positioning System).

The second and more important was vehicle security, not only from an individual point of view but also to meet more stringent international safety regulations: central door locking, immobilizing systems; ABS (anti-lock braking systems), as well as economical and environment-friendly engine management.

Both problems were tackled by intense electronification of, and communication between, the many units comprising a modern vehicle. It is estimated that vehicles manufactured by the year 2005 will contain up to 100 microcontroller and all these need to communicate with each other. It is clear that all these communications paths result in a larger and more extensive cable harness. For instance, that in a modern good-quality motor car weighs close to 100 kg (220 lb) and contains up to 2000 metres (one and half mile) of cable. Moreover, a typical large car manufacturer may have up to 600 different types of cable harness in use.

Since this is clearly an untenable situation, the car industry began to look at new ways of communication and found this in the computer industry. Obviously, the bus systems used in that industry had to be adapted for use in vehicles, more particularly as regards the following:

- low-speed and high-speed data transfer in the range 5 kbit/s to 1 Mbit/s for comfort and safety systems;
- error-free data transfer with a Hamming distance greater than 4;
- optimum transfer of tiny data streams such as obtained from sensors or actuators, that is, consisting of 0–8 bytes per message;
- ease of maintenance;
- low cost (mass production);
- simplicity of bus construction (bus media, bus topology) for easy integration into the vehicle.

Unfortunately, all large car manufac-

**2**

| ISO / OSI Layer model | CAN Bus Layer | | |
|---|---|---|---|
| ***Layer 8*** **Application: "Device on Bus"** | **CANopen** | **DeviceNet** | **Smart Distributed System (SDS)** |
| ***Layer 7*** **"Application Layer"** | **CAL: CAN Application layer for industrial Applications** | **DeviceNet Specifications** | **SDS Specifications** |
| ***Layer 3 - 6*** | ***Empty !!*** | | |
| ***Layer 2*** **"Data Link Layer"** | **LLC: Logical Link Control**<br>**MAC: Medium Access Control**<br>**acc. to ISO 11898**<br>**Result:**<br>**CAN 2.0 A, CAN 2.0 B } Specifications** | | |
| ***Layer 1*** **"Physical Layer"** | **"Low-Speed CAN" ISO 11519-2** | **"High-Speed CAN" ISO 11898** | |

990060 - 12

***Figure 2. The CAN bus in the ISO/OSI layer model.***

turers developed their own bus concept, which, of course, was not compatible with that of other manufacturers. All of them tried to have their system adopted as *the* international system, that is, to be accepted as *the* international standard – with clear economic and commercial advantages to them.

Not all systems could be readily standardized. Essentially, four of them have survived: CAN (Controller Area Network) in low-speed and high-speed versions; VAN; J1850SCP; and J1850DLC. Of these, VAN (standardized) and most others (not standardized) have been abandoned during the mid-1990s in favour of the CAN system. Today, the CAN system is *the* world leader in the field of vehicle buses. As such it is used nowadays not only in luxury cars from Mercedes, Lexus, Jaguar, and Chrysler, but also in less prestigious ones from Fiat and Volkswagen.

It was not only the vehicle industry that discovered the advantages of bus systems, but also the automation and production industries. These industries use the CAN concept for measuring, controlling and driving in SPS (Standard Positioning Service) systems, robots and motors. The concept is also used in civil engineering, elevator (lift) control, laboratory automation systems, sensor/actuator systems, and others.

The CAN protocol is available programmed in a silicon chip, so that the user need no longer concern him/herself with the finer details of the communication technology: CAN ICs are integrated simply as intelligent peripheral building blocks in existing microcontroller systems, or those being designed.

The virtually troublefree maintenance and usage, as well as the rapidly dropping prices, of CAN ICs make CAN buses of great interest and usefulness for non-industrial designers when it comes to small, decentralized communication networks.

**Figure 1** shows the estimated world-wide sales volume of CAN ICs. The costs of a complete CAN network from various semiconductor manufacturers are given in US dollars.

## STANDARDIZATION

If the structure of a communication system is to be universally accepted, a number of questions should be answered clearly and the answers incorporated in the relevant standard.

- How will the various parts of the network be physically (electrically and logically) arranged?
- What does the consequent topology of the network look like?
- How are the data coupled to, and transferred by, the relevant medium (cable, fibre optic conductor, air, or infra-red remote control)?
- What are the rules for data exchange between the various parts?
- How are data transfer errors prevented, recognized, and corrected?
- How is the data transfer protocol formed?
- How is access to the data transfer medium arranged for parts that are about to send data?
- How are conflicts resolved when several parts want to send data at the same time? This concerns the access to the medium, that is, the so-called arbitration.

As far as receiving data is concerned, there not many problems, since as a rule many receivers can be connected to the medium, all of which can receive data without any difficulty at the same time. In general, in any communication system, there should be only one sender active at any one time, although several receivers may be.

The answers to the foregoing questions, and others, must be laid down clearly and unambiguously when a communication system is to be used in a sensible manner that is acceptable world-wide.

In the early 1990s, the International Standardization Organization (ISO) started laying down an international standard for vehicle buses, during which process the CAN bus assumed an increasingly strong position.

The basis for the standardization process in the enormous area of open, non-producer-inspired data communications is a seven-layer ISO/OSI reference model. In the case of certain communication systems, including the vehicle bus system, layers 3–6 are empty, so that for the CAN bus only layers 1,2 and 7 are specified in detail (see **Figure 2**).

**Layer 1: physical layer**

In this layer, the specifications for the data transfer medium, connectors, the data transfer levels, and the send and receive elements are laid down. The two associated CAN standards are:

**ISO11529-2**: low-speed CAN. The basis of this is a development started by Bosch of Germany in the early 1980s, and continued, with the strong support of Intel, into the integration of the protocol into an IC. The low speed refers to data transfer rates from

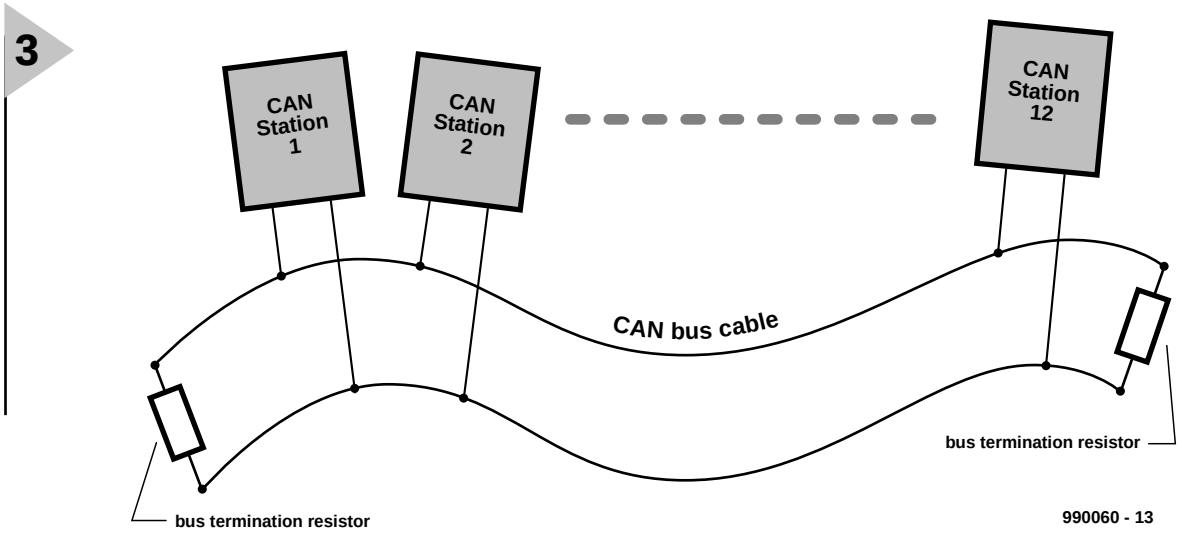



***Figure 3. Topology of the CAN bus.***

5 kbit/s to 125 kbit/s.

**ISO11898**: high-speed CAN. This standard supports data transfer rates of up to 1 Mbit/s.

**Layer 2: data link layer**
This layer lays down how, when a part wants to send data, the data transfer medium7m is accessed, how a message is composed (address, data, control and protection against errors), and how the data transfer protocol is structured. The standards for these may also be found in ISO11898.

In addition, the 1991 CAN specification is expanded in Layer 2, so that today there two versions: CAN2.0A and CAN2.0B. The similarities and differences between these two will be reverted to in Part 2.

Since the CAN concept has been adopted by so may different industries, the situation in Layer 7: application layer, is diverse and somewhat confused. This is because this layer defines the interface for the actual application (Layer 8) of the bus with industrial equipment.

Three extensive CAN branches for different applications were developed over the years: CANopen, DeviceNet, and Smart Distributed System (SDS). Since these specifications are fairly extensive, they will not be pursued in this article. All that will be said here is that the concepts are common, so that they are compatible with Layers 1 and 2. Detailed information on CANopen, DeviceNet, and SDS may be found on the internet: http://www.can-cia.de.

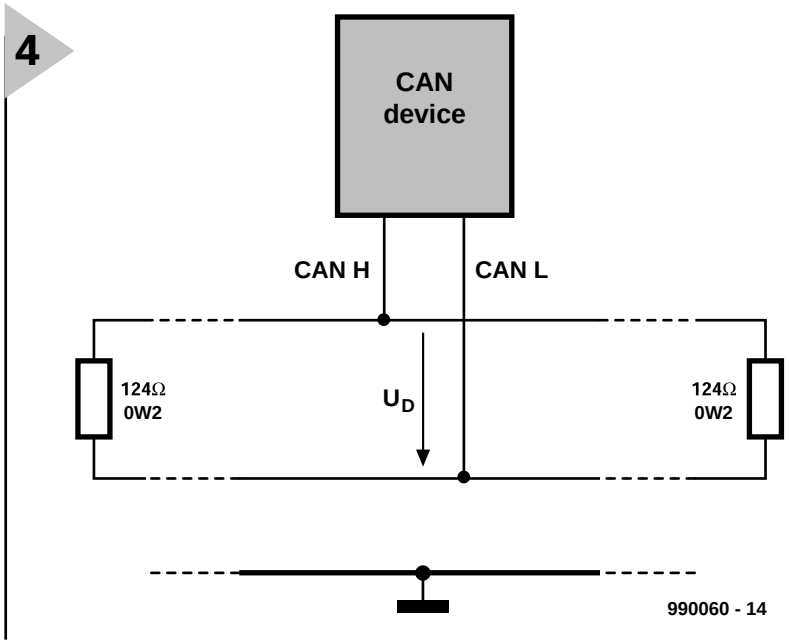
**CHARACTERISTICS**
The physical layer comprises the network topology of the CAN bus and the linking to the bus medium.

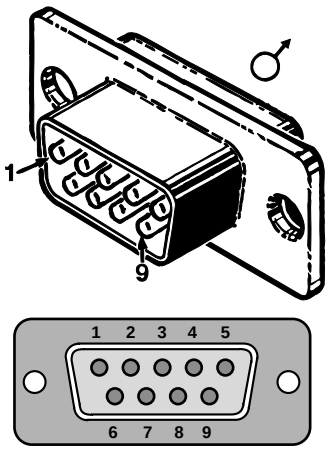
The term network topology includes the physical construction of

| Voltage at ...\\ Bus state | recessive | dominant |
|---|---|---|
| CANH | 2.5 V | 3.5 V |
| CANL | 2.5 V | 1.5 V |
| allowable voltage difference $U_D$ = CANH - CANL | 0 - 0.5 V | 0.9 - 2.0 V |

***Table 1. Absolute levels of the bus lines with respect to (local) earth according to ISO11898.***

***Figure 4. Linking an element (station) to the CAN bus.***






1 Reserved
2 CAN L
3 CAN GND
4 Reserved
5 Optional: CAN screening
6 GND
7 CAN H
8 Reserved
9 CAN V+ (optional external supply)

990060 - 15

***Figure 5. Pinout of a CAN bus connector.***

the communication system and so gives the answer to 'how are the parts (stations) linked to the data transfer medium?'

The CAN bus uses the so-called bus topology, that is, all parts are connected to a single twisted-pair cable (screened or not), which is terminated at both ends into the relevant bus termination impedance (see **Figure 3**). This arrangement ensures that each station can communicate with any other station in the network without any limitation.

The send/receive stage of a CAN network element is linked to the bus medium via two connectors: CAN High (CANH) and CAN Low (CANL) (see **Figure 4**).

In view of the requisite protection against errors, differential voltage signals are used for the actual data transfer. This means that the voltage different between the two bus lines is quantized. ISO11898 specifies two different differential-voltage ranges for data representation: recessive and dominant. There is a good reason that the usual logic 0 and 1 levels are not used here and this will be reverted to. For the time being, note that

- if the differential voltage between CANH and CANL ≤ 0.5 V, the status is recessive;
- if the differential voltage ≥ 0.9 V, the status is dominant.

The nominal level of the bus line, that is, the level of the individual lines with respect to (local) ground is shown in **Table 1**.

In practice, these levels are, of course, subject to tolerances, so that the voltage difference may reach the maximum permissible level shown in the last row.

The specifications in ISO11519-2 (CANL) are slightly different, but since ISO11898 may be used for both high-speed and low-speed, that specification is invariably used nowadays.

Users need not concern themselves over the construction of a send/receive link since most manufacturers have available ready-made ICs for this purpose. These are optimized particularly as regards electromagnetic compatibility (EMC), board space and thermal overload (in case of a short-circuit of CANH or CANL), and output standard CAN signal levels. All that is necessary to establish a CAN link is for them to be coupled to the bus line.

All that is necessary is to make sure to which standard the IC is built: ISO11519-2 or ISO11898: the latter should be preferred. It should be noted that there are also other differential voltage procedures that may be used in CAN signal transfer, for instance, RS485.

Final questions to be asked about the CAN bus system are

- What is the maxi8mum bus expansion for a given data transfer rate?
- How many elements (stations) may be connected to the bus?

The answers to these questions depend solely on the bus medium used. **Table 2** shows the correlation between the data transfer rate, length of the bus, bus medium, and the bus termination impedance.

The data transfer medium should preferably be a twisted-pair cable with a cross-sectional area of 0.34–0.6 mm$^2$, while the bus termination impedance should be around 127 Ω. The resistivity of the cable should be not greater than 60 mΩ/m, a condition that is met when the cross-sectional area is greater than 0.30 mm$^2$.

Care should be taken with the length of branching lines in case the station is not connected directly to the CAN bus. These lines should not be longer than 2 metres when the data transfer rate is 250 kbit/s, and not longer than 30 cm when the data transfer rate is greater. The total length of all branching lines should not exceed 30 metres.

Finally, it should be noted as regards Layer 1 that the connectors and their pinout for linking elements to the bus are standardized.

[990060]

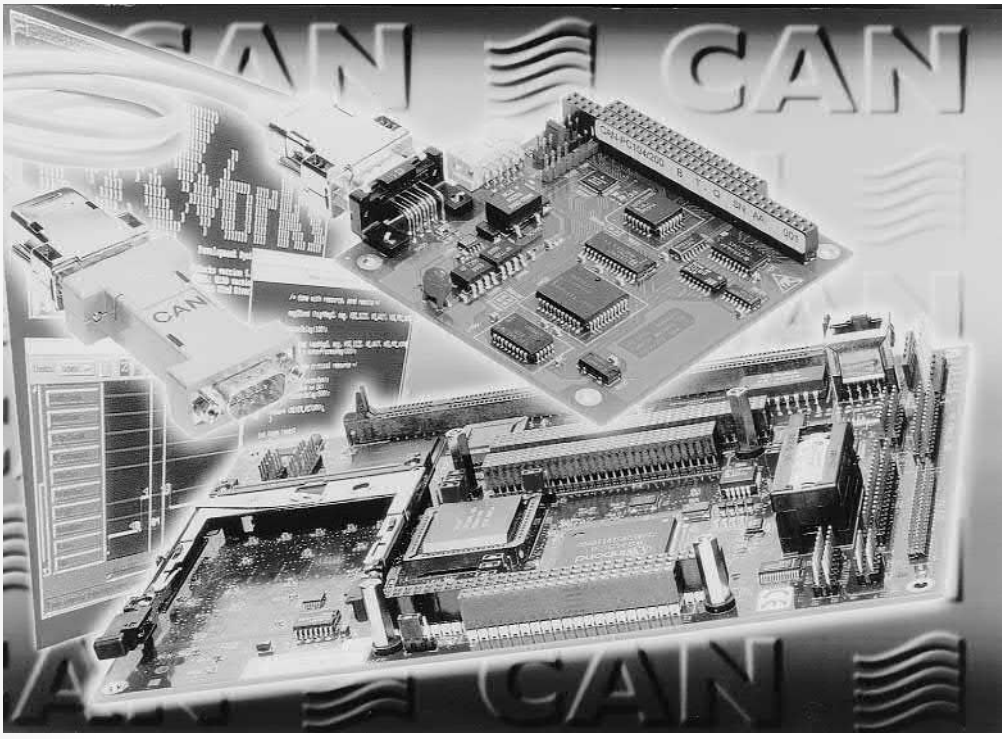
***Table 2. Correlation between data transfer rate, length of the bus, bus medium, and bus termination impedance.***

| Bus length | Bus cable | | Bus termination resistance | Maximum data rate |
|---|---|---|---|---|
| | resistance | cable c.s.a. | | |
| 0 - 40 m | 70 mΩ/m | 0.25 - 0.34 mm$^2$ AWG23, AWG22 | 124 Ω (1%) | 1 Mbit/s at 40 m |
| 40 - 300 m | < 60 mΩ/m | 0.34 - 0.6 mm$^2$ AWG22, AWG20 | 127 Ω (1%) | 500 Kbit/s at 100 m |
| 300 - 600 m | < 40 mΩ/m | 0.5 - 0.6 mm$^2$ AWG20 | 150 Ω to 300 Ω | 100 Kbit/s at 500 m |
| 600 m - 1 km | < 26 mΩ/m | 0.75 - 0.8 mm$^2$ AWG18 | 150 Ω to 300 Ω | 50 Kbit/s at 1 km |

See also:
'CAN – the Controller Area Network' in the September 1992 issue (p. 56) of *Elektor Electronics*.

# controller area network (CAN)

## *intelligent, decentralized data communications Part 2*




The first part of this article described the history, standardization, and the basic setup of the Controller Area Network (CAN) developed by the Robert Bosch Company in Germany. In this second part, the attention is focused on the data transmission protocol that determines the capabilities and reliability of this automotive digital data system.

By B vom Berg & P Groppe

**INTRODUCTION**

As already stated in Part 1, CAN is a serial asynchronous communication protocol that connects sensors and actuators of electronic control stations in cars. Among its many functions is a digital data link. It is an asynchronous system because each station (also called 'node') synchronizes to messages of other stations on the leading edge of the first message bit and on subsequent leading edges throughout the rest of the message. The ability of any station to synchronize to another station is determined by the maximum differences in oscillator frequencies. Other factors are, for instance, bit duration, message duration and composition, and handshaking.

The most important parts of the network are the *physical layer*, comprising the topology of the network and the link to the bus, and the *data link layer*, which lays down how the data transmission medium is accessed, how a message is constructed (address, data, control and protection against errors) and how the data transmission protocol is structured.

**IN PRACTICE**

The exchange of messages between two network stations may take place in two basically different ways: station-

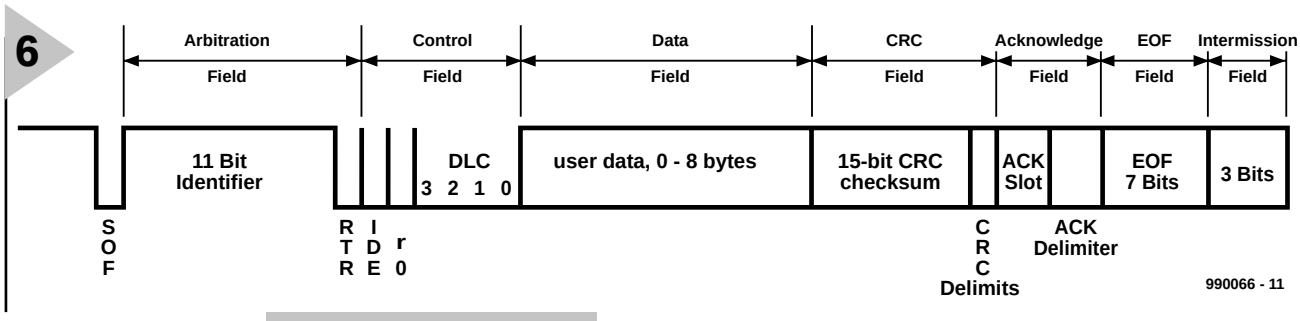



***Figure 6. Composition of a data frame (standard frame format – Specification CAN 20A).***

oriented and message-oriented.

*Station-oriented exchange*

In this mode, the sender addresses the receiver simply by means of the receiver address, for instance: 'Station 25 is sending a message to station 37'. In this way, a virtual link between the sender and receiver is established via the bus.

The transmitted packet of data therefore contains the address of the receiver station and that of the sending station. All other stations connected to the bus ignore the packet since it is not addressed to them.

The receiving station evaluates the message and normally acknowledges its receipt. In case of an error during the data transmission (no acknowledgment from the receiver), the sender repeats the message.

*Message-oriented exchange*

In this mode, the sender adds to the message an unambiguous identifier and sends the message and identifier via the bus, for instance: 'Station A is sending a voltage measurement with identifier 978'. In this mode, the addresses of the sender and receiver are not included.

Such a message is clearly intended for several receivers connected to the bus under the motto: 'Take from the bus what you need' (broadcast principle). The various receive stations must determine, on the basis of their programming, whether the message is relevant to them or not.

*Flow of communications*

The flow of communications between the individual stations connected to the CAN bus takes place in the form of a broadcast of event-controlled, prioritized (through-numbered) messages or frames (communication messages).

*Dominant and recessive bus/bit states*

The actual data transmission via the data transmission medium does not take place, as usual, in the form of '1s' and '0s', but by dominant and recessive bits. Recessive typifies a bus state that may be overwritten by a dominant bus state. So, when a station connected to the bus sends a recessive bit and another station at the same time sends a dominant bit, the dominant bit takes priority over the recessive bit, that is, the dominant state is accepted by the entire bus. The assignment of logic states to the bus is generally so that a logic 0 repretransmitteds a dominant state, and a logic 1, a recessive state.

These arrangements constitute one of the foundations of the CAN specification and will be elaborated on later.

*Data packets*

The network uses four kinds of data packets, normally called frames, to exchange data on the bus: data frame; remote frame; error frame; and overload frame.

*Data frame*

The data frame is used by the stations to send their data in line with their programming. The composition of a typical data frame, which consists of a single field, is shown in **Figure 6**. This is a standard frame format according to Specification CAN 20A. The meaning of the various terms in the figure is as follows.

**SOF**. This is the start-of-frame bit, which is always dominant (0). All stations connected to the bus synchronize their internal receive stages to the trailing edge of this bit.

**Arbitration field**. This field, which is 12 bits long, contains the data for accessing the bus.

**11-bit identifier**. This section contains the identifier (ID) of the transmitted frames. The 11 bits allow up to $2^{11} = 2048$ different identifiers to be constructed, of which only 2032 are freely available: the remaining 16 are reserved for certain special functions. This means that a single controller area network can process 2032 different messages (measurement values, switch positions, light functions, and so on). Although this seems a fairly large number, in many applications it is not enough. Therefore, an Extended Frame Format with 29 identifier bits (CAN 20B) has been formulated. In this, $2^{29} = 536\,870\,912$ frames can be handled.

**RTR** (Remote Transmission Request) bit. This bit, which is always dominant (0), enables a station to address and send messages to another specified station. This is of great value when certain data are urgently needed to be processed (more about this later).

**Control field**. This 6-bit long section contains the information as to how a data frame is composed.

**IDE** (Identifier Extension) bit. This bit indicates whether a standard-format frame with an 11-bit identifier (IDE = dominant = 0), or an extended-format frame with a 29-bit identifier (IDE = recessive = 1) is being transmitted.

**r0** (Reserve bit 0). This dominant bit is transmitted as a spare bit for future expansion specifications.

**DLC** (Data Length Code). This 4-bit long section indicates how many data bytes are being transmitted successively in a data field. The CAN Specification allows data field lengths of 0–8 bytes, that is, a single data frame may transmit not more than eight data bytes.

**Data field**. This 8-bit long section contains the data bytes (0–8) to be transmitted.

**CRC field**. The 16-bit long CRC field contains additional information for protecting the data being transmitted against interference. For this purpose, the sender station constructs, according to specific rules, a 15-bit CRC check sum from the preceding data and sends this, together with the frames. The receiver station calculates a similar check sum according to the same rules and compares this with the transmitted check sum. If the two sums are identical (the normal case), data transmission can commence. If the sums are not identical, an error handling procedure is initiated. The CRC field is limited by a CRC delimiter bit which is always transmitted recessively.

**Acknowledge field**. The 2-bit long acknowledge field serves to transmission acknowledgments of correctly received data frames.

**ACK slot**. This 1-bit long section is transmitted as a recessive bit and may, therefore, be overwritten by a dominant bit transmitted by another station connected to the bus. It allows receive stations to send an acknowledgement of a correctly received data frame. The acknowledgment bit is dominant and is transmitted by each and every relevant station upon error-free reception of messages. Since it is dominant, it overwrites the recessive bit sent by the transmitting station. Thus, if the transmitting station receives a dominant bit during the ACK slot window, instead of its own transmitted recessive bit, it 'knows' that at least one station has received the message.

The ACK slot window is restricted by a recessively transmitted ACK Delimiter bit.

**EOF** (End Of Frame) **field**. This field consists of seven recessive bits and serves to terminate the data frame.

Before the next data frame can be transmitted, the receive stations need a short intermission to enable them to process, or at least store, the received data. The intermission is arranged by a recessive 3-bit intermission field ending the data frame.

Owing to lack of space, the Extended Frame Format cannot be discussed; its principles are, however, the same as those discussed for the Standard Frame Format.

## AVOIDANCE OF CONFLICT

Since all stations connected to the Controller Area Network bus, two questions arise:

* What happens when several stations want to send a message at the same time?
* How is it decided which station can start and which stations must wait their turn?

Clearly, these matters may give rise to conflicts and to avoid those there is a special bus access procedure which must be obeyed by all stations when they want to send a message. In this, an important role is played by the dominant and recessive bits in the Arbitration Field.

Basically, each sender 'hears' its own transmission to the bus: it sends a bit, receives it back and compares the two. If they are identical, transmission of the message is allowed. If, however, the two bits are dissimilar, there is a problem. As explained earlier, a recessive bit (1) can be overwritten by a dominant bit (0).

**Figure 7** gives a highly simplified repretransmittedation of some linking stages of the bus. Basically, these are open-collector output stages that are arranged as wired-AND gates. With reference to station 1, a recessively transmitted bit (1) ensures that transistor $T_1$ remains cut off. This means that the recessive level is pretransmitted at the bus. After this bit has been transmitted, station 1 reads the bus status and recognizes the bit it has transmitted. If then a dominant bit (0) is transmitted, $T_1$ comes on and switches the bus line to earth. The bus line is then dominant (0). Again, station 1 reads back the bit it transmitted.

Considering the three stages, if one of them sends a dominant bit, the busline becomes dominant (0) and the other stations read this level.

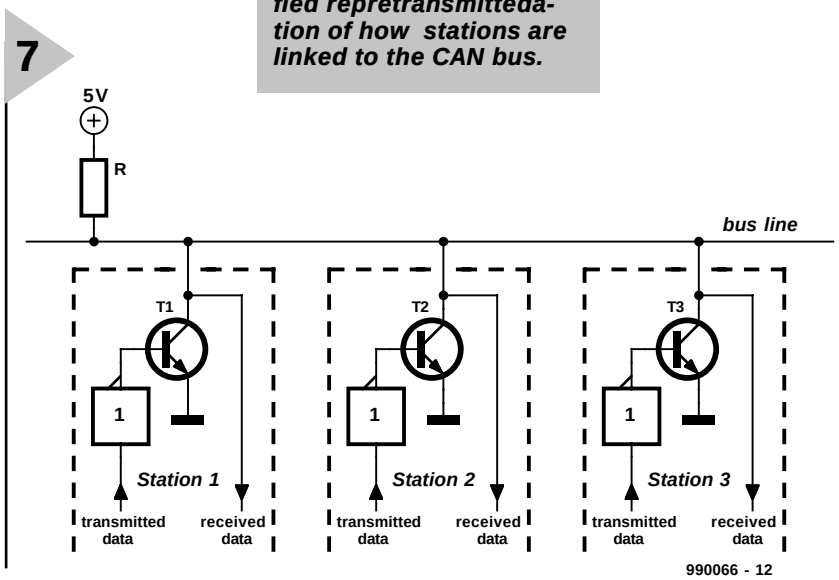
An example will show how the bus access procedure takes place. Assume that the stations in Figure 2 are all ready to transmit their data frames with three different identifiers:

Station 1: identifier 367
Station 2: identifier 232
Station 3: identifier 239.

All three start with the arbitration (bus access) phase by transmitting a SOF bit (see Figure 8). This is a dominant bit, and each station reads back its own (correct) bit from the bus. Then, the identifiers are transmitted. Up to time b, all stations send a dominant bit and all is well. At time c, there is still no problem. At time d, station 1 sends a recessive bit, but stations 2 and 3 continue with a dominant bit. When reading back, station 1 notices that its recessive bit has been overwritten, which means that it has lost access to the bus to at least one other station. Station 1 then assumes the receive mode (but tries to send its message at a later time again). Stations 2 and 3 continue as before.

At time j, station 3 sends a recessive bit that is promptly overwritten by the dominant level transmitted by station 2. This is noticed by station 3, which thereupon also assumes the receive mode (and, like station 1, tries to send its message at a later time again). Station 2 is the 'victor' and can send its message without further hindrance to the bus.

***Figure 7. Highly simplified repretransmittedation of how stations are linked to the CAN bus.***




A closer look at the identifiers shows that it is the station with the smallest identifier that gains access to the bus first: it has the highest send priority. In other words: the identifier also automatically contains the message priority. A message with identifier 0 will always be the first to be received by the stations connected to the bus, since it has the highest priority. A message with identifier 2032 has a long wait since it has the lowest priority.

**Remote Request Frame**

The remote request frame is an important one in the network. Assume that station D connected to the CAN bus transmits three temperature measurement data every five minutes with Identifier 598. This means that the data field contains three bytes. These messages are received and processed by other stations.

However, station G urgently needs the actual temperature measurement and cannot in any circumstances wait for five minutes. It has the facility,

therefore, to request the measurements directly from station D, that is, it can bypass the data transmission cycle. To do so, it sends a so-called Remote Request Frame, which is composed similar to a Data Frame (Figure 6), but with some small differences:

- The identifier of the station to whom the request is transmitted (here, 598) is entered in the identifier field.
- In the DLC field, the number of useful bytes contained in the requested message (here, 3) is entered.
- The Remote Transmission Request (RTR) which is dominant (0) in the Data Frame is made and transmitted recessively (1). This is a typical identification of a station that requests data direct from another specific station.
- There is no data field in the Remote Request Frame: the DLC field is followed immediately by the CRC field. In other words, the Remote Request Frame is composed like a Data Frame but with 0 bytes of data.

The transmitted Remote Request Frame functions as follows. All stations connected to the bus receive the frame and recognize by the set RTR bits that a station has requested specific data from another station. Station D recognizes that the identifier in the Remote Request Frame is the same as its own identifier and immediately sends its response in the form of a Data Frame with the requested data.

## ERROR DETECTION AND REMEDIES

One of the most striking properties of the Controller Area Network concept is its uncanny capability of detecting a multitude of errors during the data transmission and react to them accordingly. It has a Hamming Distance (also called signal distance) of 6. The signal distance between two binary words of the same length is the number of the corresponding bit positions in which the two words have different bit values. For instance, the signal distance between 11011010 and 10000110 is four, since the 3rd, 4th, 5th, and 7th bits (counting from the left) are different.

In a CAN data are transmitted permanently at a transmission rate of 500 kbit/s. Every 0.7 s a one-bit error is caused by external interference. The network operates eight hours a day, 365 days a year. The built-in protection against errors in a CAN guarantees that in 1000 years of operation only one error will not be detected. Errors can and do, of course, occur, but once they are known, they can be remedied. Only unknown errors can cause false measurements to be processed.

## DETECTION OF TRANSMISSION ERRORS

In a CAN, several means are used simultaneously to detect errors.

*Bit error detection*

Each and every station receives its own transmission back. If, therefore, after the arbitration phase, a station is the only one that sends a message to the bus and it receives back a different bus status than it transmitted, it is clear that an error has occurred on the bus. The station then shifts its operation to an error treatment routine (see later).

*Stuffbit error detection*

The CAN specification states clearly that when in a data frame more than five bits of the same value are transmitted in sequence (for instance, seven times a 0 in a field), each and every group of five bits is followed by a complementary bit (here, a 1, of course). This introduced bit, which, of course, contains no information whatever, is called a stuffbit. At the receive end, these bits are removed from the data stream, so that only the original message is processed.

The stuffbits may readily be used for error checking. If the receiver detects more than five sequential bits of the same value in a frame (but not in the EOF field), it is clear that this cannot be right and that an error in the data transmission has occurred which has inverted one or more bits. The receiver then shifts its operation to an error treatment routine (see later).

*CRC error detection*

This consists, as already described, of an evaluation of the CRC check sum at the receiver. When the received and calculated check sums are dissimilar, the receiver shifts its operation to an error treatment routine (see later).

*Acknowledgment error detection*

In the description of the frame format (see Figure 6) mention was made of the ACK slot bit, which is transmitted by a station as a recessive bit. All stations that have received the previous frame correctly overwrite this bit with a dominant bit. The sender detects this and 'knows' that at least one station has received its data correctly.

If the sender detects that its ACK slot bit is not overwritten, it 'knows' that not one station has received its message correctly. It then shifts its operation to an error treatment routine (see later).

*Format error detection*

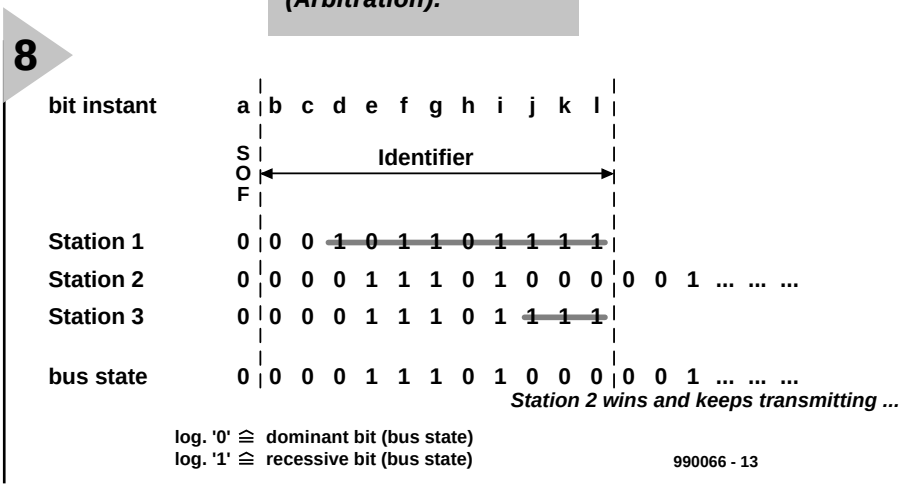
In this, use is made of the fact that the network format has several fields that must always have a fixed content: the CRC delimiter, the Acknowledgment Delimiter, and the EOF field are always composed of recessive bits. If a dominant bit is detected, this can have been caused only by a data transmission error. Here also, operation is shifted to an error treatment routine (see later).

## ERROR TREATMENT

The error treatment routine in response to a data transmission error takes two forms.

In the first place, frames in which an error has been detected are immediately rejected by the relevant station and not processed. Secondly, if any station within the system detects an error, it immediately transmits an error frame that consists of six dominant bits (= error flag) and an error delimiter of eight recessive bits. The result of this is that all recessive bits on the bus are overwritten, so that six dominant bits remain. This is, however, a contravention of the stuffbit rule that not more than five sequential bits may have the same value.

***Figure 8. Diagrammatic repretransmittedation of how a station does access the bus (Arbitration).***



All other stations connected to the bus detect this error condition and treat the frame just transmitted as faulty, reject it and also send out an error frame. In other words, a station that detects an error purposely mutilates the entire transmitted frame so that all other stations connected to the bus receive a faulty frame. This means that an error local to a station is immediately communicated to all other stations. The motto of the network is that all stations receive correct data that can be processed as required, or all stations receive faulty data that are rejected. The original sender detects, of course, that the frame it transmitted is mutilated, adjust its message and resends it after a short while.

## ERROR INSIDE A STATION

What happens when a station itself becomes defect, is damaged, operates with an inaccurate transmission rate, or is the only station that gets interference? Such a station would permanently send out error frame and so disable the entire network. The CAN concept has adequate protection against such an occurrence, but space prohibits describing this in this article.

**Table 3.**

| | CAN 20A | CAN 20B |
|---|---|---|
| Maximum number of identifiers | $2^{11}$ | $2^{29}$ |
| Number of stations (nodes) | 32 | 32 |
| Data transfer rate | 5–125 kbit/s | 5–1000 kbit/s |
| Number of permissible bytes per frame | 0–8 | 0–8 |
| Maximum length of a frame | 117 bits | 13 bits |
| Maximum bus expansion | see text | see text |

**Table 3. Comparison of CAN 20A (standard frame format) and CAN 20B (extended frame format).**

## SUMMARY

The specifications of the two CAN versions, CAN 20A (standard frame format) and CAN 20B (extended frame format) are compared in **Table 3.**

Although the Controller Area Network is a powerful and highly reliable system for data communications, the reader and prospective user may well ask how it can be turned into a practical application. There are dominant and recessive bits, an 11-bit identifier, a 15-bit CRC check sum, a 1-bit delimiter, a 7-bit EOF field, a 6-bit error frame, and many more. None of this resembles the 8-bit or 16-bit data structure of the microcontroller.

So how is it possible to program according to the network protocol? Here, the future constructor need not worry. There is a plethora of ready-made, inexpensive building blocks available for the network. It is this support by IC manufacturers for the CAN that has made the network so popular in such a short time.

The next instalment will deal with these building blocks, with the programming according to the CAN protocol and with practical application of the network.

[990066]

## CONSTRUCTION GUIDELINES

Elektor Electronics (Publishing) does not provide parts and components other than PCBs, fornt panel foils and software on diskette or IC (not necessarily for all projects). Components are usually available form a number of retailers – see the adverts in the magazine.

Large and small values of components are indicated by means of one of the following prefixes :

| | |
|---|---|
| E (exa) = $10^{18}$ | a (atto) = $10^{-18}$ |
| P (peta) = $10^{15}$ | f (femto) = $10^{-15}$ |
| T (tera) = $10^{12}$ | p (pico) = $10^{-12}$ |
| G (giga) = $10^{9}$ | n (nano) = $10^{-9}$ |
| M (mega) = $10^{6}$ | µ (micro) = $10^{-6}$ |
| k (kilo) = $10^{3}$ | m (milli) = $10^{-3}$ |
| h (hecto) = $10^{2}$ | c (centi) = $10^{-2}$ |
| da (deca) = $10^{1}$ | d (deci) = $10^{-1}$ |

In some circuit diagrams, to avoid confusion, but contrary to IEC and BS recommandations, the value of components is given by substituting the relevant prefix for the decimal point. For example,

3k9 = 3.9 kΩ  4µ7 = 4.7 µF

Unless otherwise indicated, the tolerance of resistors is ±5% and their rating is ⅓–½ watt. The working voltage of capacitors is ≥ 50 V.

In populating a PCB, always start with the smallest passive components, that is, wire bridges, resistors and small capacitors; and then IC sockets, relays, electrolytic and other large capacitors, and connectors. Vulnerable semiconductors and ICs should be done last.

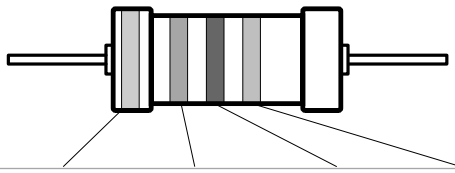
Soldering. Use a 15–30 W soldering iron with a fine tip and tin with a resin core (60/40) Insert the terminals of components in the board, bend them slightly, cut them short, and solder: wait 1–2 seconds for the tin to flow smoothly and remove the iron. Do not overheat, particularly when soldering ICs and semiconductors. Unsoldering is best done with a suction iron or special unsoldering braid.

Faultfinding. If the circuit does not work, carefully compare the populated board with the published component layout and parts list. Are all the components in the correct position? Has correct polarity been observed? Have the powerlines been reversed? Are all solder joints sound? Have any wire bridges been forgotten?

If voltage levels have been given on the circuit diagram, do those measured on the board match them – note that deviations up to ±10% from the specified values are acceptable.

Possible corrections to published projects are published from time to time in this magazine. Also, the readers letters column often contains useful comments/additions to the published projects.

The value of a resistor is indicated by a colour code as follows.



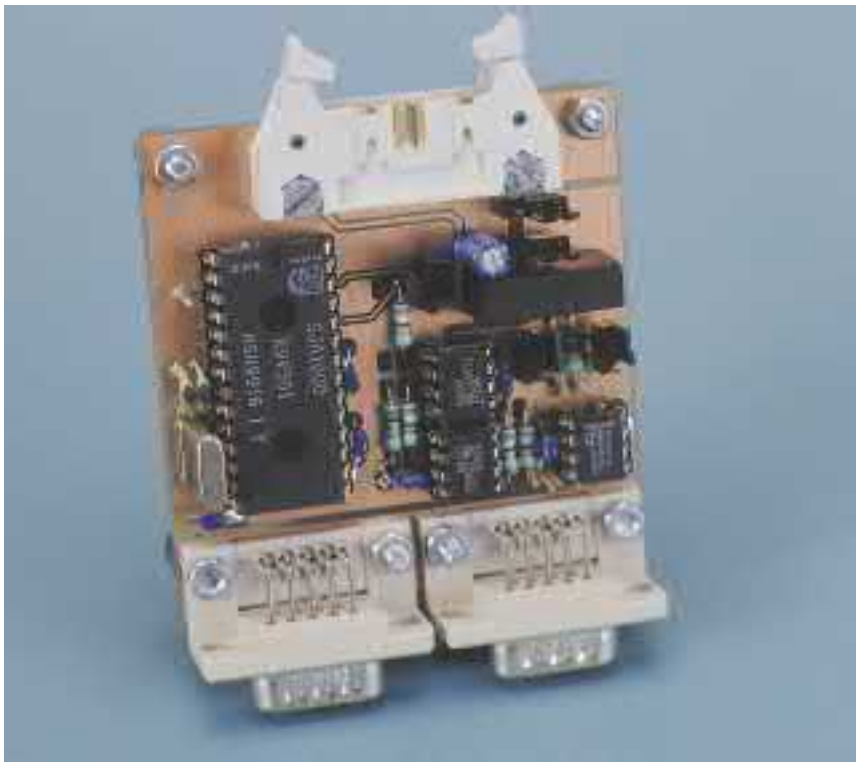
| color | 1st digit | 2nd digit | mult. factor | tolerance |
|---|---|---|---|---|
| black | – | 0 | – | – |
| brown | 1 | 1 | ×$10^{1}$ | ±1% |
| red | 2 | 2 | ×$10^{2}$ | ±2% |
| orange | 3 | 3 | ×$10^{3}$ | – |
| yellow | 4 | 4 | ×$10^{4}$ | – |
| green | 5 | 5 | ×$10^{5}$ | ±0,5% |
| blue | 6 | 6 | ×$10^{6}$ | – |
| violet | 7 | 7 | – | – |
| grey | 8 | 8 | – | – |
| white | 9 | 9 | – | – |
| gold | – | – | ×$10^{-1}$ | ±5% |
| silver | – | – | ×$10^{-2}$ | ±10% |
| none | – | – | – | ±20% |

Examples:
brown-red-brown-gold = 120 Ω, 5%
yellow-violet-orange-gold = 47 kΩ, 5%

# controller area network (CAN)

## *intelligent, decentralized data communications in practice: Part 3*

The first two parts of this article dealt with the history, standardization, basic setup, and data transmission protocol of the Controller Area Network. In this third part, the attention is shifted to more practical aspects. It deals with a network interface bus design that can be connected to any current microcontroller system.



By B vom Berg & P Groppe

**INTRODUCTION**

Current Controller Area Network (CAN) interfaces consist basically of three chips as shown in the block diagram in **Figure 9**. All the micro-controller is required to do is to write the data bytes (0–8) to be transmitted into the CAN protocol IC, fill the identifier field and the DLC field, and set the RTR bit accordingly. The remainder of the process:

- computing the CRC check sum;
- adding the remaining fields;
- accessing the bus;
- transmitting the data;
- detecting and remedying errors

is effected by the CAN controller IC.

The data are applied to the bus via the CAN transceiver IC that provides direct coupling to the bus.

The microcontroller then receives a message confirming the successful transmitting of the data or an error message, following which requisite

action can be taken.

More or less the same happens when data.are being received. The CAN controller receives the CAN frames from the bus via the CAN transceiver IC, rechecks the check sum, removes all superfluous fields from the frame and passes either the received data or an error message to the microcontroller.

The reader will already have noticed that only a modest amount of material, hardware as well as software, is required for a CAN bus interface. Moreover, microcontrollers that contain a CAN controller on the same chip are already commercially available. Such a chip makes a two-stage CAN interface possible.

Some more information is given in the following section before the design of a practical interface can be discussed.

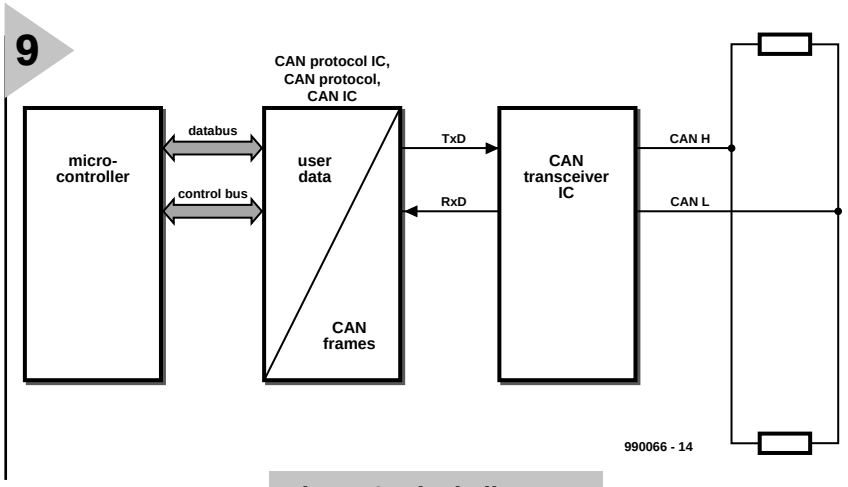



***Figure 9. Block diagram of a three-stage Controller Area Network design.***

## ACCEPTANCE FILTERING

We have seen in Part 2 that a Controller Area Network operating with standard frame format (CAN 20A) can process up to 2048 different identifiers. It is, of course, not necessary that each and every station linked to the bus can receive all data/remote frames. For instance, it may well be that for, say, Station K only frames with the identifiers 129, 1345, and 1999, are of interest and the other 2045 are of no consequence whatever. To avoid Station K receiving and processing all identifiers and passing them on to the microcontroller (which has to recheck whether each and every identifier is to be accepted or rejected – a time consuming activity), some kind of selection filtering of identifiers to ensure that only those of interest are passed to the microcontroller is highly desirable.

The selection of identifiers is called acceptance filtering. It allows the CAN controller chip to be programmed so that only frames with certain identifiers are passed to the microcontroller. All other frames are received and checked (incl. error correction), but not passed on. In this way, the microcontroller is freed of many superfluous comparisons and can therefore more speedily process the required data/remote frames.

There are two ICs that can be used for acceptance filtering.

### *BasicCAN*

This IC has a simple filter that is, say, eight bits wide, which allows coarse pre-selection only. Normally, this consists of passing groups of identifiers, in, say, the range 700–707. Selection of a single identifier is not possible. This IC therefore requires the microcontroller to carry out a further selection to arrive at the wanted identifier. Remote frames intended for the relevant station are also passed by the filter and applied to the microcontroller. It is only then that the microcontroller can generate the relevant response data and pass these to the CAN controller.

### *FullCAN*

The FullCAN chip allows the exact programming and selecting of a single identifier. In other words, it can be set to accept a single frame or a number of single frames, for instance, only those with identifier 798.

The drawback of the IC is, however, that a large number of frames (with different identifiers) are not passed on since the controller programme is fixed.

Therefore, if many frames with different identifiers are to be received it makes better sense to use a BasicCAN chip. It must be borne in mind, however, that in this case the microcontroller needs to carry out a substantial part of the selection process and this in turn means that a more powerful microcontroller is needed.

A beneficial property of the FullCAN chip is that the microcontroller can program the response to a remote frame on to the CAN controller IC. When this IC receives a permissible

### *Table 4: Brief parameters of interface*

| | |
|---|---|
| *CAN controller IC* | *Type SJA1000 (Philips Semiconductors)* |
| *Microcontroller interface* | *can be arranged for Intel or compatible and Motorola or compatible microcontrollers* |
| *Mode of operation 1:* | *pin-, hardware-, and software-compatible with PCA82C200*<br>*CAN 20A and 20B passive*<br>*Standard frame format*<br>*Data transfer rates up to 1 Mbps*<br>*Acceptance filter BasicCAN* |
| *Mode of operation 2:* | *Extended and standard frame format*<br>*Data transfer rates up to 1 Mbps*<br>*CAN 20B capability*<br>*Extended acceptance filter with BasicCAN properties* |
| *Transceiver IC* | *Compatible with ISO/DIS11898, high-speed CAN*<br>*Data transfer rates up to 1 Mbps*<br>*Internal protection against noise specific to motor vehicles*<br>*Internal protection against short-circuits and thermal overload*<br>*Non-powered nodes (stations) do not affect the bus*<br>*Allows the design of Controller Area Networks with up to 110 nodes* |

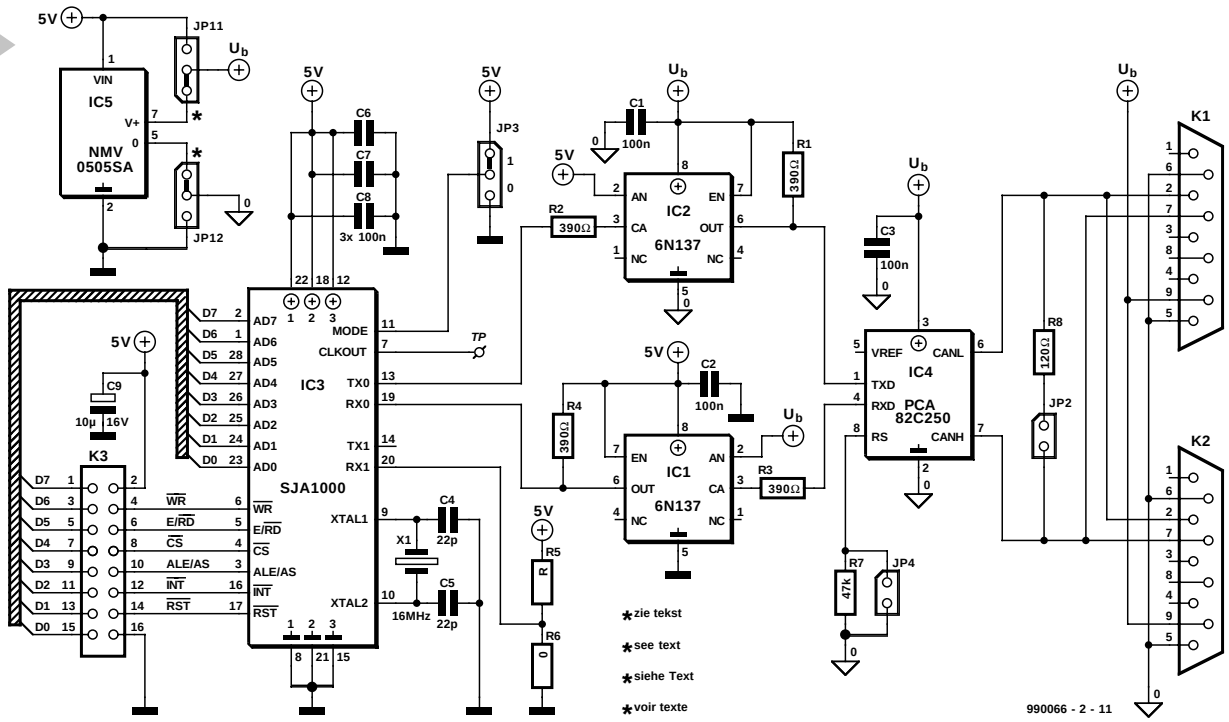



***Figure 10. Circuit diagram of the CAN bus interface.***

remote frame for the relevant station, it can send the response data frame without intervention by the microcontroller.

With the inexorable progress of the technology, the differences between the BasicCAN and FullCAN chips are becoming less defined. Also, FullCAN chips are becoming more powerful, so that they can select larger numbers of individual identifiers and can store more data registers. State-of-the-art CAN controller chips can switch between the two modes of operation by means of software.

## COMPATIBILITY BETWEEN 20A AND 20B

As we have seen during the discussion of the frame formats (Part 2), there is a Standard Format with 11-bit identifiers and an Extended Format with 29-bit identifiers. Great care must be taken in choosing a CAN controller when both formats are used in a bus system (which is perfectly possible and permissible).

*Controllers with 20A capability*

These controller chips can process standard frames only and generate an error message when an extended frame message is received. Since this could bring the whole system to a standstill, these controllers can be used only in systems that operate with standard frames.

*Controllers with 20A capability and 20B passive properties*

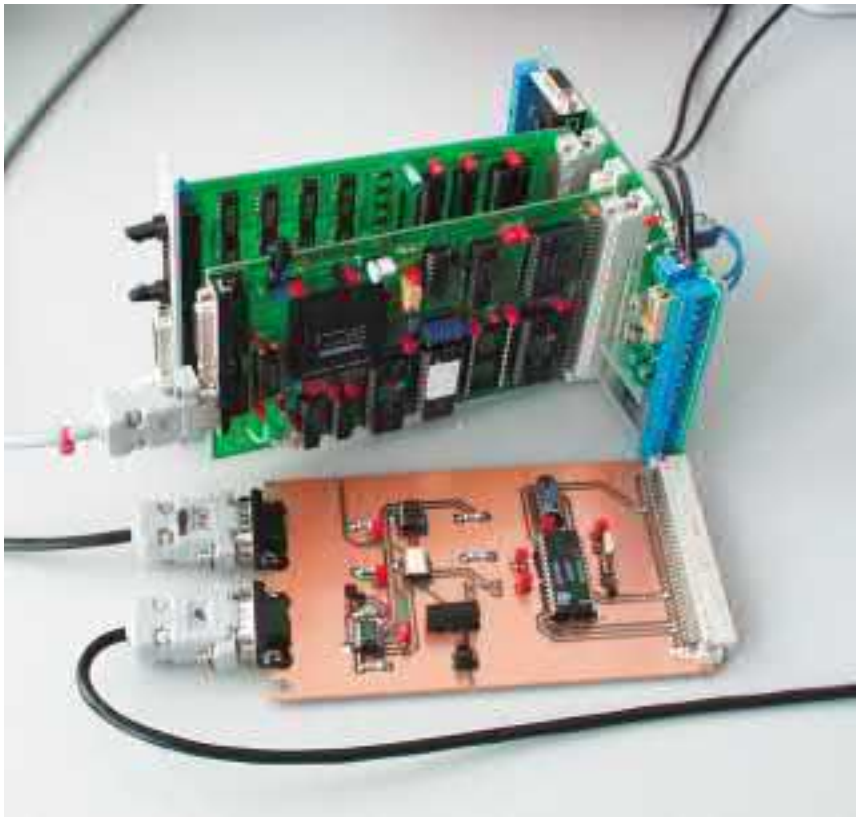
These chips accept extended frames with 29-bit identifiers, carry out an error test, and respond with and ACK bit or an error frame.

Although the communication is not disturbed, the received extended-frame data are not stored or passed on, since the chips are intended for processing frames in standard format only. Nevertheless, these controllers are perfectly all right for use in hybrid systems.

*Controllers with 20B capability*

These controllers process, store, and pass on, standard-format as well as extended-format frames.

When a decision has to be made on the purchase of a CAN controller or microcontroller with on-chip CAN controller, there is such a wide choice



that it really pays to visit the Internet Home pages of producers like Hitachi, Intel, Motorola, NSC, Philips, SGS, Siemens, and Temic, and Texas Instruments to name but a few.

## CAN BUS INTERFACE

After the lengthy discussions on theory and basic principles, the practical design of a CAN bus interface can now be described in relatively few words.

The circuit diagram of the interface is shown in **Figure 9** and a suitable printed-circuit board in **Figure 10**. Brief parameters of the interface are given in **Table 4**.

The CAN controller, $IC_3$, is a Type SJA1000, whose internal block schematic is shown in **Figure 11**. This chip is the successor of the PCA82C200 with which, in operating mode 1, it is pin-, hardware-, and software-compatible.

The interface with the microcontroller can be arranged for use with either a Motorola or an Intel chip (or compatible types).

The CAN transceiver, $IC_4$, is a Type PCA82C250.

The microcontroller is linked to the CAN bus interface by a length of flat-cable, which should be not longer than 10 cm (4 in), terminated into 16-pin header $K_3$. The pin connections of this connector are given in **Table 5**.

Via this link the microcontroller exchanges operating data, control data, and status data, with the CAN controller. These data are processed by the controller both in the send and receive directions. The microcontroller therefore 'sees' the CAN controller as an extension to its memory to which it writes operating data to be transmitted, or from which it extracts received operating data.

The clock frequency, which is divided in several stages, can be measured at test pin TP, for example, when it is to be ascertained whether the controller can be accessed and programmed safely.

The serial output signal at pin 13 of the controller is applied to pin 1 of the transceiver via optoisolator $IC_2$. The transceiver generates the standard CAN bus signals which are available at its pins 6 and 7. These signals are impressed upon unshielded twisted-pair (UTP) copper wires via connectors $K_1$ and $K_2$.

The signal received from the bus arrives at pin 4 of the transceiver from where it is applied to pin 19 of the controller via optoisolator $IC_1$. The controller converts the received bits and processes them in accordance with the relevant CAN protocol. The received signal is finally fed to the microcontroller where it is analysed.

Optoisolators $IC_1$ and $IC_2$, and 5 V DC/DC inverter $IC_5$, effectively isolate the microcontroller and bus sections of the node (station). The arrange-ment ensures that any faulty or uncertain signals on the UTP wires, although applied to the transceiver, cannot damage the microcontroller section and following system.

It is, of course, possible to build the interface without the isolating stages, so that $R_1$–$R_4$, $C_1$–$C_2$, $IC_1$–$IC_2$, $IC_5$, and $JP_1$–$JP_2$ can be omitted. The supply line terminals as well as the send and receive pins on $IC_3$ and $IC_4$ must then be interlinked as appropriate. It must be borne in mind, of course, that faulty

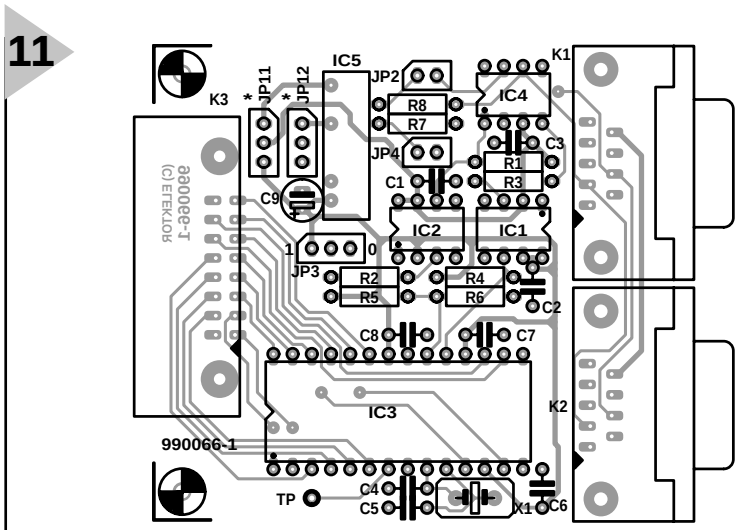
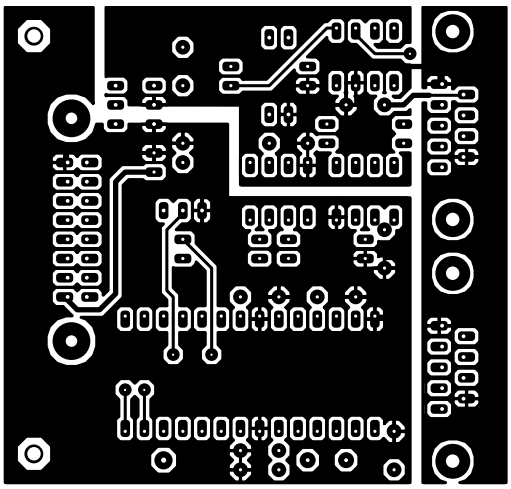



***Figure 11. Printed-circuit board for the CAN bus interface.***



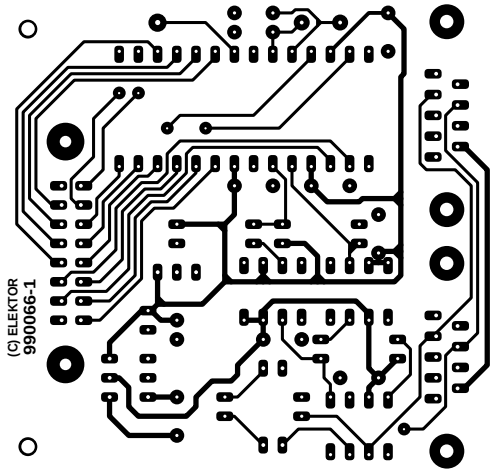
**Parts list**

**Resistors**:
$R_1$–$R_4$ = 390 Ω
$R_5$, $R_6$ = see text
$R_7$ = 47 kΩ
$R_8$ = 120 Ω

**Capacitors**:
$C_1$–$C_3$, $C_6$–$C_8$ = 0.1 µF, ceramic
$C_4$, $C_5$ = 22 pF, ceramic
$C_9$ = 10 µF, 16 V, radial

**Integrated circuits**:
$IC_1$, $IC_2$ = 6N137
$IC_3$ = SJA1000
$IC_4$ = PCA82C250
$IC_5$ = NMV505SA (Newport/Farnell)

**Miscellaneous**:
$X_1$ = 16 MHz quartz crystal
$K_1$, $K_2$ = 9-way D connector, right-angled, for board mounting
$K_3$ = 16-way header, right-angled, for board mounting, with interlock
$JP_2$, $JP_4$ = 2-way, 2.54 mm pin strip and pin jumper (Maplin)
$JP_3$, $JP_{11}$, $JP_{12}$ = 3-way, 2.54 mm pin strip and pin jumper (Maplin)
PCB Order no 990066-1 (see Readers' Services section toward the end of this issue)



| Table 5: Pin connections of header $K_3$ | | |
|---|---|---|
| Pin | Designation | Function |
| 1 | D7 | |
| 3 | D6 | |
| 5 | D5 | |
| 7 | D4 | |
| 9 | D3 | |
| 11 | D2 | |
| 13 | D1 | |
| 15 | D0 | |
| 2 | +5 V | positive supply line |
| 4 | WR\ | Write\ signal |
| 6 | RD\ | Read\ signal |
| 8 | CS\ | Chip-Select\ signal |
| 10 | ALE | Address latch enable signal |
| 12 | INT\ | Interrupt\ signal |
| 14 | RST\ | Reset\ signal |
| 16 | GND | Earth connection |

signals on the UTP lines are then applied unimpeded to the microcontroller section.

Jumpers $JP_{11}$, $JP_{12}$, and $JP_2$–$JP_4$ must be arranged as follows.

*$JP_{11}$, $JP_{12}$*
These jumpers (marked ∗ on the PCB) determine how the supply voltage is applied to the interface and the microcontroller system. With these jumpers set as shown in Figure 10, electrical isolation is provided and the supply voltage to the bus section ($IC_1$, $IC_2$, $IC_4$) is via $IC_5$.

In the other position of the jumpers, there is no electrical isolation, and all stages are supplied directly via the 0 and $+U_b$ terminals.

An alternative in the latter case is the provision of supply voltage via two lines in parallel with the UTP wires and connecting these to pins 6 and 9 of $K_2$ and $K_1$ respectively.

If the jumpers are left open, there is electrical isolation, and the bus section is then powered via pins 6 and 9 of $K_2$ and $K_1$ respectively.

*$JP_2$*
When $JP_2$ is shorted, bus terminating resistor $R_8$ is connected between pins 6 and 7 of the transceiver. It must be borne in mind that only two bus terminators must be used, one at the beginning and one at the end of the UTP wires. Since more terminating resistors (at other nodes) will be in parallel with these two, they lead to a reduction in total terminating resistance, which results in a higher output current from the transceiver and this in turn can lead to thermal overload and damage or destruction of the chip.

*$JP_3$*
This jumper must be set according to which microcontroller interface is used. The position shown in Figure 10 (pin 11 of $IC_3$ at +5 V) and marked '0' on the PCB allows Intel processors/controllers or compatible chips to be used.

When the jumper is placed in the other position (pin 11 of $IC_3$ at 0 V), Motorola processors/controllers or compatible chips may be used.

*$JP_4$*
This jumper, or rather resistor $R_7$, determines the slope of the edges of the pulses at the CAN bus.

In the case of high data transfer rates (up to 1 Mbps), steep-sloped edges are essential, but these lead to the risk of a large noise spectrum being generated by the CAN pulses. This noise can only be negated by the use of shielded twisted pair (STP) wires. The jumper must be used, thereby shorting out resistor $R_7$.

With low data transfer rates (up to 125 Kbps) the edges of the pulses need not be steep. This results in a narrow noise spectrum being caused by the CAN pulses, so that UTP (unshielded twisted pair) wires can be used. Jumper $JP_4$ must not be placed.

12

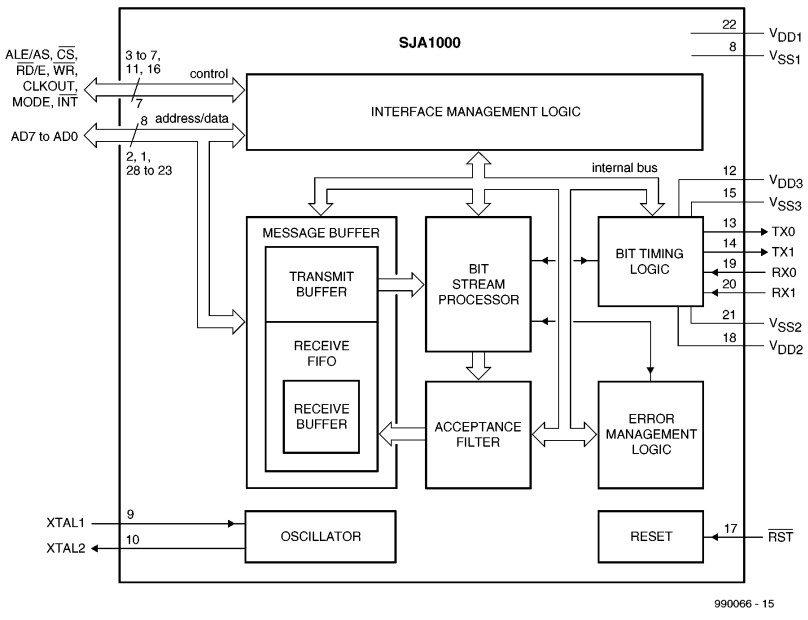



***Figure 12. Block diagram of the internal arrangement of the Type SJA1000 Controller Area Network IC.***

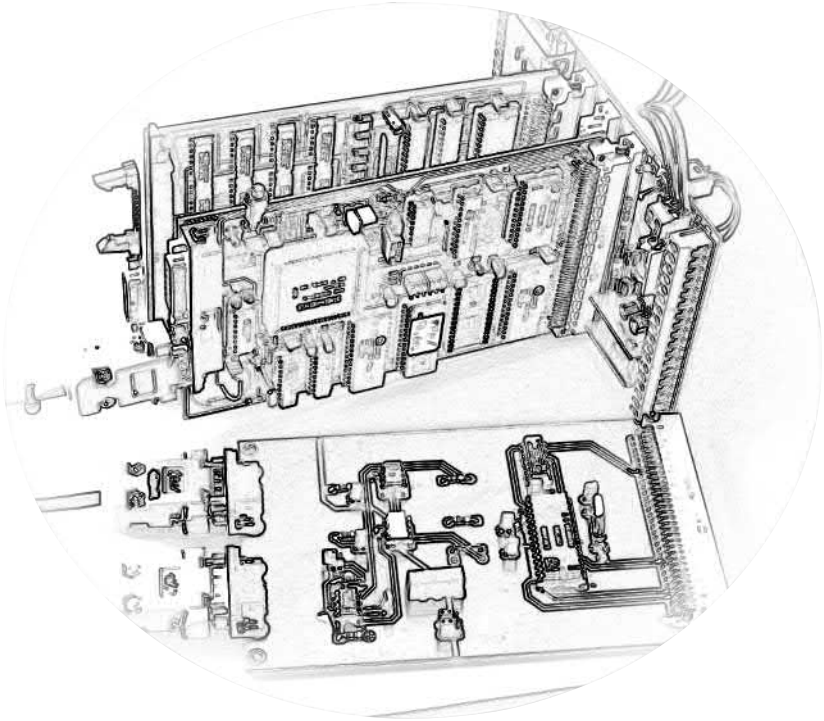
*This concludes the description of the hardware for the CAN bus interface. A forthcoming article will deal with the connection of the CAN bus to a microcontroller system and its application in an experimental CAN bus system.*

[990066]

# controller area network (CAN)

## *intelligent, decentralized data communications in practice: Part 4*

A Controller Area Network obviously consists of more than just the CAN bus interface described in last month's instalment of this article. In fact, the interface is merely the link between a microcontroller or computer and the CAN bus proper. The hardware described in detail last month clearly needs software for its proper operation and this is described in this fourth instalment of the article.



By B vom Berg & P Groppe

**INTRODUCTION**

Each station or node in a CAN bus system needs, apart from the CAN Bus Interface, a microcontroller or computer with appropriate software. Two sets of software are needed for taking the station into use, test it, and operate it: operating software and applications software.

The operating software is needed to provide an effective overall system, and also to test the interface in association with the microcontroller or computer. Such a test shows whether or not the drive from the microcontroller or computer functions correctly, whether or not the interface works correctly, both from a hardware and a software point of view, and whether the data is transferred correctly onto the CAN bus. The test thus enables a simple communication path between two or more nodes to be set up.

The applications software is specific to the particular role performed by the microcontroller or computer in the network. It therefore depends on what the station is going to be used for: logging of measurements; driving a display; transferring times and dates; and others.

Each node therefore needs its own particular software appropriate to its function. The sum of all the functions carried out by the various stations is the desired overall function of the net-

| CAN ADDRESS | SEGMENT | OPERATING MODE | | RESET MODE | |
|---|---|---|---|---|---|
| | | READ | WRITE | READ | WRITE |
| 0 | control | control | control | control | control |
| 1 | | (FFH) | command | (FFH) | command |
| 2 | | status | – | status | – |
| 3 | | interrupt | – | interrupt | – |
| 4 | | (FFH) | – | acceptance code | acceptance code |
| 5 | | (FFH) | – | acceptance mask | acceptance mask |
| 6 | | (FFH) | – | bus timing 0 | bus timing 0 |
| 7 | | (FFH) | – | bus timing 1 | bus timing 1 |
| 8 | | (FFH) | – | output control | output control |
| 9 | | test | test; note 2 | test | test; note 2 |
| 10 | transmit buffer | identifier (10 to 3) | identifier (10 to 3) | (FFH) | – |
| 11 | | identifier (2 to 0), RTR and DLC | identifier (2 to 0), RTR and DLC | (FFH) | – |
| 12 | | data byte 1 | data byte 1 | (FFH) | – |
| 13 | | data byte 2 | data byte 2 | (FFH) | – |
| 14 | | data byte 3 | data byte 3 | (FFH) | – |
| 15 | | data byte 4 | data byte 4 | (FFH) | – |
| 16 | | data byte 5 | data byte 5 | (FFH) | – |
| 17 | | data byte 6 | data byte 6 | (FFH) | – |
| 18 | | data byte 7 | data byte 7 | (FFH) | – |
| 19 | | data byte 8 | data byte 8 | (FFH) | – |
| 20 | receive buffer | identifier (10 to 3) | identifier (10 to 3) | identifier (10 to 3) | identifier (10 to 3) |
| 21 | | identifier (2 to 0), RTR and DLC | identifier (2 to 0), RTR and DLC | identifier (2 to 0), RTR and DLC | identifier (2 to 0), RTR and DLC |
| 22 | | data byte 1 | data byte 1 | data byte 1 | data byte 1 |
| 23 | | data byte 2 | data byte 2 | data byte 2 | data byte 2 |
| 24 | | data byte 3 | data byte 3 | data byte 3 | data byte 3 |
| 25 | | data byte 4 | data byte 4 | data byte 4 | data byte 4 |
| 26 | | data byte 5 | data byte 5 | data byte 5 | data byte 5 |
| 27 | | data byte 6 | data byte 6 | data byte 6 | data byte 6 |
| 28 | | data byte 7 | data byte 7 | data byte 7 | data byte 7 |
| 29 | | data byte 8 | data byte 8 | data byte 8 | data byte 8 |
| 30 | | (FFH) | – | (FFH) | – |
| 31 | | clock divider | clock divider; note 3 | clock divider | clock divider |

990066-3-13

***Table 6. Internal SFRs in the controller used in the basic CAN mode.***

work. In other words, the spatially distributed network can be controlled, driven and monitored to arrive at the final result.

## OPERATING SOFTWARE

The programming of the CAN controller is subject to the same general principles as that of other external peripheral units.

- The function of the controller is fixed or set by programming sets of data by internal Special Function Registers (SFR).
- These internal SFRs are interpreted by the microcontroller or computer as normal memory addresses in the external RAM range, to which data can be written or from which data can be read. This means that the microcontroller or computer is not aware that it is operating in conjunction with a CAN controller. Only access to specific memory locations is determinative for it as well as for the applications software.

When, therefore, the applications software for controller IC3 is being produced, the following points should be clarified or processed.

- Set the chip-select base address for the SJA1000 controller.
- Understand the internal setup of the structure of the SFRs in the controller.
- Create the routine for the basic initialization of the controller.
- Create the routine for applying data to the CAN bus.
- Create the routine for receiving data from the CAN bus.

The processing of relevant points for the basic CAN mode of the controller is looked at in some detail in the following paragraphs. Extensive and more detailed information can be found in the data sheets and application notes for the controller (see Reference at end of this article).

## SETTING THE CHIP-SELECT BASE ADDRESS

The chip is accessed via the chip-select base address. Since controller IC3 in the basic CAN mode needs a coherent external address range of 32 bytes, and in the PeliCAN mode one of 128 bytes, the maximum range is set at 128 bytes so as not to preclude the use of the PeliCAN mode in future operations.

The SJA1000 controller is enabled by a low level at its $\overline{CS}$ terminal (pin 3). This means that the microcontroller or computer must construct its address

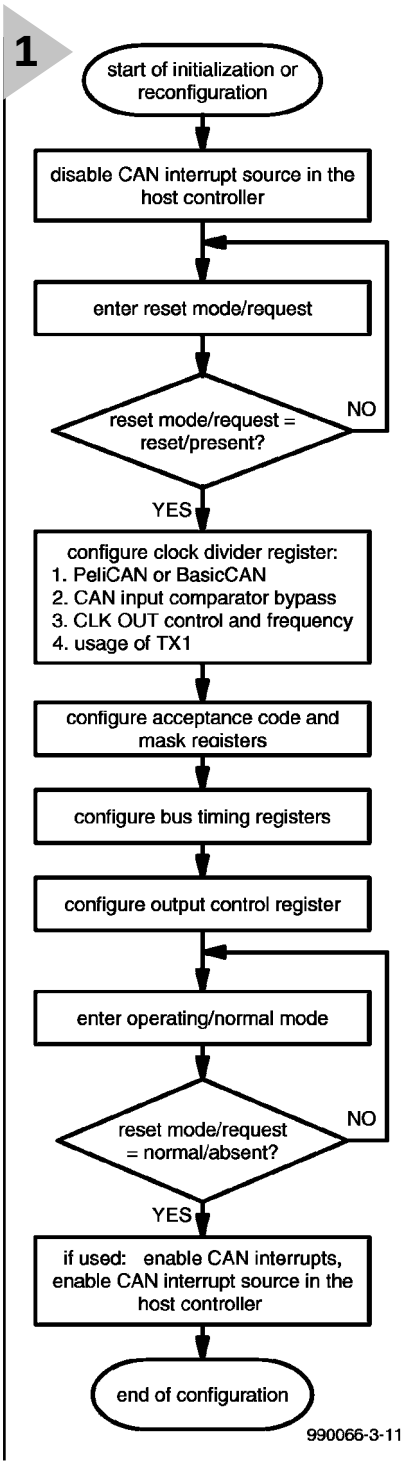



***Figure 13. Flow diagram for use when the CAN controller is to be initialized.***

coding in such a way that within a coherent address range of not fewer than 128 bytes a low signal is produced at pin 8 of connector K3 to enable the data transfer to be carried out by the CAN controller. The first address at which this is the case becomes the so-called **chip-select base address** of the controller. When the microcontroller or computer accesses a random memory location in this address range, it receives the byte content of an SFR in the controller or it can write a byte-word in an SFR of the controller.

In the following it is assumed that the chip-select base address of the SJA1000 controller is F000H.

## INTERNAL STRUCTURE OF AN SFR

The determinant SFRs of controller IC3 for operation in the basic CAN mode are shown in **Table 6**. The meaning of the various columns is as follows.

1. The first column, CAN address, gives the internal addresses of the relevant SFRs, to which only the chip-select base address of the IC needs to be added. If, for instance, the status register of the controller is to be accessed, F000H must be added to the internal address of the SFR, which is 2. If, therefore, a read or write operation on the register is desired, the software must be programmed to enable the external RAM location to be accessed with address F002H. The Clock Divider Register is from then on accessible at address F01FH (=F000H+31D – note the use of different number systems).

2. The second column shows the basic division of the SFRs into three different groups: the control group, the transmit buffer group, and the receiver buffer group.

3. The controller supports two software-controlled modes:

- **Operating mode**, which is the normal mode of operation;
- **Reset mode**, which is the mode IC3 is in when it is clearing a hardware reset or when the reset bit in the control register is set. The controller then reverts to the normal operating mode.

The reset mode is necessary when the controller is to be (re)initialized, that is, certain operating parameters can be set only in the reset mode. The reset bit is then set (the controller sets its normal operating mode), whereupon the relevant parameters can be altered, after which the reset bit is disabled. After that, the controller resumes operation with the altered parameters.

4. Columns 4 and 5 show
- the functions of the register;
- the meaning of the contents when the register is read;
- the meaning of the contents when the register is written to in the operating mode.

5. Columns 5 and 6 show the relevant data for the register in the reset mode. Here is an example of an internal SFR with address 4.

**Operating mode** (normal operation of the controller):

- Read – although reading the register is possible, there are no usable results since the read-out value is always FFH.
- Write: the register cannot be written to.

**Reset mode** (the controller is in the reset state).
- Read: reading the register gives the value of the acceptance code.
- Write: a new acceptance code can be written into the register.

This example shows that during normal operation of the controller this SFR has no special function. Note, however, that in the reset mode the acceptance code with which the controller functions during normal operation is set.

## CREATING THE ROUTINE FOR BASIC INITIALIZATION

Before work on this routine is begun, a close look at the Application Note for the SJA1000 controller (AN97076 – see Reference) is highly advisable. On page 23 of this document, the manufacturers give a flow diagram with detailed comments on how the initialization of the controller should be proceeded with (see **Figure 13**).

Another close look at the description of a single register should then enable the values of the parameters to be set readily to individual requirements and wishes.

## CREATING THE ROUTINE FOR SENDING DATA

As mentioned earlier, CAN controller type SJA1000 assumes most of the tasks involved in sending data. The sending of byte data onto the CAN bus requires only four actions.

- Delivery to the controller of the wanted identifier (ID) for the frame to be transmitted.
- Indicating how many data bytes are to be sent (0–8).
- Determining whether the frame is a remote transmission request (RTR) frame or not.
- Writing the wanted data bytes to the send data buffer of the controller.

That's all! The remainder of the process is carried out automatically by the CAN controller, that is:
- assembling the frame;
- calculation of the CRC sum;

- allocation of the other fields in the frame;
- accessing the bus;
- transmitting the frame;
- checking for errors;

and so on.

Messages indicating whether the transmission was successful or not are returned to the user via the Status register to enable action to be taken as appropriate.

## CREATING THE ROUTINE FOR RECEIVING DATA

In the reception of data, CAN controller types SJA1000 again takes on most of the necessary actions, that is, data are received almost wholly automatically by the controller. The controller processes the received frames and writes the wanted information contained in them to the error detection section and acceptance filter in its RXFIFO (=receiver first in first out memory) – see **Figure 14**.

If the acceptance filter is switched off, each received frame is evaluated. In the RXFIFO, the following data from each frame are stored (see Table 6, address range 20–29):

- frame identifier;
- remote transmission request (RTR) bit;
- data length code (DLC);
- useful data bytes.

As the range of the internal RXFIFO in IC3 is exactly 64 bytes, the number of frames that can be stored in the intermediate memory depends on the length of the frame, and more particularly on the data length code.

The receiver buffer window (see Table 6, addresses 20–29) that can be read by a user is what is shifted by the RXFIFO to the window. This consists of an actually received set of data (frame or message), which can be processed by the user via software.

Communication between the SJA1000 and the microcontroller or computer in the receive mode may take two forms.

- **Interrupt-driven.** When the controller has received a complete and error-free frame, it initiates an interrupt in the microcontroller via its pin 16 (INT). This causes the microcontroller/computer to react immediately to the received message so that this can be read without delay via the controller.
- **Polling* operation**. In this kind of operation, the receiver buffer status bit in the status register of the controller is continuously interrogated by the microcontroller/computer. When this is set – indicating that the controller has received at least one message correctly – the software reads this frame and processes it as relevant.

When a message has been read, the applications software reenables the receiver buffer window to acknowledge that the earlier message passed to it has been processed. The window is then ready to receive the next frame from the RXFIFO. In this way, the applications software ensures that one frame after another is processed.

There are two further matters to be noted.

- Immediately after a frame (message) has been read and processed, the receiver buffer window must be released by a 'release receiver buffer command' so that the controller can shift the next message to the window. If this command is not given, the same message is processed again and again, which causes the RXFIFO to overflow since other received frames are not being shifted.
- When a high frame rate is used, that is, when the data transfer rate is high or many messages are sent one after another, there is a risk of the RXFIFO overflowing rapidly if the messages are not shifted readily. If such situations are likely to occur, a sufficiently powerful microcontroller or computer must be used in association with high-quality software.

When an overflow of the RXFIFO occurs, the controller indicates this by setting an error-bit, that is, a Data Overrun Status bit in the status register. The relevant message, which is just about to be shifted into the RXFIFO (and which caused the overflow) is then erased and lost.

[990066-4]

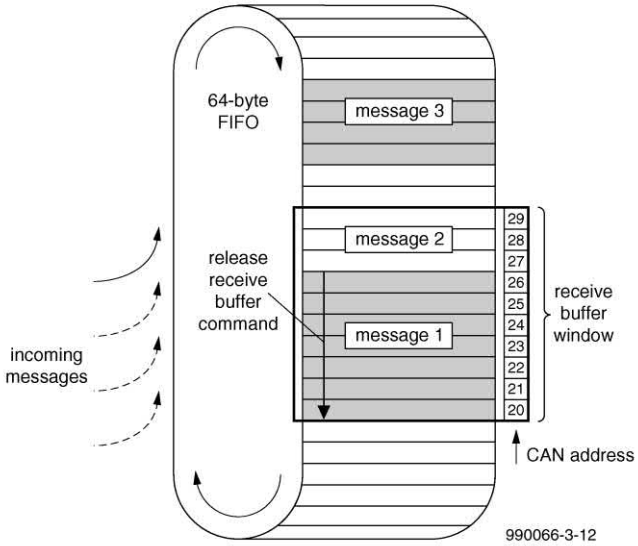



***Figure 14. Structure of the receiver memory range.***

*Only part of the software needs to be programmed by the constructor, since in next month's issue a complete applications program in Pascal will be published which enables all basic functions of the CAN bus to be tried and tested. The suggested microcontroller is a Type 80C537. Apart from the 80C537 used in the single-board compander published in the June 1997 issue of this magazine, there is now a smaller, less expensive version of this Single-Board Computer (SBC) available. This version will be highlighted in next month's issue under the title '537-Lite Computer'.*

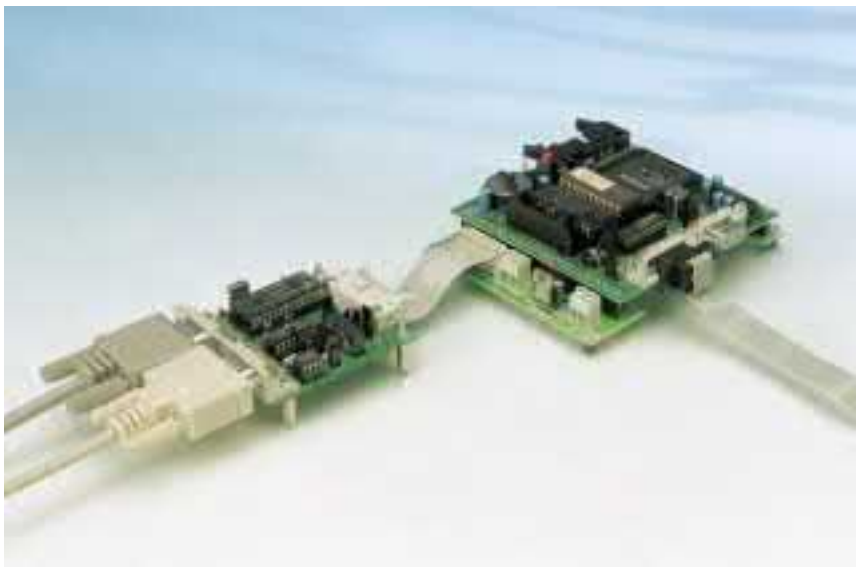
*Reference:*

*CAN bus controller and transceiver modules* (Philips Semiconductors)
www us.semiconductors.philips.com/can/
www-us.semiconductors.philips.com/ can /support

* Polling is a form of time division multiplexing. It is a process by which one node (primary station) in a network can address any other node (secondary station), giving the secondary node access to the communication channel.

# controller area network (CAN) (5)

## Part 5: a CAN interface using BASIC537



If you are new to the CAN bus, look forward to some hard work before seeing the first usable results. After all, at least two microcontroller systems have to be hooked up to CAN bus controllers, and a bus-based data link implemented based on the use of talk/listen programs. Once you have data travelling over the bus, all further expansion is really simple. This article attempts to make your first practical experiments with the CAN bus as easy as possible.

Hardware & software design
by B. Kainka

The CAN bus interface described in *Elektor Electronics* November 1999 may be controlled using the BASIC537 higher programming language. BASIC537 is an EPROM version of the well-known Intel MCS51 BASIC, specially adapted and extended for the 80C537 microcontroller. Many of you will be familiar with MCS51 BASIC because it was the subject of several articles in *Elektor Electronics*. Originally, this BASIC interpreter was developed for the (now obsolete) 8052AH-BASIC microcontroller. When stored in an external (E)PROM, however, it is also great for other controllers from Intel's 80xx series and second sources (see also the 80C32 BASIC Control Computer described in *Elektor Electronics* February and March 1998). The 537 'Lite' Computer is described elsewhere in this magazine — it too is capable of running BASIC537. The ancestor of this computer, a full-blown 80C357 microcontroller system, was published in our June 1997 magazine.

The new, smaller and cheaper 537 'Lite' Computer is employed here in

combination with the CAN bus interface. The 537 'Lite' is prominently featured on this month's cover.

## HARDWARE

An adaptor board was developed to implement a simple connection between the 537 'Lite' Computer (fitted with a BASIC537 EPROM) and the CAN interface board. The circuit diagram of the adaptor is given in **Figure 1**, the copper track layout and component mounting plan, in **Figure 2**. As you can see on the photograph, there are no wiring problems because the 537 'Lite' board is plugged directly on to the adaptor board. The link with the CAN interface is then made using a length of flatcable (see introductory photograph).

To solve the power supply problem in a simple way, a 5-volt voltage regulator and supply reversal protection are accommodated on the adaptor board. In this way, the adaptor board can supply +5 V to the two other boards. The upshot is that any low-cost wall adaptor supplying 9-12 volts unregulated at about 300 mA may be connected to PCB connector K2. If you already have a stabilized 5-volt line available, you may omit IC1, D1, C1 and C2 from the adaptor board, and connect the 5-V supply voltage directly to the K1 terminals on the adaptor board.

To keep cost down, the adaptor board is much smaller than the 537 'Lite' Computer board to be plugged on it. If you cut the adaptor board in two along the line indicated on the PCB overlay, and fit the two sub-boards at the right distance above a carrier (e.g. aluminium sheet), the 537 'Lite' Computer is easily plugged on to this assembly. The only wire connection to make (if necessary) is $\overline{INT2}$ between pin 12 (K3) and pin 32 (K6) (see the photograph showing the 537 'Lite' Computer board with the two adaptor sub-boards). The two adaptor sub-boards have connecting pins for the $\overline{INT2}$ link, which is also shown as a wire link on the component mounting overlay (Figure 2).

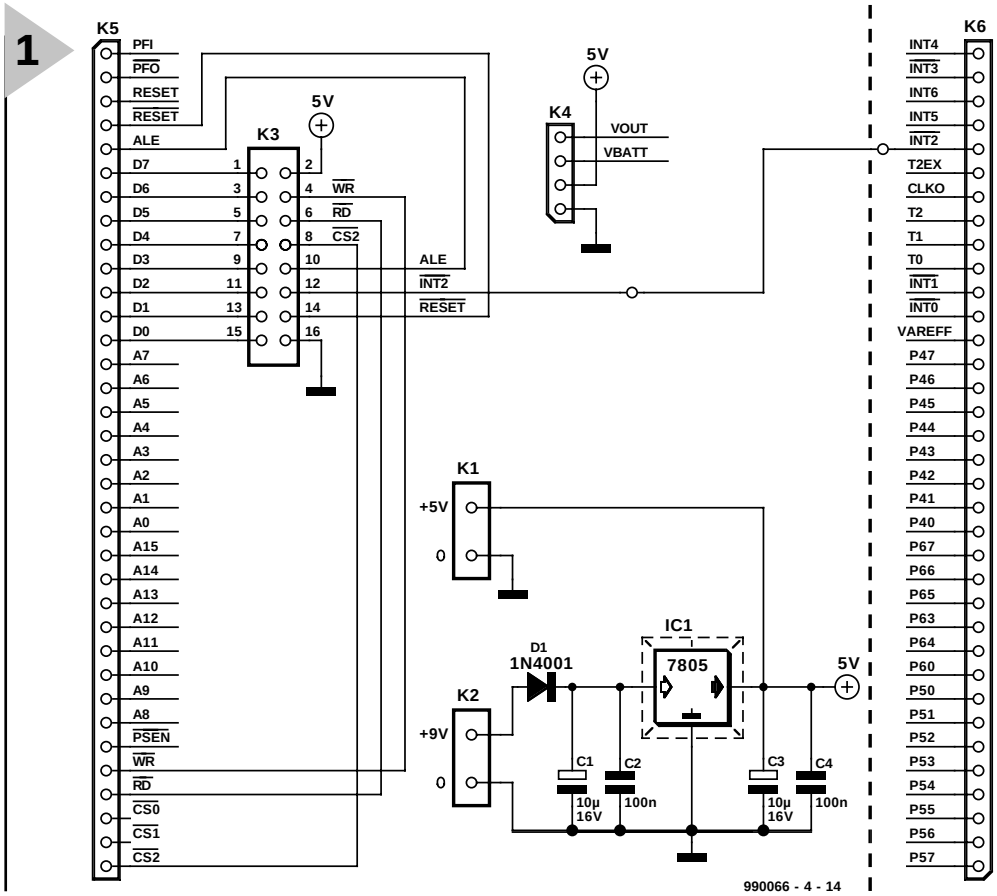



***Figure 1. Circuit diagram of an adaptor board that creates a simple link between the 537 'Lite' Computer board and the CAN bus interface.***

## CONTROL IT ALL IN BASIC

In essence, all you need to be able to control the CAN interface board is a program that looks after a bank of registers starting at address F000h in the CAN controller SJA1000. The XBY operator is employed for all access to addresses in the external RAM area and peripherals.

To make the introduction as easy going as possible, the following paragraphs describe the simplest case of a data link between two 80C537 systems. The essential settings are supplied as defaults by the program. The communication runs at 20 kbits/s. Message are sent without the RTR bit — in other words, no reply is requested. The two systems should fulfil the following functions:

**System 1** regularly sends messages with Identifier 300 and containing eight bytes. The data originates from the first eight channels of the A-D converter. The system continuously performs measurements on eight analogue inputs. The messages it transmits may be received and processed by any other system connected to the bus.

**System 2** receives all messages on the bus and copies them to the PC by way of the RS232 interface. In fact, this is a simple CAN monitor that allows you to pick up and examine all data traffic on the CAN bus.

A block diagram of the system configuration is given in **Figure 3**. A special cable is not required to link the two systems. Our first experiments in the lab indicated that a simple two-wire link between pins 4 and 8 of the CAN plugs is adequate if the total length does not exceed about 1 metre. With such a short cable, no difference was noted between the termination resistors being present or not.

## TRANSMIT PROGRAM AND TEST

The transmit program for Controller 1 is given in **Listing 1**. The CAN controller SJA1000 is addressed by the 80537 system via base address 0F000h. The address range is defined in line 95 (BA = 0F000h). If you use a different system, all you have to do (initially) is modify BA accordingly. The initialisation is carried out as described in the article on the CAN bus interface hardware. In lines 110 and 200, the results of the register programming are requested. The program then waits for a register bit to go to a specific state. In case the controller is not found on the bus, or does not function correctly, the program will 'hang' at this point. If everything is successful, however, you are greeted with these messages:

```
Reset OK
Init OK
```

To start with, it is sufficient to execute the initialisation routine up to line 200. An initial check may be made by looking for a rectangular signal at the test pin on the controller board. Whereas this pin supplies 8 MHz before the initialisation, you will then find 2 MHz. If this is okay so far, you may safely

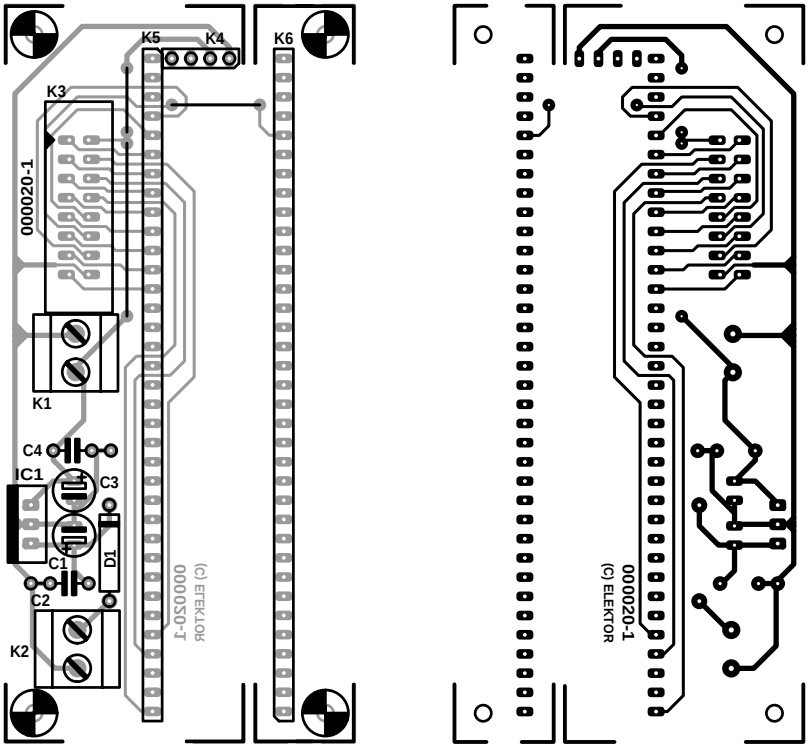


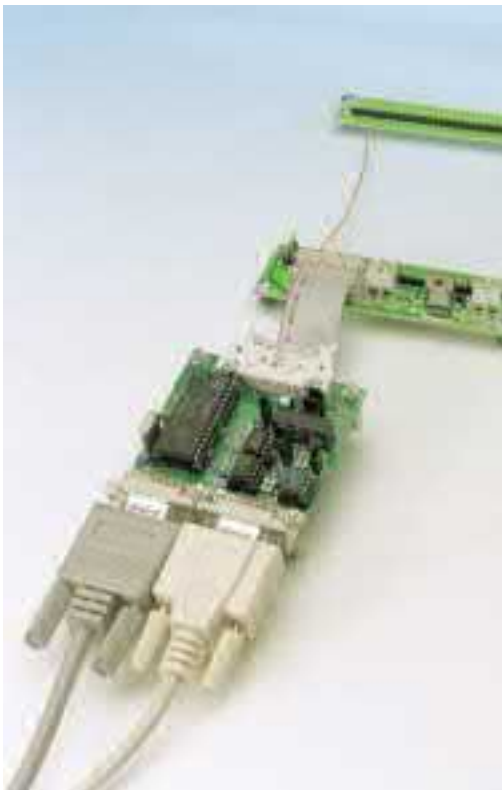

***Figure 2. Layout and component mounting plan of the adaptor board.***

**COMPONENTS LIST**

**Capacitors:**
C1, C3 = 10µF 16V (radial)
C2, C4 = 100nF (ceramic)

**Semiconductors:**
D1 = 1N4001
IC1 = 7805

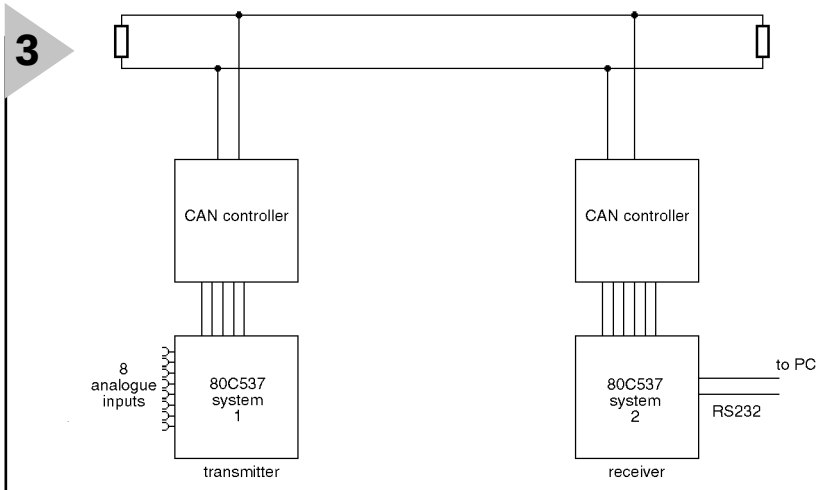
**Miscellaneous:**
K1 = 2-way PCB terminal block, raster 5mm
K2 = 2-way PCB terminal block, raster 5mm
K3 = boxheader, straight, 16 pins
K4 = pin header, 1 row, 4 pins
K5, K6 = pin header, one row, straight, 35 pins



assume that the controller is driven with the proper signals.

Now the complete program may be loaded and executed. Experienced as you are, you will no doubt have an oscilloscope ready to observe data traffic on the bus. Without a connection having been made to a second system, you will be able to find signals on the data lines. After a hardware reset and without an initialisation you will not be able to find the 'inactive' level of 2.5 V on the two wires. As soon as the transmit program is started, data is easily recognized as rectangular signals with a level of 1 V. The shortest logic levels are just 50 µs long, which means that the transmission rate is indeed 20 kBits/s. However, you will not fail to see that there is a quasi-steady datastream with 2-ms pauses, rather than short data packets as would be expected. Not to worry, however, this is the normal behaviour of the controller as long as it has not detected any intelligence on the CAN bus. Yet, it is not sufficient to connect the second controller via the two-wire link, because this, like system 1, has to be initialised. By the way, the transmitting station will continue to send a quasi-continuous datastream even if the BASIC program is terminated.

***Figure 3. Block diagram showing how the CAN bus is linked to the two 80537 systems programmed in 537 BASIC.***

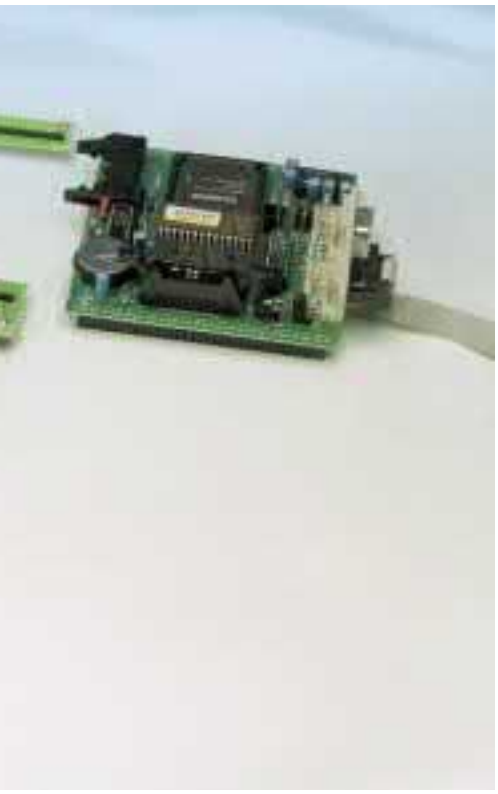



## AT THE RECEIVER SIDE

Now we are ready to 'deploy' the receiver program as given in **Listing 2**. This program is for system number 2. As you can see from the listing, the initialisation is the same as that used for the transmitter. As soon as the initialisation is done and the message "Init OK" has appeared on the PC display, the transmitting controller will com-

mence its normal operation. From then on, short data packets with a length of just over 5 ms will start to appear on the CAN bus. Finally, the CAN bus functions as you, the interested reader, would like to see it: data packets being sent back and forth over the bus without any indication of their being read anywhere at all!

The actual receiver program starts at line 500 and waits for a message which is announced by controller status bit zero. As soon as a data packet has arrived, the program may read ten bytes from the controller. The first two contain the message ID. It is recovered from two bytes in line 570 and then sent to the display. As expected, it is the 'ID', 300, which was arbitrarily defined in the transmitter program.



The user data are read in a loop and displayed in line 610. There , you (finally…) get the measured values on the eight analogue inputs of the first controller system. **Figure 4** shows the received data in a terminal window.

## FINALLY: THREE ON THE BUS

Of course, the results up to now could have been obtained with a rather simpler RS232 interface. The CAN bus however will typically not unleash its power until more than two devices are connected to the bus. To end the 'lonely' existence of the two systems discussed so far, a third 'CANable' device should be added. The program CAN3.BAS shown in **Listing 3** (without initialisation!) performs the following functions:

***Listing 1. Transmitter program CAN1.BAS.***

```
90    REM Init CAN Controller
95    BA=0F000H
100   XBY(BA+00H)=01H :  REM Reset Mode
110   IF (XBY(BA+00H).AND.1)<>1 THEN  GOTO 110
111   PRINT ″Reset OK″
120   XBY(BA+1FH)=43H :  REM CDR, 2 MHz
130   XBY(BA+04H)=0 :  REM ACR
140   XBY(BA+05H)=0FFH :  REM AMR, Acceptance Mask, all
150   XBY(BA+06H)=53H :  REM BTR0, 20 Kbit/s*
160   XBY(BA+07H)=2FH :  REM BTR1
170   XBY(BA+08H)=1AH :  REM OCR;
180   XBY(BA+01H)=0EH :  REM CMR, end sleep mode
190   XBY(BA+00H)=0 :  REM CR, end reset mode
200   IF (XBY(BA+00H).AND.1)>0 THEN  GOTO 200
201   PRINT ″init ok″
500   REM ************* Main Loop ***************
501   REM Send 8 Bytes of AD-Data in message 300
510   FOR N=0 TO 7
520   XBY(BA+0CH+N)=AD(N) :  REM fill TB1..TB8
530   NEXT N
540   ID=300 :  REM Message Identifier
550   DFL=8 :  REM 8 Bytes
560   GOSUB 1000 :  REM Send Massage
570   FOR T=1 TO 1000 :  NEXT T
580   GOTO 500
1000  REM ************* Send CAN Telegram *************
1010  IF (XBY(BA+02H).AND.4)=0 THEN  GOTO 1010 :  REM SR
1020  XBY(BA+0AH)=INT(ID/8) :  REM IDT1
1030  XBY(BA+0BH)=(ID-8*INT(ID/8))*32+DFL :  REM IDT2
1040  XBY(BA+01H)=0DH :  REM CMR, start transmission
1050   RETURN
```

***Listing 2. Receiver program CAN2.BAS***

```
90    REM Init CAN Controller
95    BA=0F000H
100   XBY(BA+00 H)=01H :  REM Reset Mode
110   IF (XBY(BA+00H).AND.1)<>1 THEN  GOTO 110
111   PRINT ″Reset OK″
120   XBY(BA+1FH)=43H :  REM CDR, 2 MHz
130   XBY(BA+04H)=0 :  REM ACR
140   XBY(BA+05H)=0FFH :  REM AMR, Acceptance Mask, all
150   XBY(BA+06H)=53H :  REM BTR0, 20 Kbit/s*
160   XBY(BA+07H)=2FH :  REM BTR1
170   XBY(BA+08H)=1AH :  REM OCR;
180   XBY(BA+01H)=0EH :  REM CMR, end sleep mode
190   XBY(BA+00H)=0 :  REM CR, end reset mode
200   IF (XBY(BA+00H).AND.1)>0 THEN  GOTO 200
201   PRINT ″Init OK″
500   REM ******* Receiver Main Loop *************
510   SR=XBY(BA+02H) :  REM Status Register
520   REM Error Detection and Clear Data Overrun
530   if (SR .AND. 2) = 2 then XBY(BA+01H)=8: :Goto 510
540   REM Get Receive Status
550   if (SR .AND. 1) =0 then goto 510
560   REM Read received message
570   ID=XBY(BA+14H)*8+INT(XBY(BA+15H)/32) :  PRINT ID
580   DFL=XBY(BA+15H).AND.15 : rem   Data Length
590   RTR=(XBY(0FE15H).AND.16)/16 :  REM RTR not used
600   FOR N=0 To 7
610   PRINT N ,XBY(BA+16H+N)
620   NEXT N
630   XBY(BA+01H)=0CH :  REM Release Receive Buffer
640   GOTO 510
```

***Listing 3. Receiver & Transmitter program CAN3.BAS without initialisation.***

```
500 REM ************ Main Loop ***************
505 REM ************ Receiver ****************
510 SR=XBY(BA+02H) :  REM Status Register
520 REM Error Detection and Clear Data Overrun
530 IF (SR.AND.2)=2 THEN XBY(BA+01H)=8 :  GOTO 510
550 IF (SR.AND.1)=0 THEN  GOTO 510
560 REM Read received message
570 ID=XBY(BA+14H)*8+INT(XBY(BA+15H)/32): Print ID
580 DFL=XBY(BA+15H).AND.15 :  REM   Data Length
590 RTR=(XBY(0FE15H).AND.16)/16 :  REM RTR not used
600 IF ID<>300 THEN  GOTO 660
610 PORT=0
620 IF XBY(BA+16H+0)>100 THEN PORT=PORT+1
630 IF XBY(BA+16H+1)>100 THEN PORT=PORT+2
640 IF XBY(BA+16H+2)>100 THEN PORT=PORT+4
650 WRSFR 0E8H,PORT :  REM Port 4 Output
660 XBY(BA+01H)=0CH :  REM Release Receive Buffer
800 REM ******** Send AD-Data ***********
810 FOR N=0 TO 7
820 XBY(BA+0CH+N)=AD(N) :  REM fill TB1..TB8
830 NEXT N
840 ID=500 :  REM Message Identifier
850 DFL=8 :  REM 8 Bytes
860 GOSUB 1000 :  REM Send Message
870 FOR T=1 TO 1000 :  NEXT T
880 GOTO 500
1000 REM ******* Send CAN Telegram *************
1010 IF (XBY(BA+02H).AND.4)=0 THEN  GOTO 1010 :  REM SR
1020 XBY(BA+0AH)=INT(ID/8) :  REM IDT1
1030 XBY(BA+0BH)=(ID-8*INT(ID/8))*32+DFL :  REM IDT2
1040 XBY(BA+01H)=0DH :  REM CMR, Start Transmission
1050 RETURN
```

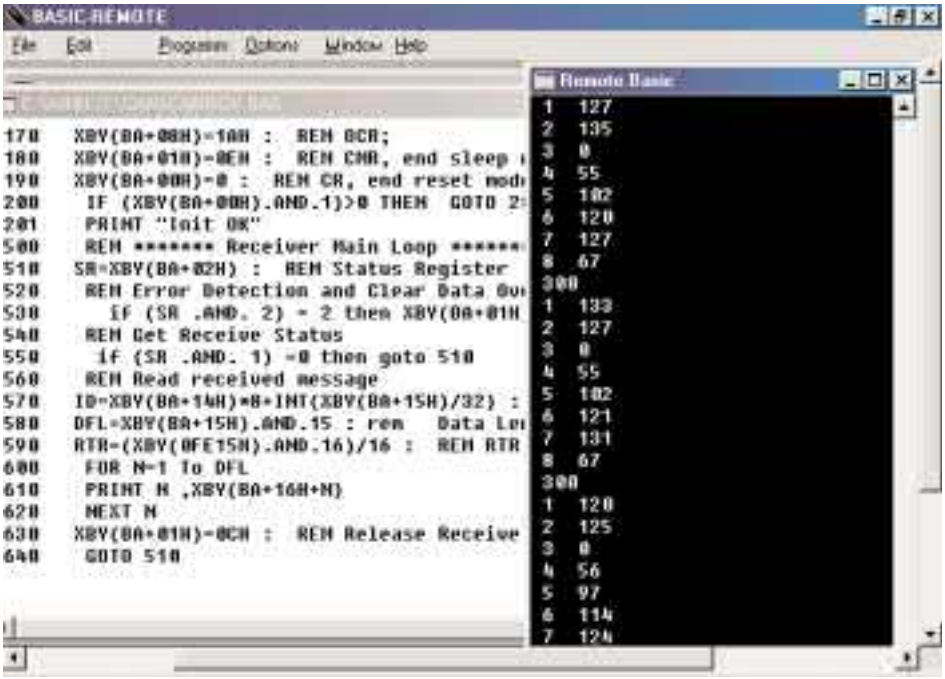



***Figure 4. Received data in the terminal window of BASIC537.***

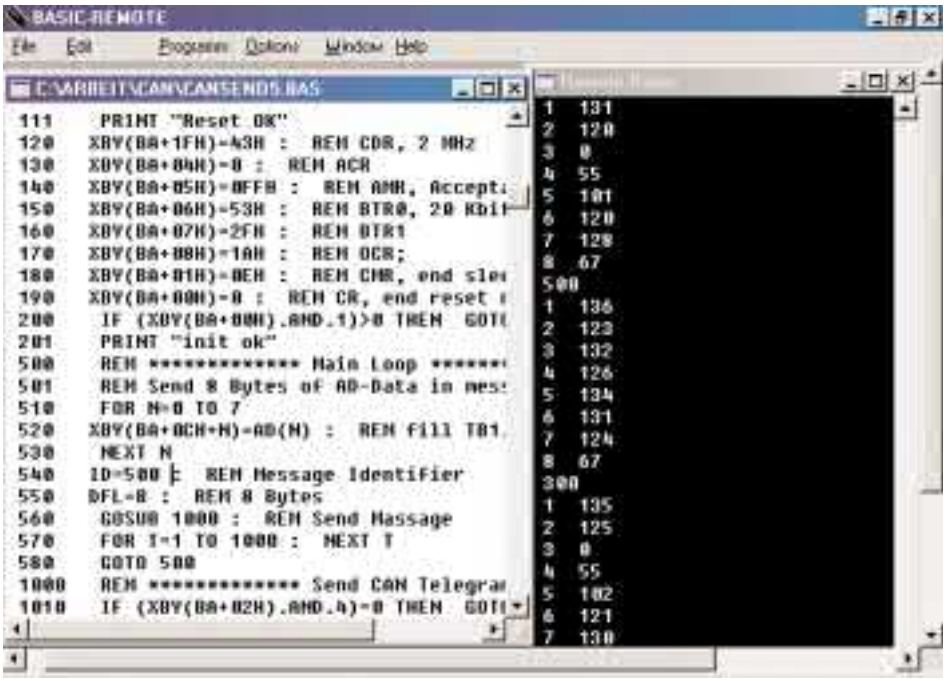



***Figure 5. Reception of messages with Identifiers (IDs) '300' and '500'.***

It receives all messages but only processes the ones with the ID '300'. The first three transmitted measurement values are compared with certain extreme values and switch on three lines on Port 4 if a particular extreme is exceeded.

After processing of the received message, a message with the ID '500' is returned, where all A-D channels are measured and transmitted again. As soon as the third system is connected to the bus, System 2 will also supply data with ID '500' to the terminal (see **Figure 5**).

990066-4

*Note:* the three program listings discussed in this article are available for downloading from the Elektor Electronics website at *www.elektor-electronics.co.uk*

*Article editing (German original): E. Krempelsauer*
*Design editing: K. Walraven*