An overview of Controller Area Network

by M. Farsi, K. Ratcliff and Manuel Barbosa

The Controller Area Network is a well-established networking system specifically designed with real-time requirements in mind. Developed in the 1980s by Robert Bosch, its ease of use and low cost has led to its wide adoption throughout the automotive and automation industries. However, for the beginner using CAN may seem somewhat bewildering. This article goes some way into explaining how CAN is used both at the hardware and the software levels.

The Controller Area Network (CAN) was originally developed in the 1980s for the interconnection of control components in automotive vehicles. The complexity of the control functions implemented by engine management systems, anti-lock brakes and skid controls normally requires dedicated lines for the interconnection of the different control components. However, a continuous increase in complexity has led to a physical maximum not only in the quantity of wires required but also in physical connector size. CAN enabled a huge reduction in wiring complexity and, additionally, made it possible to interconnect several devices using a single pair of wires, allowing data exchange between them at the same time.

Needless to say, it was not long before this idea migrated from vehicles into the machine and automation markets. Nowadays CAN has found its way into such diverse areas as agricultural machinery, medical instrumentation, elevator controls, fairground rides, public transportation systems and industrial automation control components. It is because of its widespread use that CAN semiconductors are inexpensive. Furthermore, since a large number of semiconductor manufacturers, such as Philips, Motorola, National Semiconductors, Siemens and Intel (to name but a few) produce CAN devices, CAN technology is guaranteed well into the future.

The basic features of CAN are:

- *High-speed serial interface:* CAN is configurable to operate from a few kilobits per second right up to 1 Mbit/s transmission rates.
- *Low-cost physical medium:* CAN operates over a simple twisted wire pair, therefore cabling a CAN network is inexpensive compared to multicore or coaxial cables often required by other bus systems.
- *Short data lengths:* The short data lengths of CAN messages mean that CAN has very low latency when compared to other systems.
- *Fast reaction times:* The ability to transmit information without requiring a token or permission from a bus arbiter results in extremely fast reaction times.
- *Multi-master and peer-to-peer communication:* Using CAN it is simple to broadcast information to all or a subset of nodes on the bus and just as easy to implement peer-to-peer communication.
- *Error detection and correction:* The high level of error detection and number of error detection mechanisms provided by the CAN hardware means that CAN is extremely reliable as a networking solution.

CAN operating principles

CAN allows the implementation of peer-to-peer and broadcast or multicast communication functions with lean bus bandwidth use. The basic principles of CAN communication are explained in the following subsections.

Communication modes and data exchange

When data is transmitted over a CAN network no individual nodes are addressed. Instead, the message is assigned an identifier that works as a unique tag on its data content. The identifier not only defines the message contents but also the message priority.

When a node wishes to transmit information it simply

Authorized licensed use limited to: University of Maryland College Park. Downloaded on August 16,2010 at 19:37:42 UTC from IEEE Xplore. Restrictions apply.

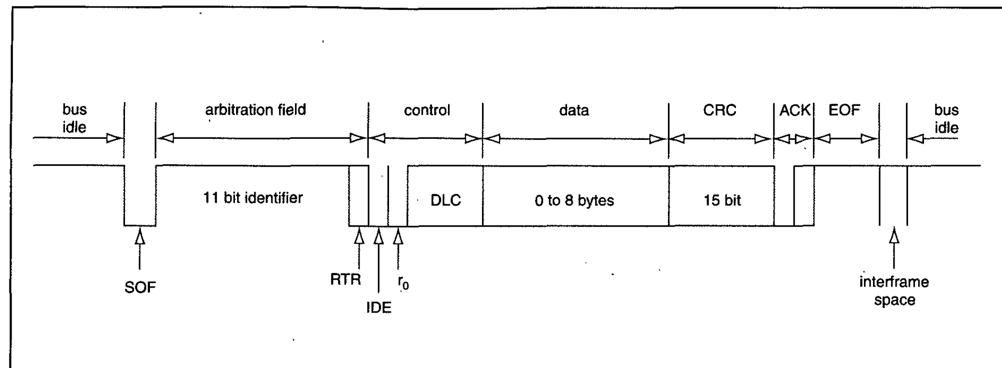

Fig. 1 Format of a CAN telegram

passes the data and the identifier to its CAN controller and sets the relevant transmit request. It is then up to the CAN controller to format the message contents and transmit the data in the form of a CAN frame. Once the node has gained access to the bus and is transmitting its message, all other nodes become receivers. Having received the message correctly, these nodes then perform an acceptance test to determine if the data is relevant to that particular device, based on the identifier of the message.

Therefore, it is not only possible to perform communication on a peer-to-peer basis where a single node accepts the message but also to perform broadcast and synchronised communication whereby multiple nodes can accept the same message using a single transmission. Furthermore, the ability to send data on an event basis means that bus load utilisation can be kept to a minimal amount.

This concept has become known in the networking world as the producer/consumer mechanism whereby one node produces data on the bus for other nodes to consume. One difference with CAN over other fieldbus solutions is that this mechanism requires no interaction from a bus master or arbiter.

The ability to send data on an event basis means that bus load utilisation can be kept to a minimal amount

Telegram format

Fig. 1 shows the format of a CAN telegram (standard format). It shows the CAN message format that uses 11-bit identifiers (2.0A format); however, an extended CAN format (2.0B format) also exists that uses 29-bit identifiers instead. CAN controllers supporting the extended format will in general also work with the standard format communication using 11-bit identifiers although the reverse is not always true. Some devices supporting purely the standard format will be able to tolerate other devices transmitting CAN frames using the extended format (2.0B passive devices) and function correctly.

A message in the standard format begins with the start bit or start of frame (SOF). This is followed by the arbitration field which contains the identifier of the CAN telegram and is used to arbitrate access to the bus. Also part of the arbitration field is the RTR bit (remote transmission request) which indicates whether the frame is a request frame (without any data, this type of message is used to trigger a transmission by another node) or a data frame.

The control field contains the IDE bit (identifier extension), which indicates whether the frame is a standard format frame or an extended one, the r_0 bit that is reserved for future extensions and four additional bits containing the length of the data field (data length code).

Next comes the data field which can be from zero to eight bytes in length and the CRC field that contains a 15-bit code that is used to check frame integrity.

The acknowledge (ACK) field comprises an ACK slot bit and an ACK delimiter bit. The ACK slot is transmitted as a recessive bit (a bit with a value of 1) and receivers that retrieve the message correctly (regardless of whether the message is meant for the controller or not) overwrite this field with a dominant bit (a bit with a value of 0). The detection of this dominant bit by the transmitter means that the message was accepted by at least one node and was therefore error-free (a further explanation of what are dominant and recessive bits can be found in the next section).

Authorized licensed use limited to: University of Maryland College Park. Downloaded on August 16,2010 at 19:37:42 UTC from IEEE Xplore. Restrictions apply.

The end of frame field (EOF) denotes that the frame terminated. Finally, the intermission (Int) space represents the minimum number of bit periods that need to elapse following the frame before another station is allowed to transmit a message. If no other transmissions follow the frame the bus remains in its bus idle state.

Arbitration

CAN employs the carrier sense multiple access with collision detection (CSMA/CD) mechanism in order to arbitrate access to the bus. It uses a priority scheme based on numerical identifiers in order to resolve collisions between two nodes wishing to transmit at the same time.

On the CAN bus a 'zero' is called a dominant bit because it overwrites a 'one' (a recessive bit). Therefore, a node transmitting a 'one' whilst another transmits a 'zero' will result in a 'zero' level on the bus (the one is overwritten). This process is shown in Fig. 2.

When two or more nodes wish to transmit, they sense the bus and if there is no bus activity, they begin to transmit their message identifier (most significant bit first). At the same time that they transmit their identifiers, they also monitor the bus levels. If one node transmits a recessive bit on the bus and the other transmits a dominant bit the resulting bus level is a dominant bit. Therefore, the node transmitting a recessive bit will see a dominant bit on the bus (situation where A and B lose in Fig. 2) and stop transmitting any further information. This allows the node with the lowest number in its identifier field to gain access to the bus and transmit its message. Any node that has lost during the arbitration process then waits until the bus becomes free before trying to retransmit its message.

Note that this scheme means that no bandwidth is wasted during the arbitration process. Ethernet (for example) also uses CSMA/CD, but if there is a collision between two nodes, one node will transmit a jamming signal causing both nodes to abort the transmission. Both nodes will then wait a random period before trying to retransmit.

The bus arbitration process used by CAN means that the node with the highest priority (lowest value in the identifier field) will continue to transmit without any interruption. This gives CAN very predictable behaviour (no random waiting) and very efficient use of the bus. In fact, it is possible to have CAN networks operating at near 100% bus bandwidth.

Acknowledgment and error checking/signalling mechanisms

Unlike other bus systems CAN does not use acknowledgment messages that by comparison represent a waste of bandwidth on the bus. As mentioned previously, each receiver that receives the message correctly acknowledges the message by transmitting a dominant bit in the ACK slot. This will notify the transmitter that the message was received correctly by at least one node. All nodes check all frames for errors and any node in the system that detects an error actively signals this to the transmitter. This means that CAN has network wide data security as a transmitted frame is checked for errors by *all* nodes regardless of any filtering of the CAN telegrams.

The error-checking mechanisms implemented in CAN are:

- *Bit errors:* When a transmitter places a bit on the bus it simultaneously monitors the bus to determine whether the actual bit level on the bus matches the intended one.

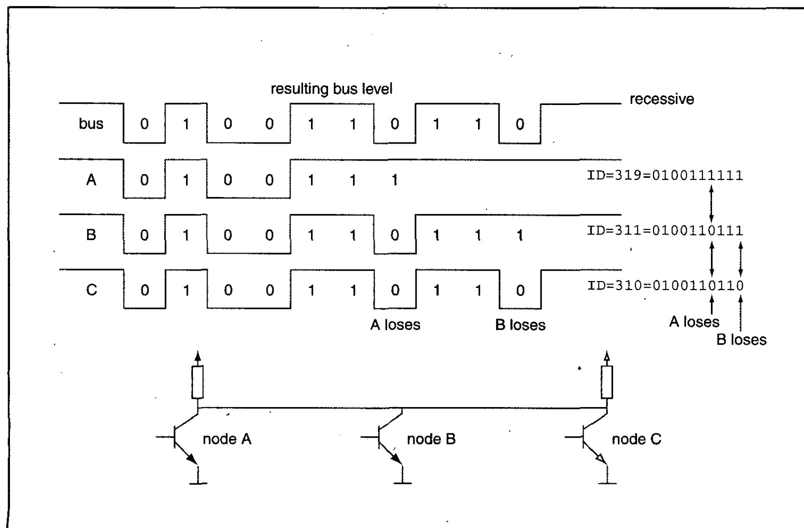

Fig. 2 Arbitration mechanism in CAN showing the bit levels as transmitted by three nodes A, B and C and the resulting bus bit levels

Authorized licensed use limited to: University of Maryland College Park. Downloaded on August 16,2010 at 19:37:42 UTC from IEEE Xplore. Restrictions apply.

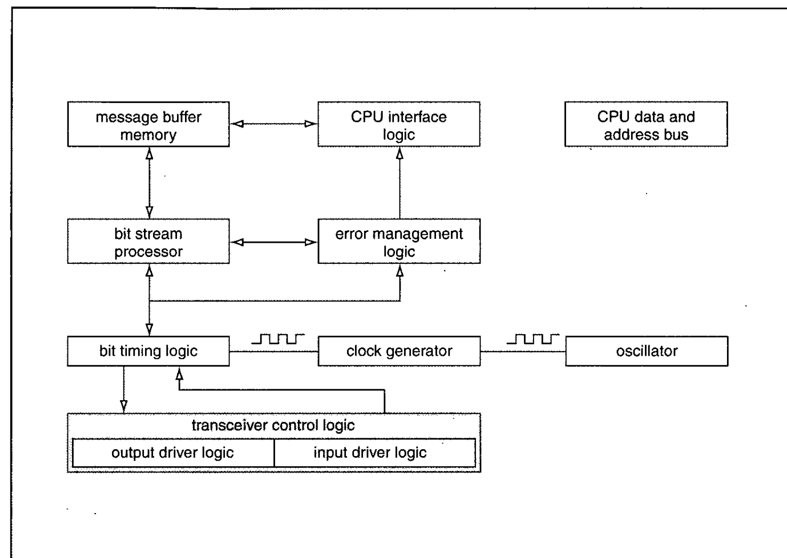

Fig. 3 CAN controller organisation

- *Bit stuffing errors:* Bit stuffing consists in inserting a bit of opposite polarity when five consecutive bits of the same polarity are transmitted on the bus. The stuffing bits are removed at the receiver end before the message is processed. CAN uses bit stuffing for two purposes. The first is to provide frequent level transitions on the bus to allow receivers to re-synchronise and adjust internal timing accordingly. The second is as an error checking mechanism whereby a violation of the bit stuffing rule is deemed an error. For example, the reception of six consecutive recessive bits is a bit stuffing error.
- *Cyclic redundancy check (CRC):* Each CAN telegram carries a 15-bit CRC code. This 15-bit CRC code is calculated by both the transmitter and the receiver. The transmitter transmits the CRC as part of the frame and this is compared with the receiver's own independent CRC calculation. If the two calculations do not agree, an error has occurred during transmission of the frame.
- *Form errors:* Incoming CAN frames are checked by the receiver to make sure that the size in bits of individual parts of the frame are as expected, i.e. there are no illegal bits in a predefined field of the frame.
- *Acknowledgment errors:* As mentioned earlier, frames are acknowledged by receivers by inserting a dominant bit into the ACK slot of the frame. If no acknowledgment is detected by the transmitter, there may be an error detected by the recipients. It could also mean the ACK slot has been corrupted or that no receivers exist on the network.

If an error is detected by *any* of the other nodes (regardless of whether the message was meant for it or not) the transmission is aborted by transmission of an active error frame from at least one node. An active error frame consists of six consecutive dominant bits and it prevents the other nodes from accepting the erroneous message. The active error frame violates bit stuffing and may corrupt the fixed form of the frame causing other nodes to transmit their own active error frames. After an active error frame, the transmitting node begins re-transmission of the erroneous frame automatically.

CAN controllers implement two transmit and receive error counters through which they keep track of the number of errors detected during transmission and reception of frames, respectively. These counters are implemented in hardware and their operation is regulated by a rather intricate set of rules. In a very simplistic view of this mechanism, we can say that the counters are incremented by 'eight' every time a frame is found erroneous and decremented by 'one' every time a message is transmitted or received correctly. Over a period of time, the error count may increase even if there are fewer corrupted frames than uncorrupted ones.

During normal operation, the CAN controller is said to be in its *error-active state*. In this state, the node is able to transmit an active error frame every time a CAN frame is found to be corrupt. If one of the error counters reaches a warning limit of 96 error counts (indicating significant accumulation of errors) this is signalled by the controller usually using an interrupt. The controller operates in its error active mode until a limit of 127 error counts has been exceeded.

Once 128 error counts have been reached, the CAN controller enters an *error-passive state*. In this state, an error-passive controller is still able to transmit and receive messages but signals errors by transmitting a

Authorized licensed use limited to: University of Maryland College Park. Downloaded on August 16,2010 at 19:37:42 UTC from IEEE Xplore. Restrictions apply.

passive error frame. A passive error frame consists of six recessive bits and this frame will only abort transmissions performed by the node itself or in situations where the node is the only receiver. Otherwise it will be ignored and overwritten by other CAN controllers. If the error count drops below 128 again the controller then becomes error-active again transmitting active error frames as required.

If the error count reaches or exceeds a limit of 256, the controller enters its *bus-OFF* state. In this state the controller can no longer transmit or receive messages until it has been reset by the host processor. A node can also recover from its bus-OFF state when a series of 128 frames of 11 recessive bits have been detected on the bus. In this case the error counters are reset to zero by the controller, which then becomes error-active again.

CAN controller organisation

Fig. 3 shows how a typical CAN controller is organised at the silicon level:

- The *CPU interface logic* (CIL) executes commands from the host processor and controls data transfers on the serial bus. Global status and control registers bits as well as the control bits of the communication objects are used primarily by the CPU interface logic.
- The *bit stream processor* (BSP) controls the data stream between the message buffer memory (parallel data) and the bus line (serial data). It controls the entire protocol, differentiates between the frame types and detects frame errors.
- The *error management logic* receives error messages from the bit stream processor and, in turn, sends back information about the error state to the bit stream processor and the CPU interface logic.
- The *bit timing logic* (BTL) determines the timing of the bits and synchronises with the edges of the bit stream on the CAN bus. It monitors the bus line through a differential input comparator. The BTL synchronises on a transition at the start of the frame and re-synchronises on further transitions during reception of the frame. The BTL also provides programmable time segments to compensate for propagation delays and phase shifts.
- The *transceiver control logic* (TCL) consists of bit stuffing logic, programmable output driver logic, CRC logic and data shift registers. The BSP co-ordinates the individual elements of the TCL. Message reception, arbitration, message transmission and error signalling are actually performed by the TCL.
- The *message buffer memory* stores individual CAN objects for transmission or reception. The CPU communicates only with this area in order to transmit and receive messages. The bus interface logic manages the bus traffic.
- The *clock generator* is simply used to derive a suitable clock frequency for the CAN controller based on the frequency of an external clock oscillator.

Software register organisation

Due to the popularity of CAN, there is a large variety of CAN controllers and integrated microcontrollers available on the market. However, they all present common functionality and many of the registers are programmed in a similar manner, from one controller to another.

Message filtering

Generally two kinds of CAN controllers exist and these were formerly known as full CAN and basic CAN. The distinction between the two is less important nowadays, considering most of the newer full CAN controllers also provide some of the functionality of basic CAN.

In full CAN controllers, individual sections or objects

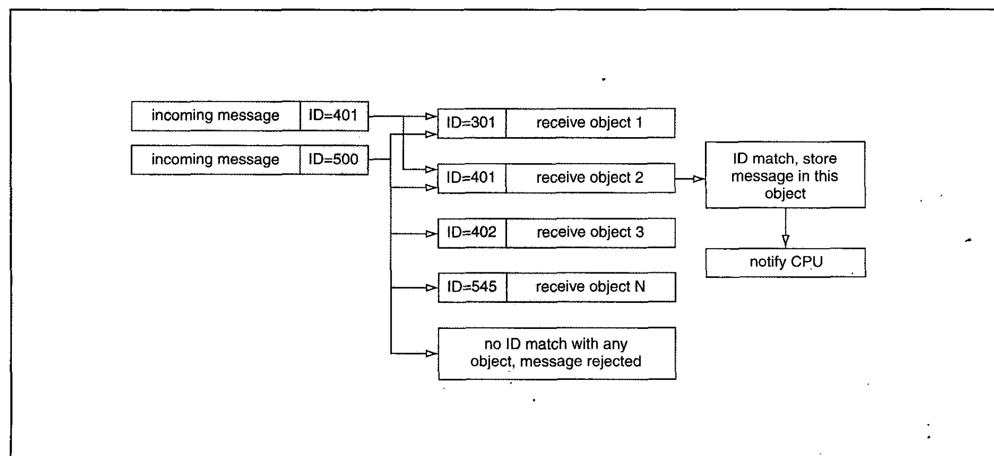

Fig. 4 Principles of full CAN operation

Authorized licensed use limited to: University of Maryland College Park. Downloaded on August 16,2010 at 19:37:42 UTC from IEEE Xplore. Restrictions apply.

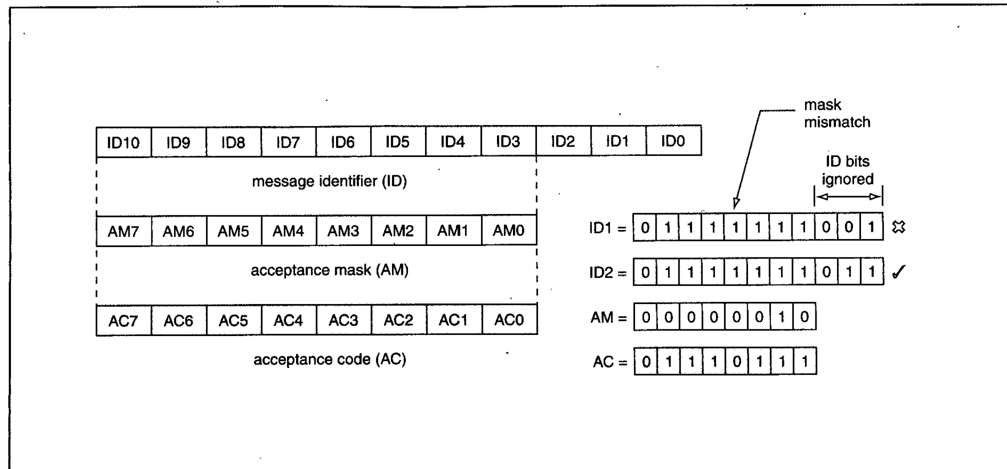

Fig. 5 Basic CAN acceptance filtering registers

of the message buffer memory (see Fig. 4) are reserved for the reception or transmission of CAN frames with preset programmable identifiers. When receiving a message, if the identifier matches the one programmed into the header of the object, the data is stored in that object or memory area. If the identifier does not match any of the programmed object identifiers, the message is rejected by the hardware.

In basic CAN implementations, the controller receives all messages regardless of their identifier and puts them into a receive message buffer. It is then up to software to accept or reject the incoming messages. Therefore, a software interrupt routine is invoked every time a CAN message is received, regardless of whether the CAN message is intended for the application or not. This can add a large amount of code and processing overhead to an application. Overheads may be reduced in some instances as a basic CAN controller will normally provide a rudimentary acceptance filtering scheme that allows the controller to reject a subset of the CAN identifier range. In a Philips 8×592 microcontroller, and the Motorola 68HC05X family of microcontrollers, for example, the filter consists of an acceptance mask (AM) register and an acceptance code (AC) register as shown in Fig. 5.

Both the acceptance code and acceptance mask registers are normally eight bits in length and the filtering is usually based on the eight most significant bits of the CAN identifier. The acceptance mask register defines whether the corresponding bits in the acceptance code register and in the CAN identifier must match to pass the acceptance test (the acceptance mask bits set to zero if a match is required). For this reason, in Fig. 5, ID1

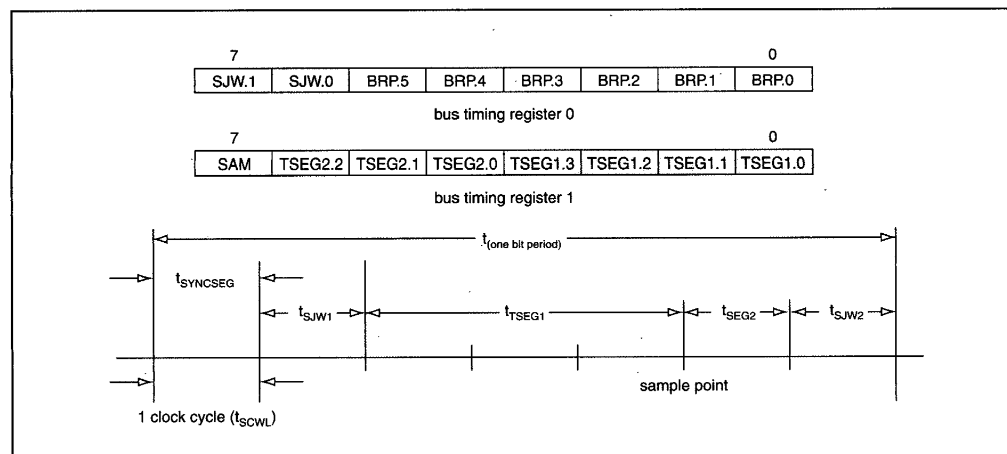

Fig. 6 Bus timing registers and bit periods

Authorized licensed use limited to: University of Maryland College Park. Downloaded on August 16,2010 at 19:37:42 UTC from IEEE Xplore. Restrictions apply.

fails the acceptance test whilst ID2 passes. It should be noted that the last three bits of the identifier are not taken into consideration. Therefore if the acceptance code in the example is set to 01110111 binary (952 decimal) and all bits in the acceptance mask are set to 'zero' (meaning that the top eight bits of the identifier must equal the eight bits of the acceptance code register) then a range of identifiers from 952 to 960 will pass the filtering test by the CAN controller. The software is then required to complement the filtering if only some of these messages are to be accepted.

Bus timing

In most CAN controllers, two eight bit registers are used to program the bit rate for CAN communication. Additionally, it is also possible to control the bit sample point and the maximum amount of adjustment of bit width that can be applied in order to resynchronise with the bit stream on the bus. Fig. 6 shows the two bus timing registers and their relationship with these parameters.

The timing of one bit period, shown as $t_{(\text{one bit period})}$, comprises several bit timing logic (BTL) cycles (t_{SCL}). Furthermore, the bit period is divided into five segments: $t_{SYNCSEG}$, t_{SJW1}, t_{SJW2}, t_{TSEG1} and t_{TSEG2}.

During the period $t_{SYNCSEG}$ the incoming edge of a bit is expected. This segment corresponds to one BTL cycle. The synchronisation jump widths (SJW1 and SJW2) are adjusted to compensate for phase shifts between the clock oscillators of the bus nodes. The width of SJW1 is increased to a maximum of twice the programmed width during resynchronisation. The width of SJW2 is reduced or cancelled to shorten the bit time during resynchronisation. Thus, the overall position and width of the bit time are adjusted according to incoming edge transitions. Both SJW1 and SJW2 are values set between one and four BTL cycles and programmed using SJW1 and SJW0 in bus timing register 0.

The position of the sample point is defined by t_{TSEG1} and t_{TSEG2}. These are periods programmed by TSEG2 and TSEG1 in bus timing register 1. For most applications, the sample point will be set at around 75 to 88% (i.e. TSEG2 normally set to a value of 1) of the total bit width. This will allow for any distortion effects to bit levels caused by signal propagation and the transmission media, particularly at higher baud rates. The bit SAM controls the number of samples that the CAN controller makes when determining the bit level on the bus. If SAM= 0 then one sample is taken. If SAM=1 then three samples are taken and a simple majority rule scheme determines the bit level. The use of three samples is restricted to low-speed applications due to the operating speed of the CAN controller but may help reduce errors in bit sampling.

Table 1 Worst case inter-frame spacing

baudrate	inter-frame space
1000 kbit/s	47 μs
500 kbit/s	94 μs
250 kbit/s	188 μs
125 kbit/s	376 μs

The baud rate prescaler (BRP) divides the oscillator clock to give the BTL cycle time. The BTL cycle time is set to twice the oscillator cycle time multiplied by BRP+1. Therefore, given how one bit period is made up of several smaller segments it is possible to calculate the overall bit time $t_{(\text{one bit period})}$ in BTL cycles as:

- SJW1 = SJW2 = 2SJW.1+SJW.0+1 (BTL cycles)
- TSEG1 = 8TSEG1.3+4TSEG1.2+2TSEG1.1+ TSEG1.0+1 (BTL cycles)
- TSEG2 = 4TSEG2.2+2TSEG2.1+TSEG2.0+1 (BTL cycles)
- $t_{(\text{one bit period})}$ = SYNCSEG + SJW1 + TSEG1 + TSEG2 + SJW2 (BTL cycles)

Note that certain restrictions apply to the values placed in these parameters. This is due to restrictions in the processing times of the various functional parts of the CAN controller.

Software implementation

Consideration of how the application software handles the hardware interface between the CAN controller and processor can play a major part in the communication process. Whether we are using basic or full CAN devices it is vitally important that messages

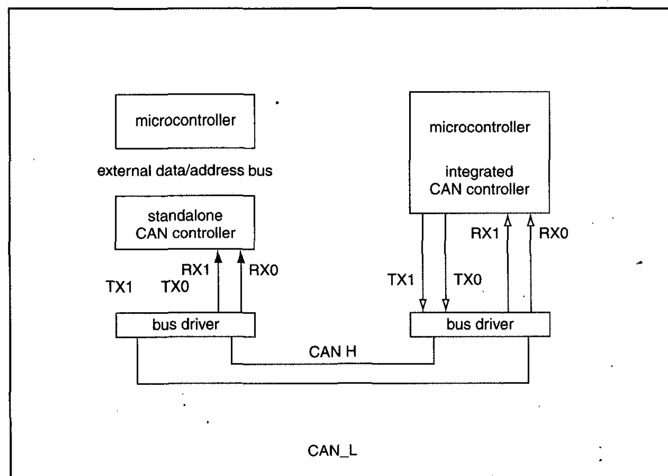

Fig. 7 Variants of CAN hardware

Authorized licensed use limited to: University of Maryland College Park. Downloaded on August 16,2010 at 19:37:42 UTC from IEEE Xplore. Restrictions apply.

Table 2 Selection of transceiver chips

part number	max speed	manufacturer
SN75LBC031D	500 kbit/s	Texas Instruments
UC5350	1 Mbit/s	Unitrode
82C250, 82C251	1 Mbit/s	Philips Semiconductors

Table 3 Standalone and integrated CAN semiconductors

part number	extended ID	microcontroller	manufacturer
FM2C	yes	16-bit	Fujitsu
H8/300H	yes	16-bit	Hitachi
AN 82527	yes	none	Intel
AN87C196C4	yes	16-bit	Intel
68HC05X4	no	8-bit	Motorola
M37630 E4/M4	yes	8-bit	Mitsubishi
COP684BC	passive	8-bit	National Semiconductors
μPD70F3xxx	yes	32-bit	NEC
P8X592/8	no	8-bit	Philips
ST10F167	yes	16-bit	SGS Thompson
SAE81C90/91	passive	none	Siemens
C167CR	yes	16-bit	Siemens
TMP88PP87	yes	8-bit	Toshiba
TCG54AF	yes	none	Toshiba

are retrieved from the CAN controller receive buffer areas before they are overwritten by further incoming messages. This is especially true at high network speeds, as shown in Table 1.

Given this data, it is generally not possible for a processor to retrieve a message from the CAN controller buffer and process it before the next message arrives. In some cases, the CAN hardware implements its own buffering scheme whereby multiple areas of buffer memory in the hardware are used to store incoming telegrams. However, in most cases received messages must be temporarily stored by the software in memory so that they can be processed at a later date. If possible, copying to a memory buffer should be done using any on-chip facilities available for optimum data transfer speed. For example, Philips 80C592 microcontroller on-chip DMA facilities allow transfer of a CAN message from its CAN controller to internal data memory in a period of two instruction cycles.

An important consideration is the fact that when assessing the capability of the software avoiding message overruns, the number of back-to-back messages is not the only important factor. The length of each CAN message also plays a vital role. If all the messages in the 'burst' are of 8 bytes in length the processor has more time before having to deal with the next message in its receive buffer. If the burst contains messages of just a few bytes in length, a particular processor cannot deal with them in time. In other words, it is not only the number of messages in the burst but also the length of each message that counts. This was found out by the authors at an early stage when, after having built an automation cell where the master (based around a Basic CAN controller) could quite happily cope with the other microcontrollers on the network sending 8 byte messages, it could not cope when one or two of the processors were configured to send 1 or 2 byte messages.

Most of all, it is important that a device that does suffer from message overruns is able to detect this situation and notify the application using predefined error recovery mechanisms.

Hardware implementation

Two basic mechanisms exist for integrating CAN into a product. For existing products a standalone CAN controller can be interfaced to the same microcontroller using the external address and data bus. For new products a different microcontroller with integrated CAN interface can be used. The two are shown in Fig. 7.

A number of different manufacturers produce bus driver or transceiver chips and a few of them are shown in Table 2.

Table 3 shows a list of some of the integrated and standalone CAN controllers available. This is by no means an exhaustive list and many other variants exist. Most of the manufacturers listed here produce more than one CAN product. Note that most of these CAN controllers support the extended 29-bit identifier range although some only support 29-bit identifiers passively, i.e. they can be used on the same network as CAN controllers using the extended 29-bit ID but use standard 11-bit identifiers only.

Concluding remarks

The popularity of CAN (11 million chips were sold to the end of 1997) is due to its ease of use, to its extremely high efficiency and reliability and to the low costs of CAN implementations. CAN is as simple to use as a serial UART, and currently the cost of CAN controllers is still decreasing as CAN finds its way into more and more applications, not only in the car and automation industries, but also into fields such as medical instrumentation and domestic appliances.

References

1 'CAN specification 2.0', Parts A and B, Robert Bosch, September 1991
2 MACLAUGHLIN, R.: 'Introduction to CAN', CANopen Workshop, Savoy Place, London, October 1997
2 MONK, F.: 'Producer/consumer the new network paradigm', Fieldcomms UK '97 Conference, Hanover International Hotel, Leicestershire, October 1997
4 'CAN—a serial bus system not just for vehicles', CAN in Automation Organisation (CiA)
5 'P8x592 8-bit microcontroller with on-chip CAN', datasheet, Philips Semiconductors, June 1996
6 '8x196 microcontroller family', datasheet, Intel, June 1995
7 'MC68HC05X4, MC68HC705X4 Technical Data', datasheet, Motorola, 1996
8 'SAE81C90/91 standalone full CAN controller', datasheet, Siemens, January 1997
9 'CAN communication model and its implications', Holger Zeltwanger (CiA), CAN Solutions Directory, 1997, Miller Freeman

© IEE: 1999

The authors are with the Department of Electrical and Electronic Engineering, Merz Court, The University of Newcastle, Newcastle upon Tyne NE1 7RU, UK.

Authorized licensed use limited to: University of Maryland College Park. Downloaded on August 16,2010 at 19:37:42 UTC from IEEE Xplore. Restrictions apply.