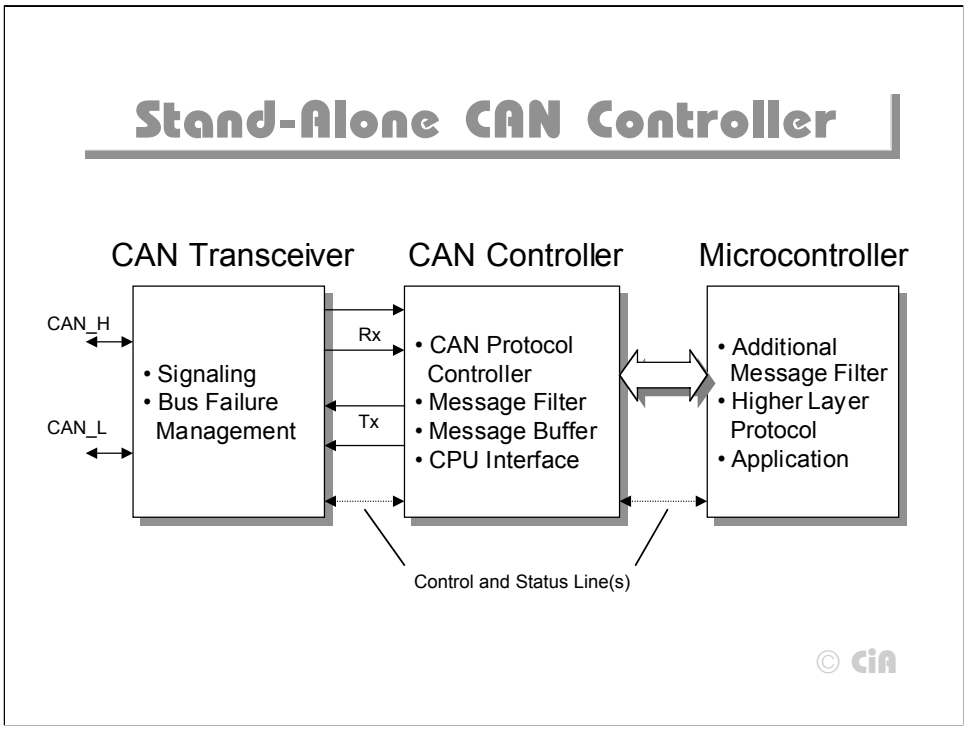
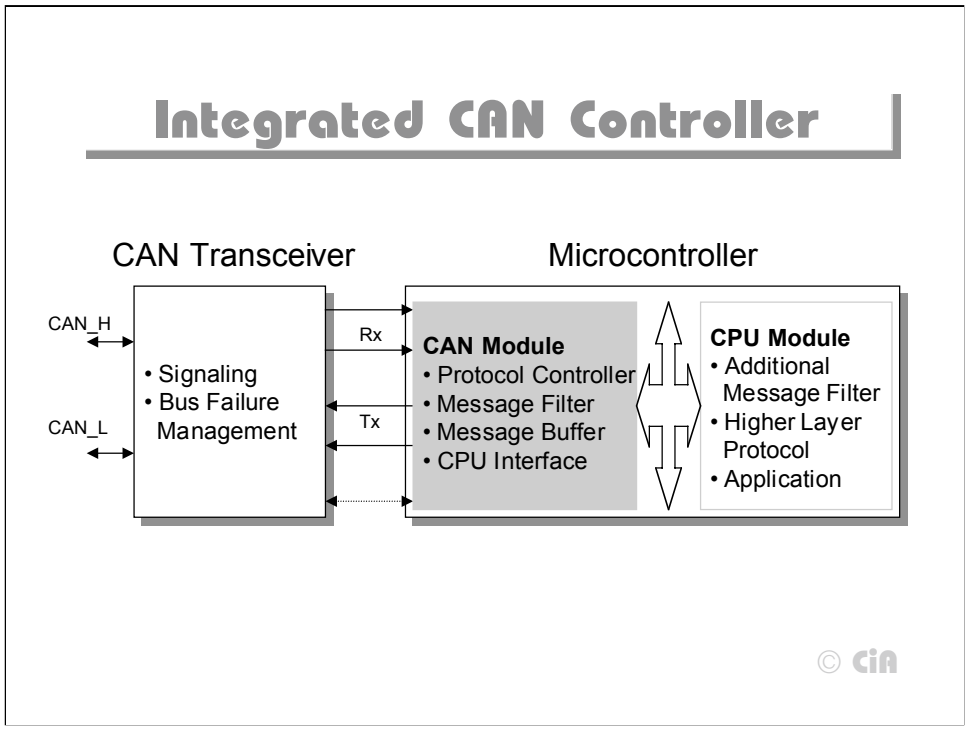
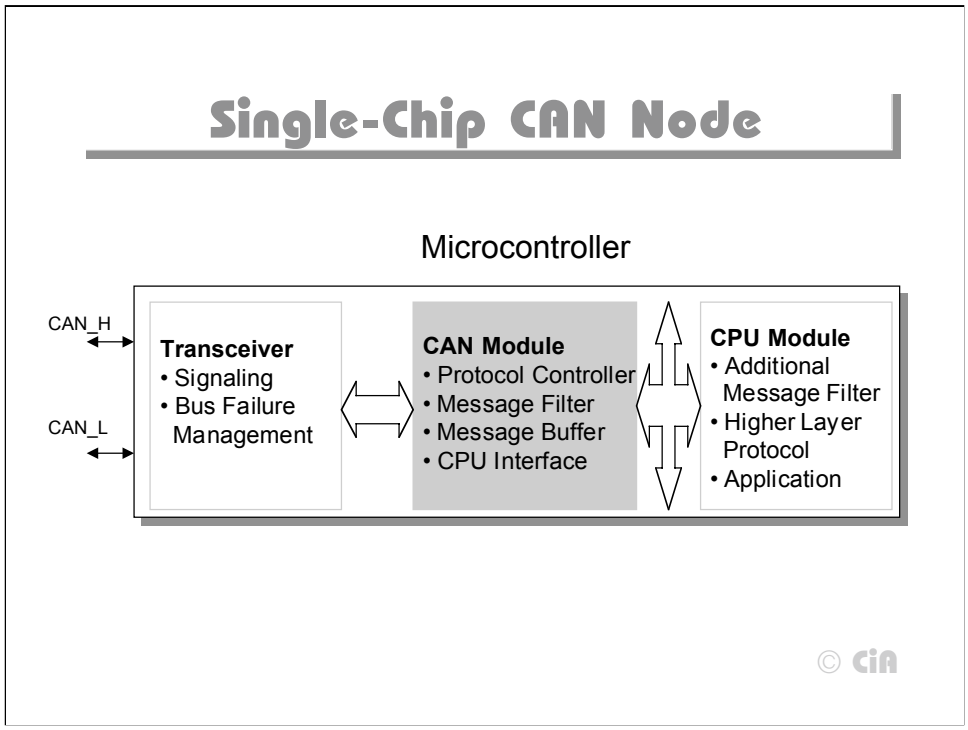

Several companies have implemented in silicon the international standardized Controller Area Network (ISO 11898). The layered data link layer provides basic communication services such as transmission of data and requesting of data. The CAN protocol also handles the detection of failures as well as the error indication. All existing implementations use the same protocol controller, which has to be compliant with the C or VHDL reference model from Bosch. The implementation differ in the acceptance filtering, the frame storage capabilities, and in additional, nice-to-have features.



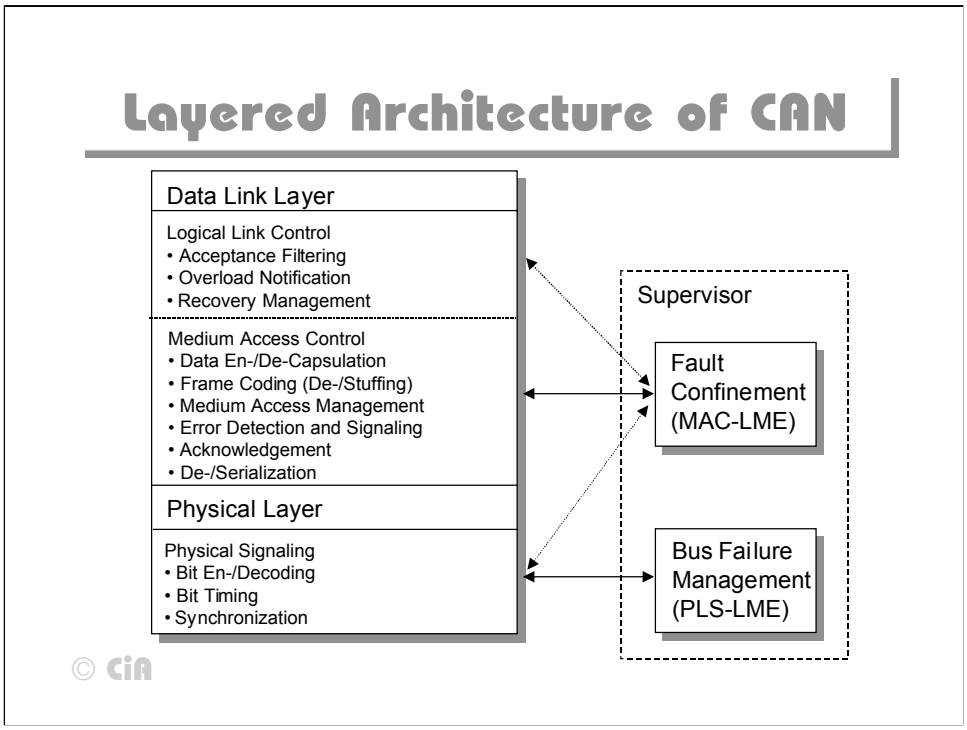

There are some trade-offs between stand-alone and integrated CAN peripherals. The integrated CAN peripheral is cheaper not only because of the chip price itself, the design of the printed circuits boards is easier, and space requirements and costs are lower. Although the software development costs are about the same for both solutions, software reusability may differ. Stand-alone CAN chips are designed to interface different CPUs allowing the software developed for one system to be reused in another system, even if the CPU is different. Software developed for an integrated CAN peripheral may not function on another CPU with on-chip CAN.



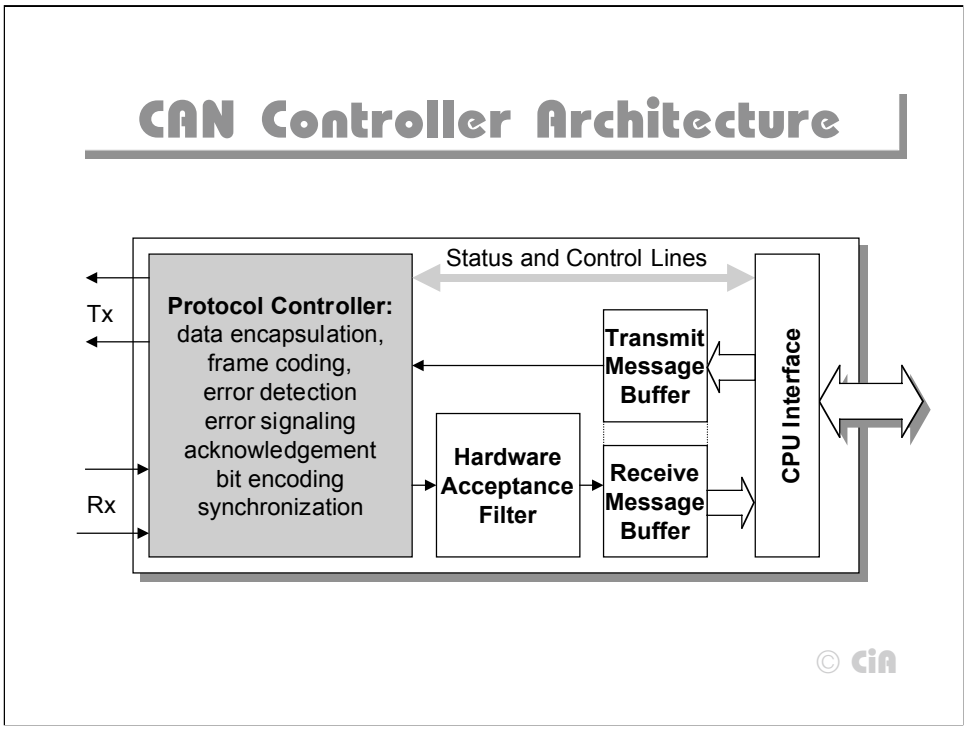

Integrated CAN peripherals cause a lower CPU load than do stand-alone controllers. The most critical factor is the amount of time required to read/write to the CAN peripheral. In the case of an on-chip CAN controller, the CAN registers are addressed using the internal address/data bus designed for high-speed access. In the case of a stand-alone CAN chip the CPU uses its external address/data bus or a serial communications link, which of course is slower. The CPU load on an on-chip CAN is approximately one-half of a stand-alone CAN chip. A CAN node using an integrated CAN peripheral has a reliability advantage over a system using a stand-alone CAN chip because of its smaller form factor. CAN chips are shipped in large numbers and the combined price of a CPU and a stand-alone CAN chip is still competitive. However, as CAN gains more market acceptance, CPUs with on-chip CAN will be the solution of choice.



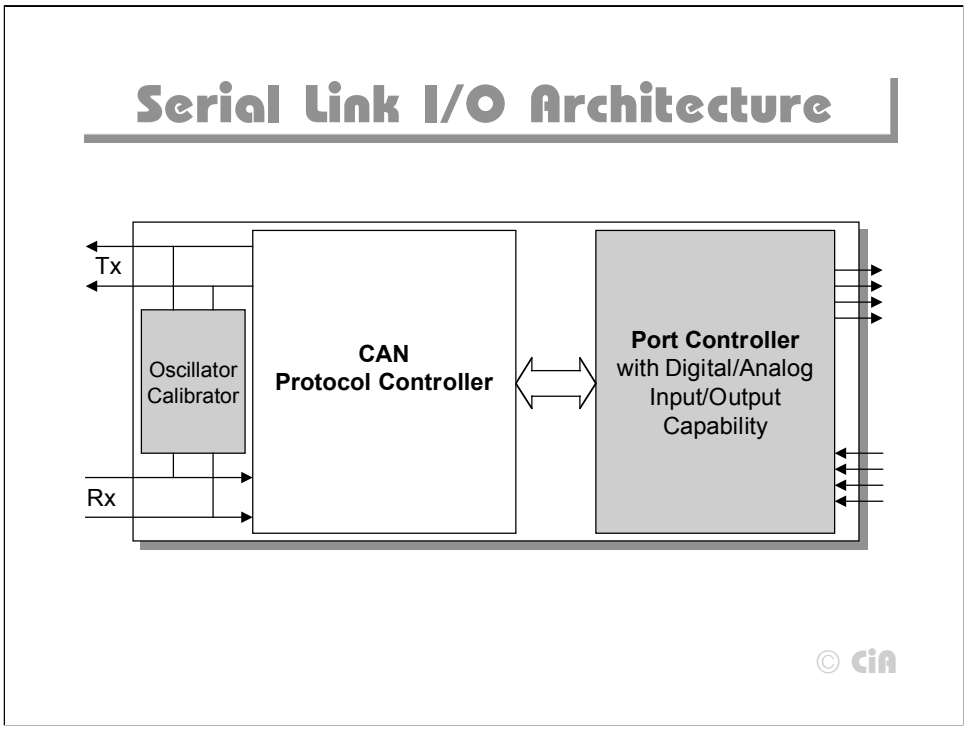

The trend to integrate also the transceiver on the same chip as CAN controller and microcontroller is becoming reality. First products for the automotive applications are already designed. The main problem is the combination of different technologies on one single chip.



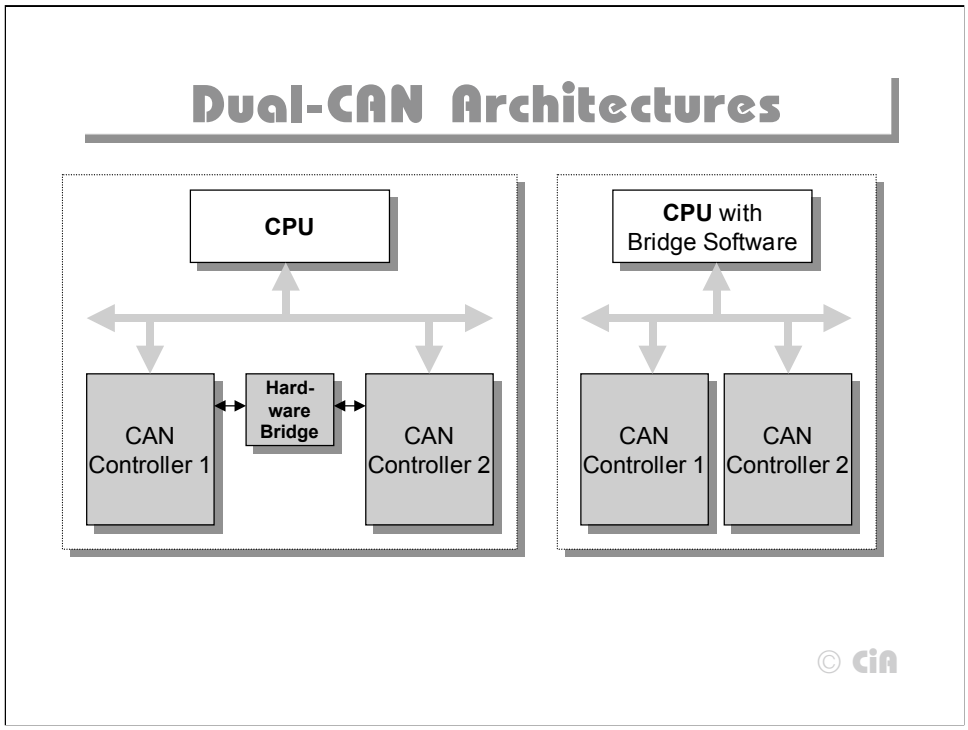

Communication is identical for all implementations of the CAN protocol. There are differences, however, in the extent to which the implementation takes over message transmission from the microcontrollers. All CAN controllers have a common structure consisting mainly of a CAN protocol controller, a hardware acceptance filter, message memory, and a CPU interface. The CAN protocol controller is responsible for handling all messages transferred via CAN bus lines. This includes tasks such as synchronization, error handling, arbitration, parallel/serial and serial/parallel conversions. There is no difference between the CAN protocol implementations, provided that they are based on the same protocol version.



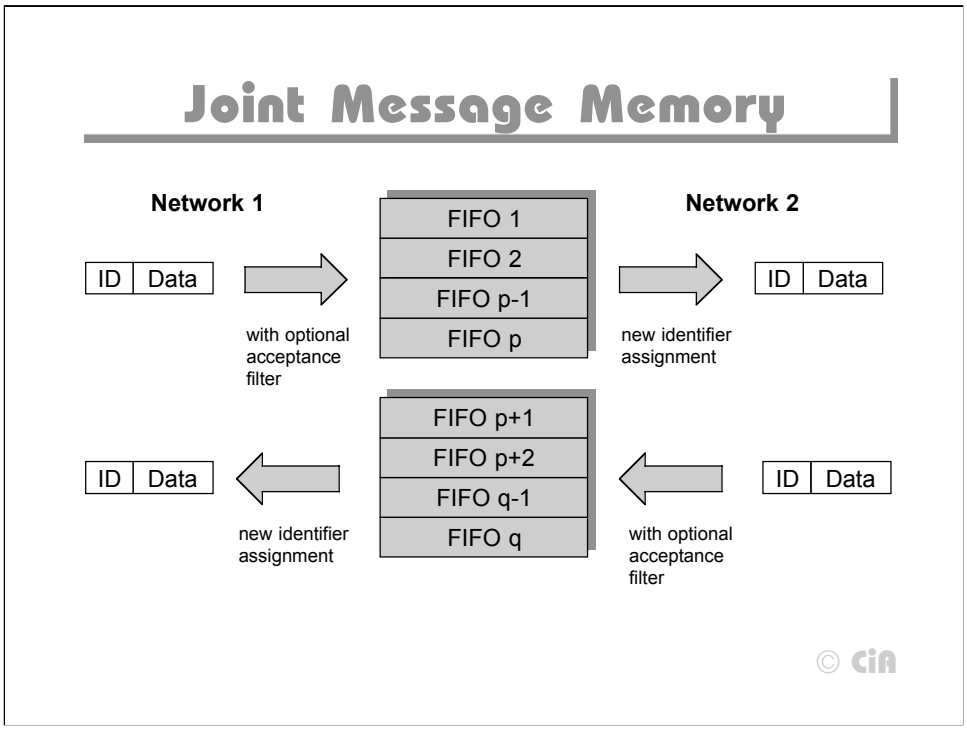

Besides the protocol controller each CAN controller provides a hardware acceptance filter to unburden the microcontroller from filtering all messages. In addition, the CAN controller implements message buffers for CAN data frames to be transmitted and for those which are received. The acceptance filter and the message buffers as well as the CPU interface are implementation-specific. These functions make the difference between the CAN implementations.




So-called Serial Link I/O (SLIO) components are CAN controllers without programming capabilities. They are pre-configured CAN interfaces that require a programmable CAN node to control this node.

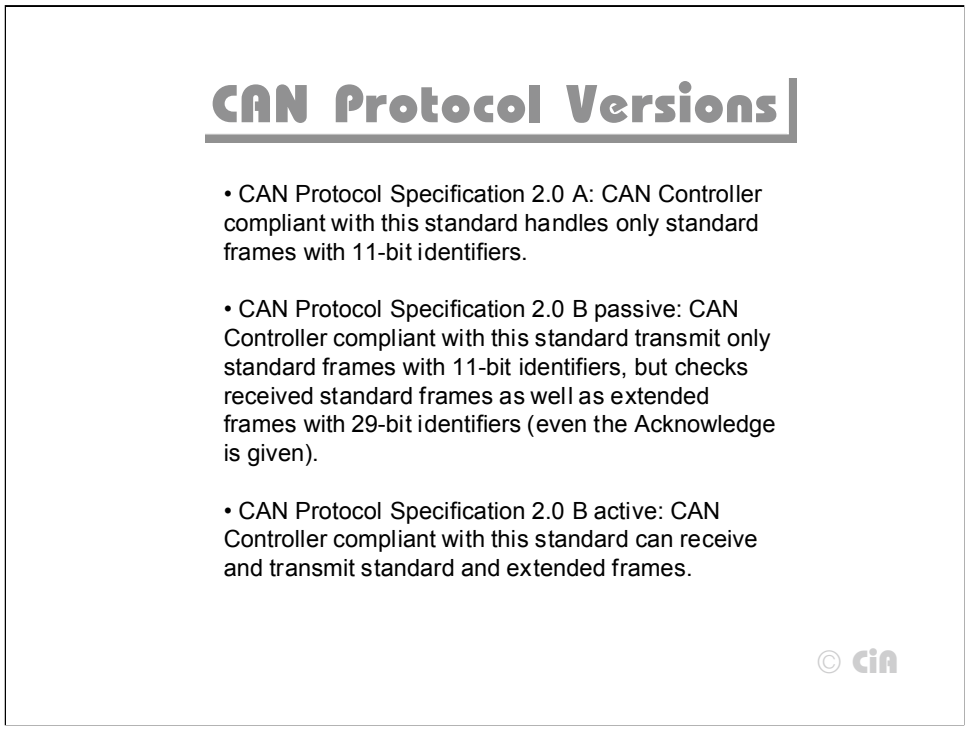



Several CAN manufacturers have implemented two CAN modules on one microcontroller to support bridge gateway developments. There are several ways of transferring data between different CAN networks. Any realistic, i.e. efficient solution is located somewhere between the following two strategies: transfer and filtering of objects is either done entirely by the microcontroller, or is handled by additional logic. For such double CAN controllers message storage capability is required not only for data frames received but also for those to be transmitted. Where both CAN modules use the same object memory it is possible to receive a message on one network and to transmit on the other without the interaction of the microcontroller. This automatic transmission feature is also possible for Remote Frames. Different identifiers on the source and the destination network are also supported. Some of these implementations provide totally flexible allocation of the message mailboxes. Of course, it is possible to run different baud rates on the networks.



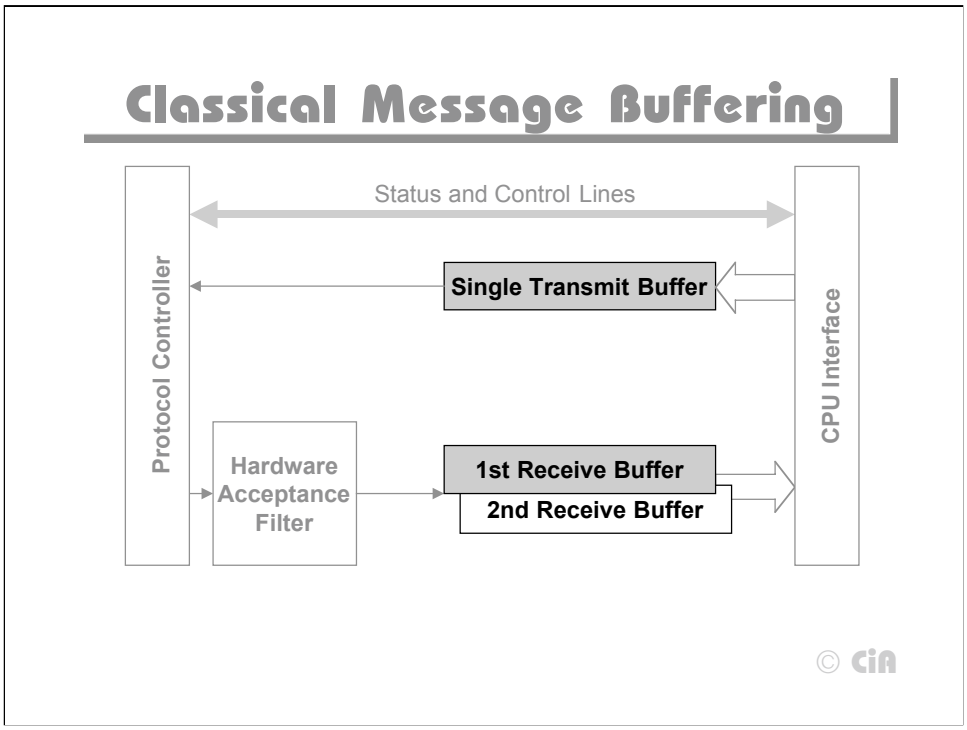

CAN controllers with bridge functionality may have implemented a configurable data FIFO. The received CAN frames are filtered and stored in the FIFO, and may be transmitted to the other CAN network with a different identifier.

When both networks are running different baudrates, the CAN bridge controller must implement to independent bus timing logic. To avoid data corruption or loss of data a FIFO structure is a reliable solution.

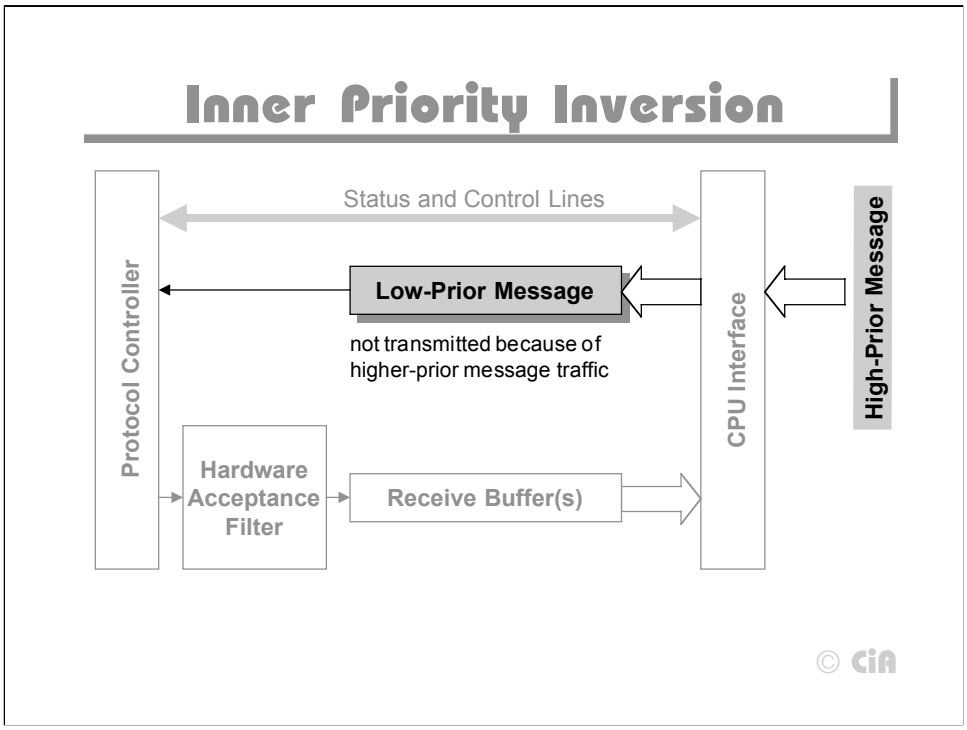



CAN (Controller Area Network), the serial bus system originally developed by Bosch has been standardized in ISO 11898. Different versions have been specified in the course of CAN's history. The Bosch CAN specification version 1.1 are upwards compatible to version 1.2, which is now the current Bosch CAN version 2.0 part A. All these versions describe the Standard CAN message format with 11-bit identifiers. In contrast to version 1.1, CAN version 1.2 allows the use of low-cost ceramic resonators at a bus speed of up to 125 kbit/s. A quartz oscillator is required for the full bus speed range of the CAN protocol (1 Mbit/s). In an existing CAN network oscillator accuracy is determined by the CAN node with the highest accuracy requirement. Ceramic resonators can only be used if all the nodes in the network use the enhanced protocol.

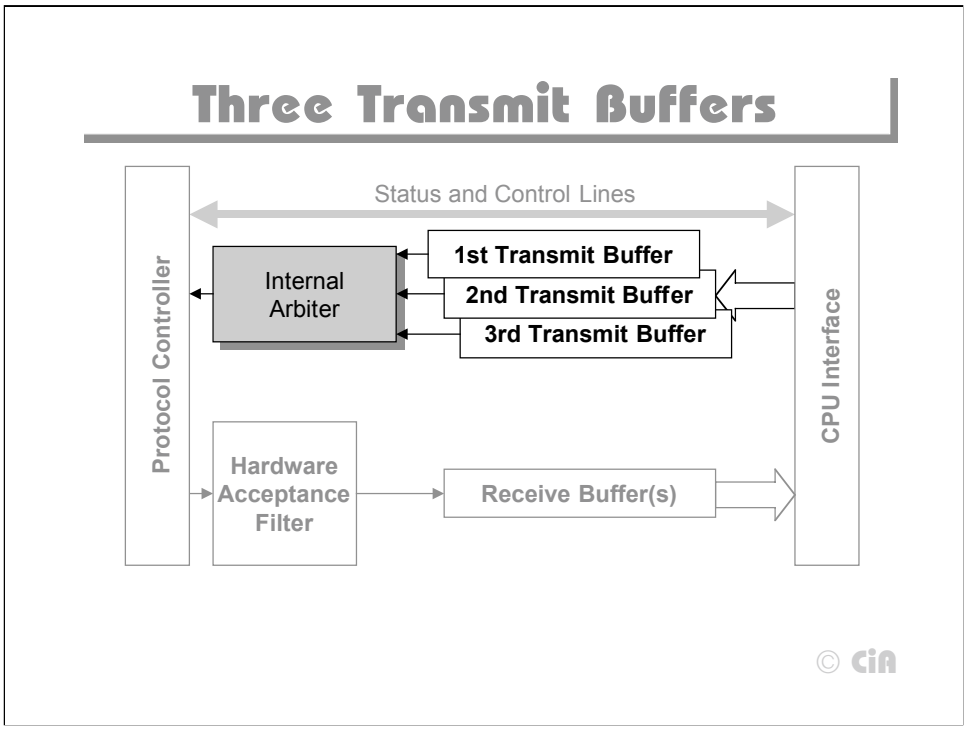
New CAN controller designs must be compliant in minimum with CAN 2.0 B passive.



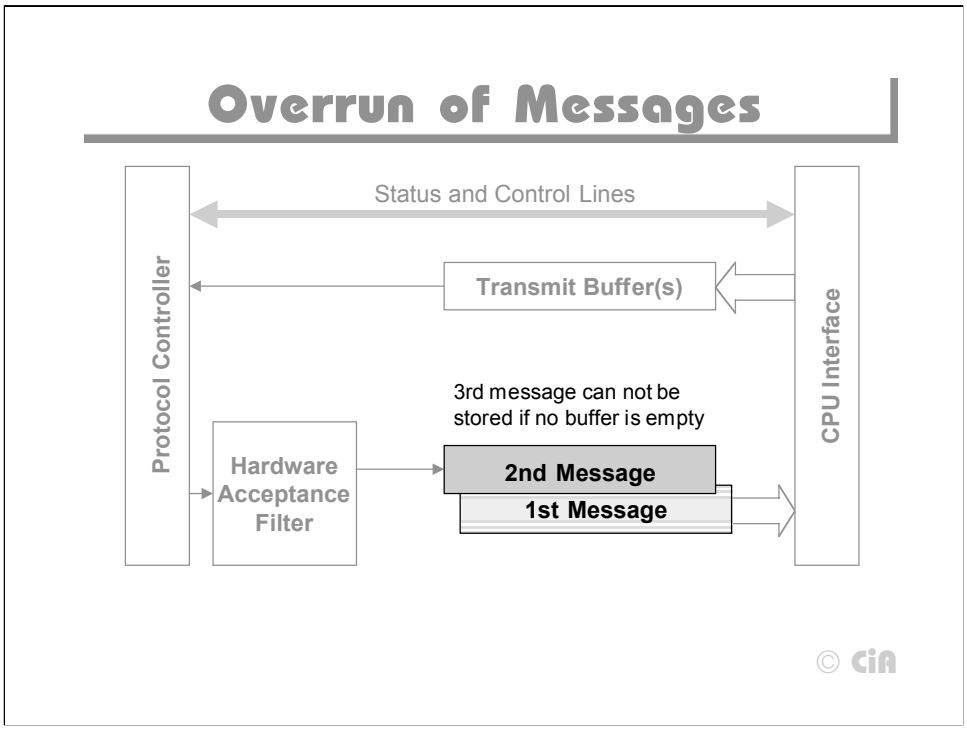

CAN controllers with intermediate buffers (formerly called BasicCAN chips) have the logic necessary to create and verify the bitstream according to the CAN protocol implemented as hardware. The simplest CAN controllers with intermediate buffer have only two reception and one transmission buffers.




If only a single transmit buffer is used inner priority inversion may occur. Because of low priority a message stored in the buffer waits until the ”traffic on the bus calms down”. During the waiting time this message could prevent a message of higher priority generated by the same microcontroller from being transmitted over the bus.

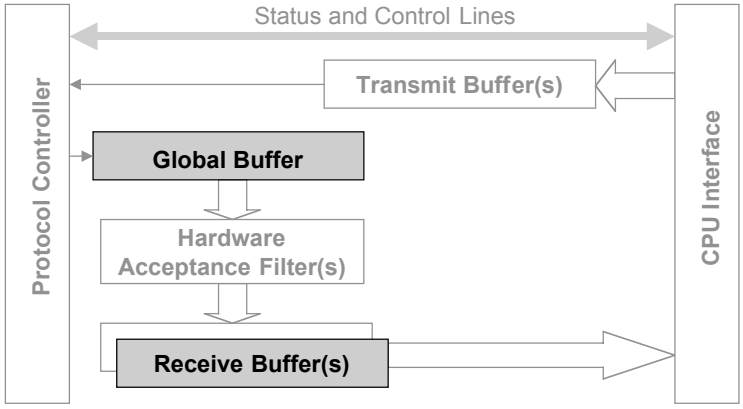


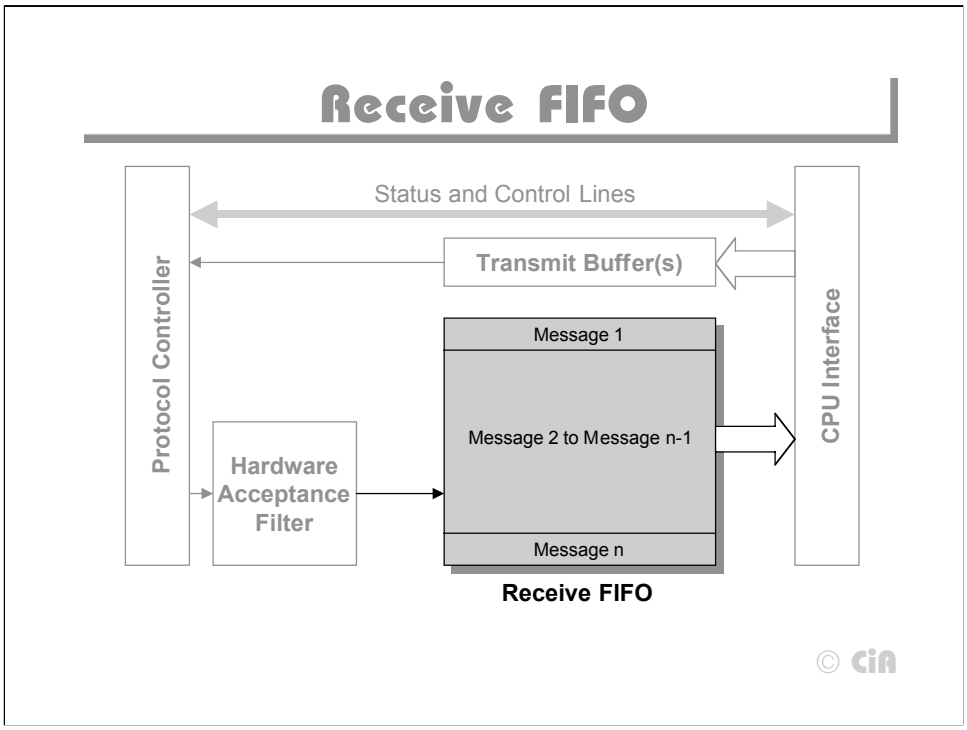

To overcome the inner priority inversion problem some of today's CAN controllers with an intermediate buffer provide more than one transmit buffer.



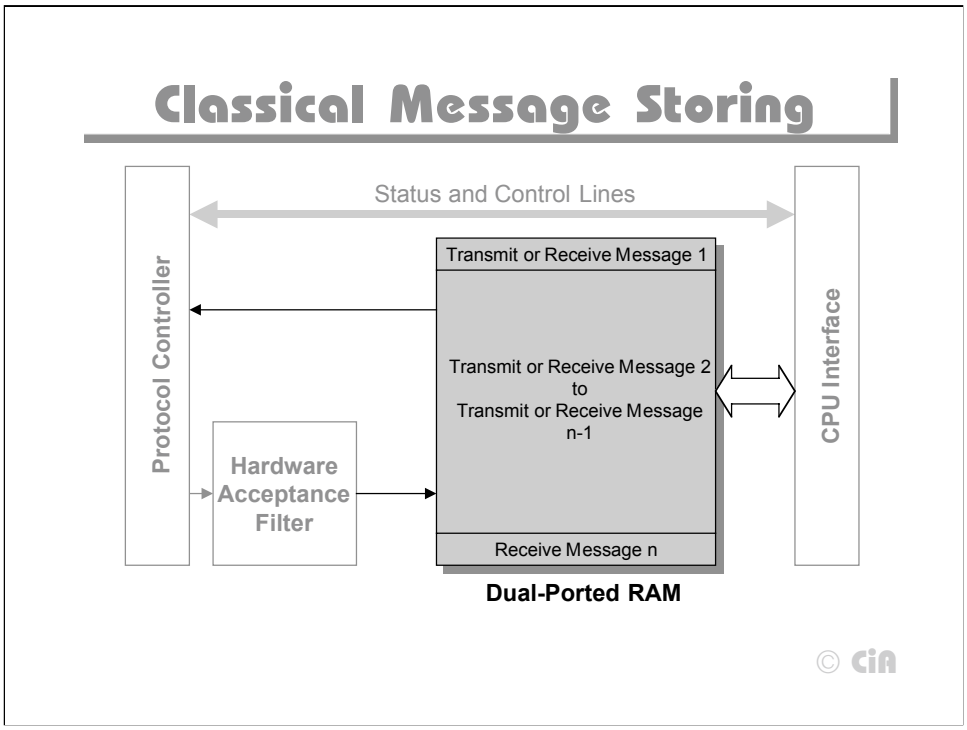

Implementing only a few reception buffers may cause the loss of data frames if the microcontroller is not fast enough to handle the interrupt requests produced by received messages.

# Global Receive Buffer

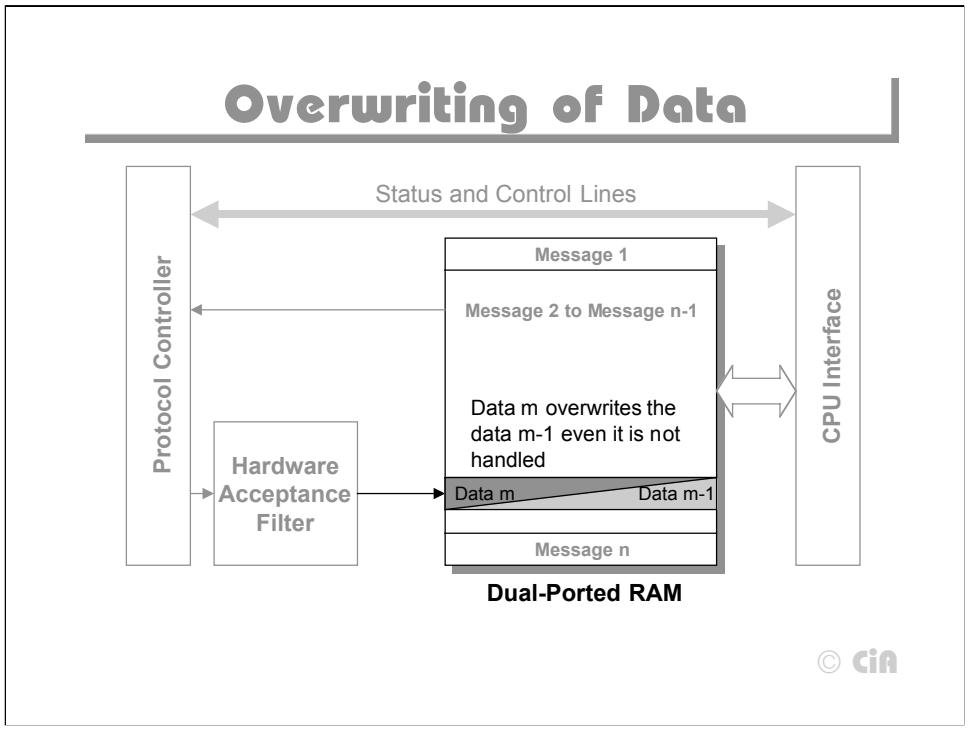




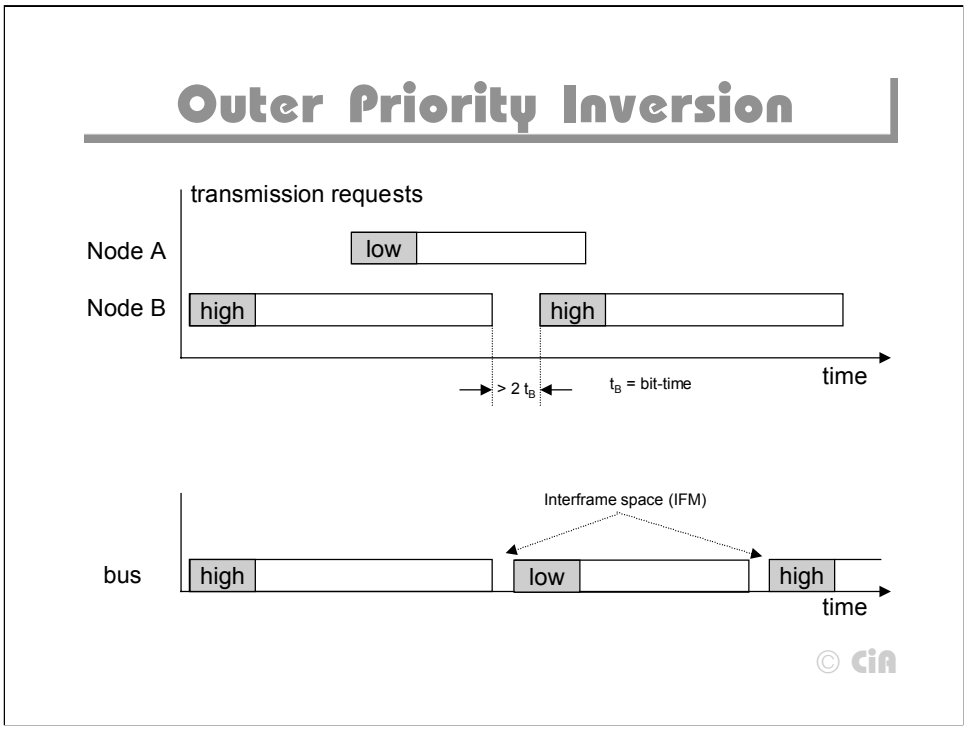

Modern CAN controllers with intermediate message buffering capabilities use deeper FIFO buffers, e.g. 64-byte. In the case of a data overrun condition, a CAN controller of this kind can optionally generate an overrun interrupt.



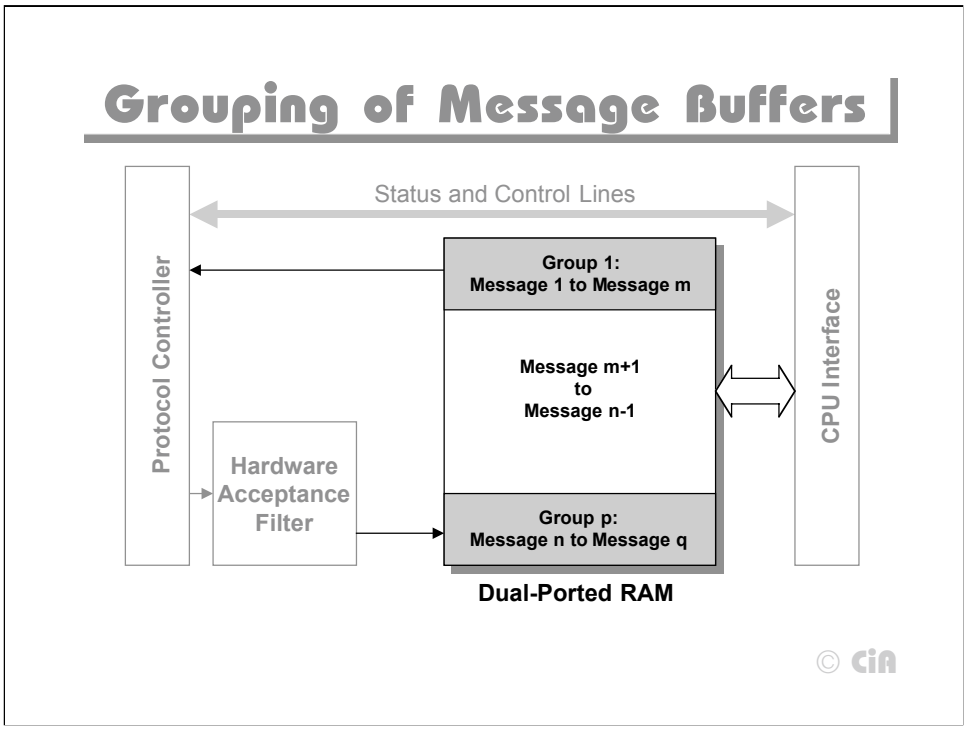

CAN controllers with object storage (formerly called FullCAN) function like CAN controllers with intermediate buffers, but also administer certain objects. The interface to the following microcontroller corresponds to a dual-ported RAM. Messages received are stored in the defined memory area, and only will be updated if a new message with the very same identifier is received.



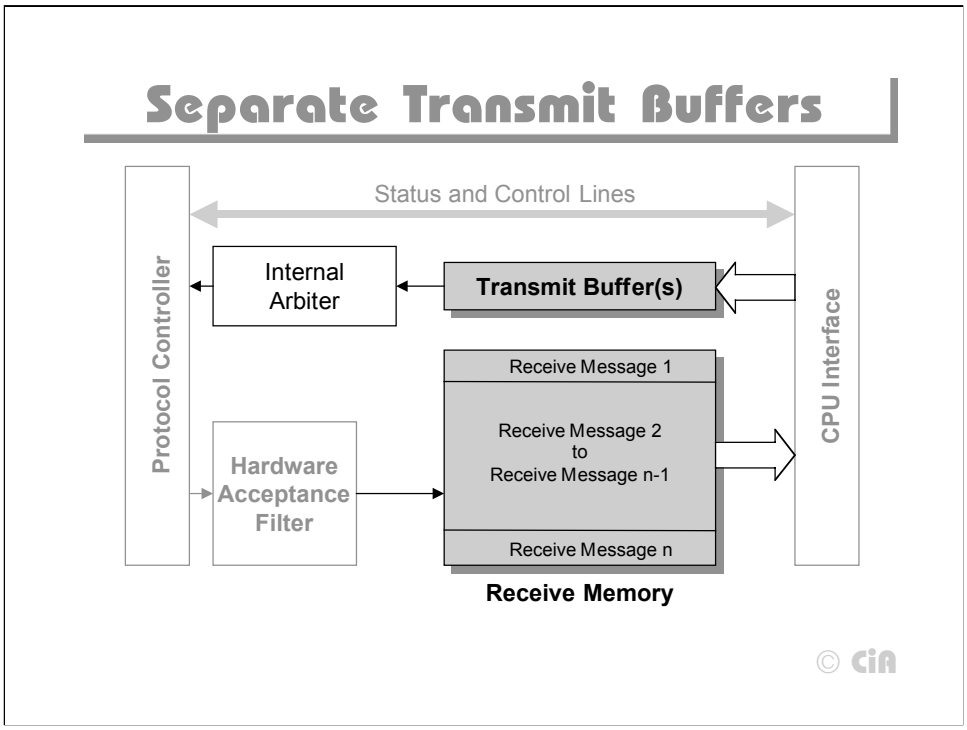

No message will be lost, but a newer one may overwrite the previous data. This dual-ported memory is practically restricted to a limited number of objects. Implementations are available today for between 16 and 64 object memories.



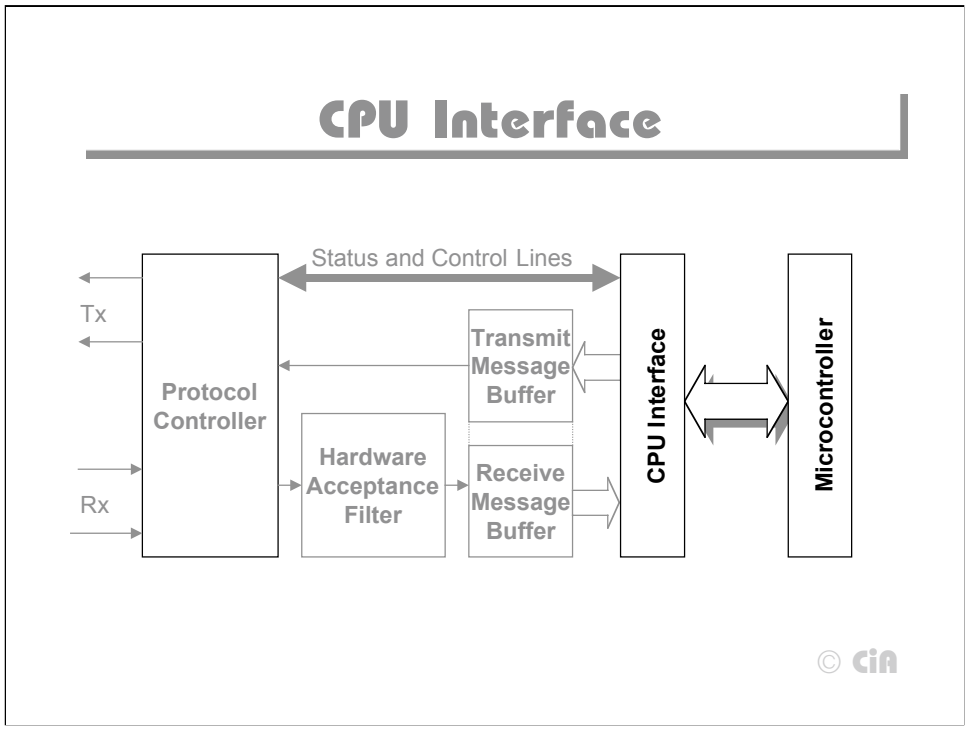

The problem of outer priority inversion may occur in some CAN implementations. Let us assume that a CAN node wishes to transmit a package of consecutive messages with high priority, which are stored in different message buffers. If the interframe space between these messages on the CAN network is longer than the minimum space defined by the CAN standard, a second node is able to start the transmission of a lower prior message. The minimum interframe space is determined by the Intermission field, which consists of 3 recessive bits. A message, pending during the transmission of another message, is started during the Bus Idle period, at the earliest in the bit following the Intermission field. The exception is that a node with a waiting transmission message will interpret a dominant bit at the third bit of Intermission as Start-of-Frame bit and starts transmission with the first identifier bit without first transmitting an SOF bit. The internal processing time of a CAN module has to be short enough to send out consecutive messages with the minimum interframe space to avoid the outer priority inversion under all the scenarios mentioned.




To optimize transfer of segmented data with the same or consecutive identifiers some CAN implementations support the grouping of message buffers.




For nodes requiring to transmit the complete range of objects, some CAN implementations provide a separate transmit buffer capability without loosing the benefits of the advanced acceptance filtering.




Most of the stand-alone CAN controllers support several modes to connect the microcontroller. Integrated CAN controllers have only one optimized CPU interface.

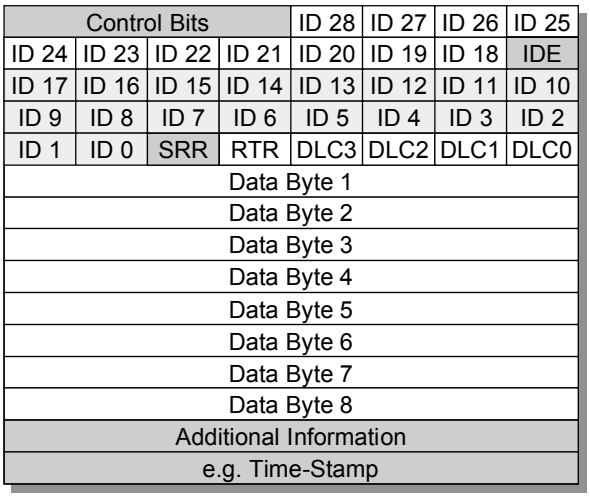
## Simple Message Structure

| ID 10 | ID 9 | ID 8 | ID 7 | ID 6 | ID 5 | ID 4 | ID 3 |
|---|---|---|---|---|---|---|---|
| ID 2 | ID 1 | ID 0 | RTR | DLC3 | DLC2 | DLC1 | DLC0 |
| Data Byte 1 | | | | | | | |
| Data Byte 2 | | | | | | | |
| Data Byte 3 | | | | | | | |
| Data Byte 4 | | | | | | | |
| Data Byte 5 | | | | | | | |
| Data Byte 6 | | | | | | | |
| Data Byte 7 | | | | | | | |
| Data Byte 8 | | | | | | | |



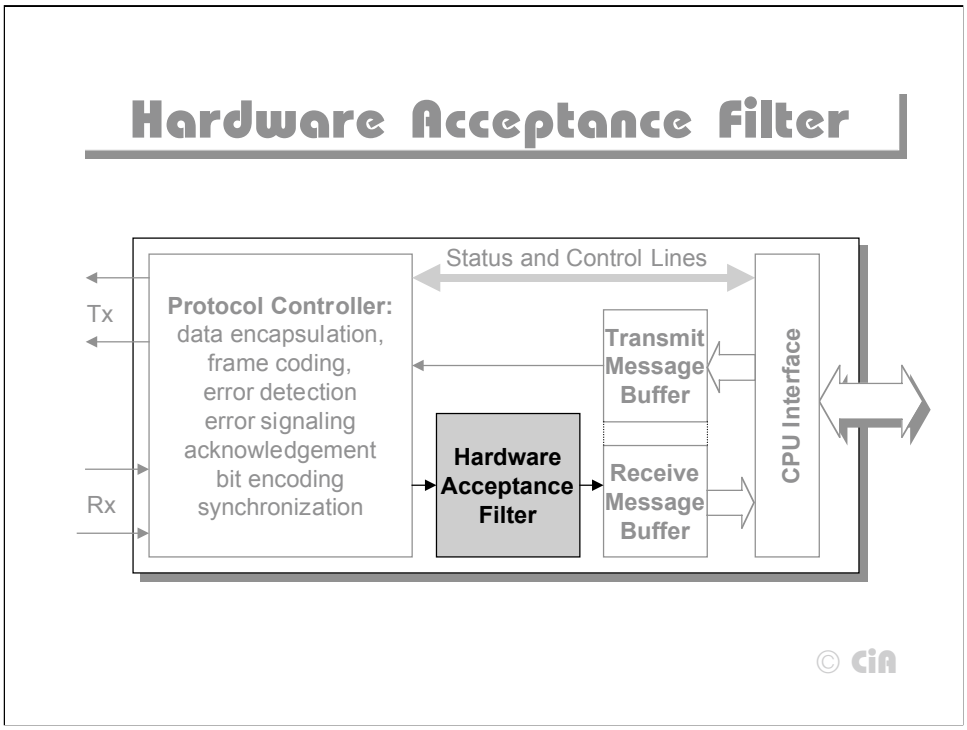
The message buffer structure has to provide in minimum the data bytes, the 11-bit identifier, and the data length code.

## Advanced Message Structure



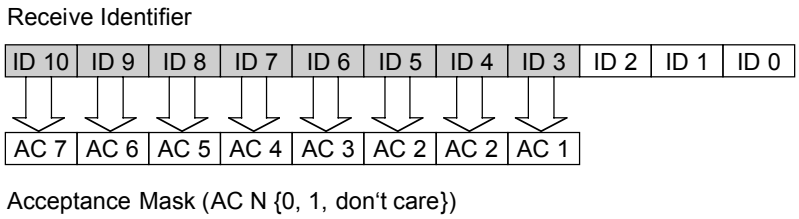



Implementation-specific, the CAN controller may provide additional information such control bits, further arbitration bits (29-bit identifiers), and time-stamps



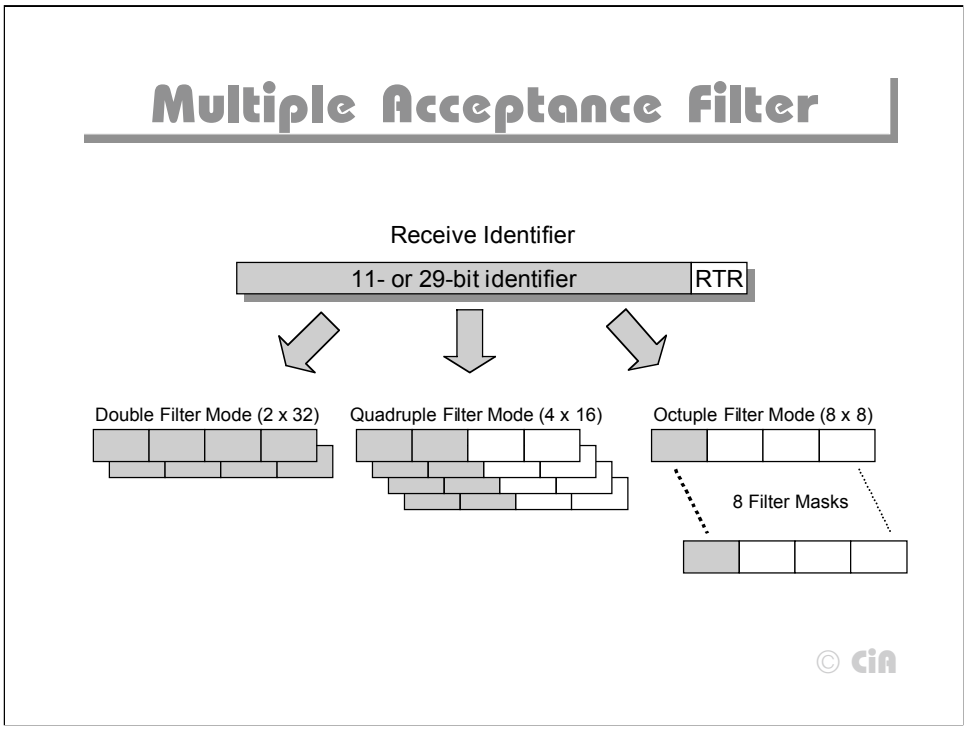

One of the most significant features of a CAN protocol controller is the acceptance filtering capability. All CAN implementations provide some hardware acceptance filters to relieve the microcontroller from the task of filtering those messages which are needed from those which are not of interest.

## Simple Acceptance Filtering

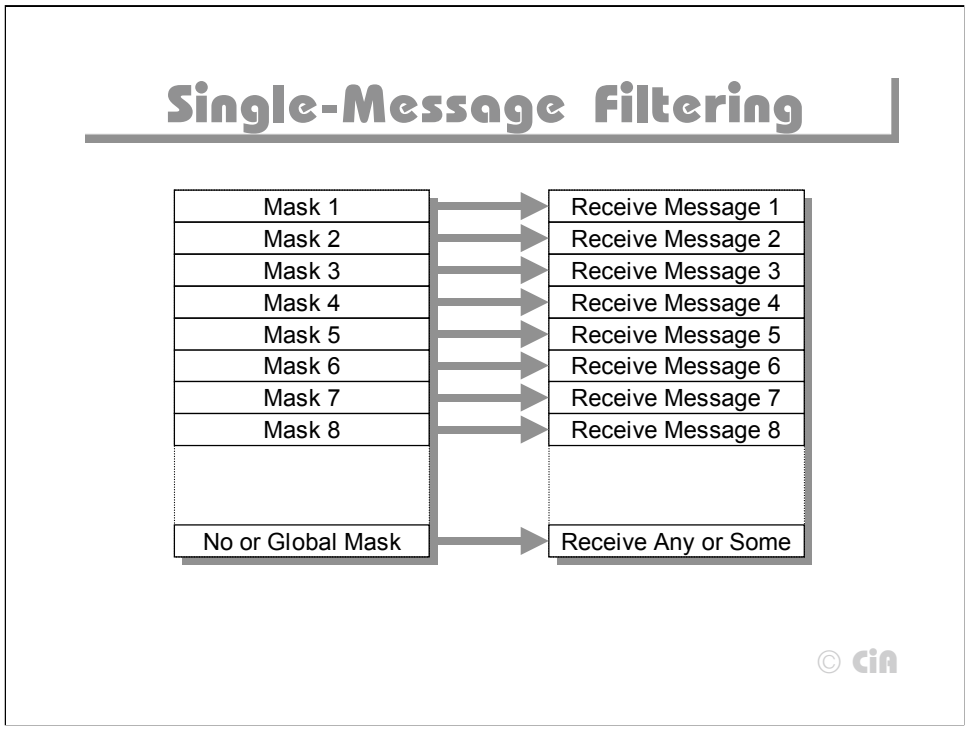





A very simple hardware acceptance filter is a single 8-bit mask. Each bit may be set to 0, 1 or don’t care.




Most of the modern implementations provide a multiple acceptance filtering with programming capability, which allows a more flexible filtering strategy as the simple acceptance filtering.




The so-called FullCAN implementations allow a single-message filtering. Each of the receive message buffers is assigned to a specific identifier. In many implementations one or two message buffers can be used to receive all data frames (so-called BasicCAN option).

## Remote Frame Handling

- ❏ CAN controller with receive buffer or receive FIFOs answers Remote Frames only under CPU control.
- ❏ CAN Controller with standard message storing answers Remote Frame automatically without CPU control.
- ❏ CAN Controller with advanced message storing answers Remote Frames automatically *and* optionally under CPU control.



Most of the so-called FullCAN implementations provide automatic transmission of remotely requested frames. This is not required in some applications because there is no control if the data frame is still valid. In cases where the microcontroller is gone, the CAN controller could transmit information which is no longer valid. Some CAN chips can therefore disable the automatic response to Remote frames and the CAN controller will indicate the reception of a Remote frame by an interrupt. The microcontroller then has to transmit the requested Data frame.

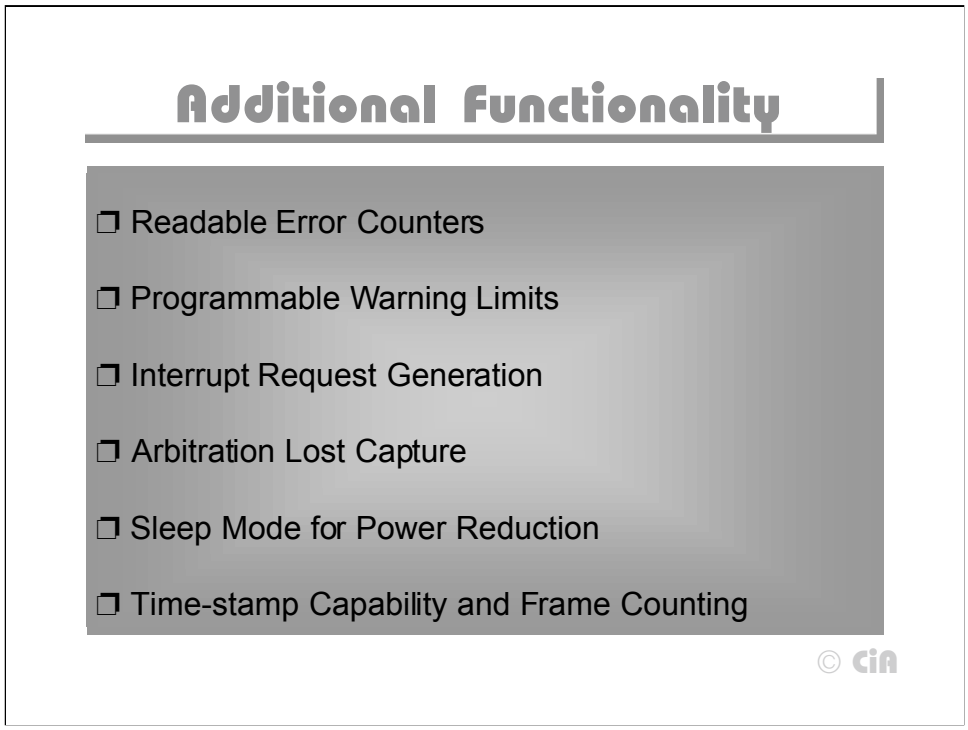
## DLC > 8 in Transmit Messages

The behavior of CAN Controllers regarding DLC greater than 8 in transmitted frames is different:

- Some CAN implementations ignore “out of range” DLC, and transmit 8 byte of data and the „wrong“ DLC.
- Some CAN implementations ignore not allowed DLC, and transmit 8 byte of data and a DLC of 8.
- Some CAN implementations don‘t transmit any frame, if they detect not allowed DLC.



All these implementations are compliant with the ISO 11898-1 standard. The CAN protocol allows to use all DLCs without any restrictions, the DLCs 9 to 15 may be used for application-specific purposes.




Readable Error Counters. For diagnostic purposes some of the CAN controllers provide readable Receive and Transmit Error Counters.

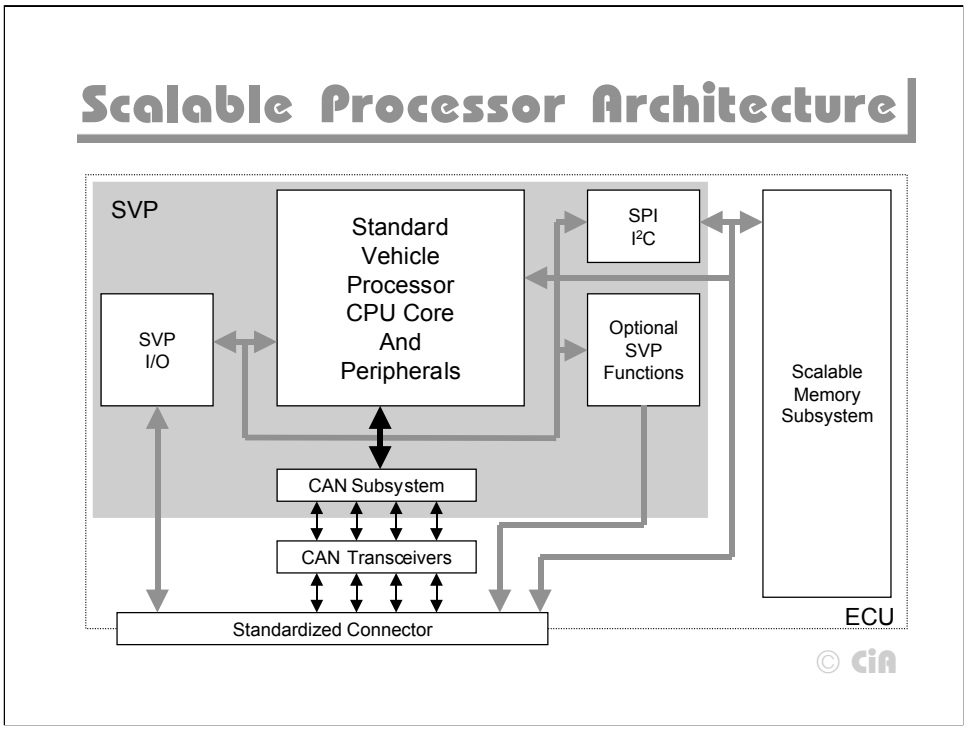
Programmable Warning Limits. An Error Code Capture Register can distinguish between the different error types.

Interrupt Request Generation. Besides the interrupt requests for reaching the passive error state and the bus-off state, other interrupt requests are optionally generated by reaching the warning limit, reception of a message, successfully transmitting a message etc.

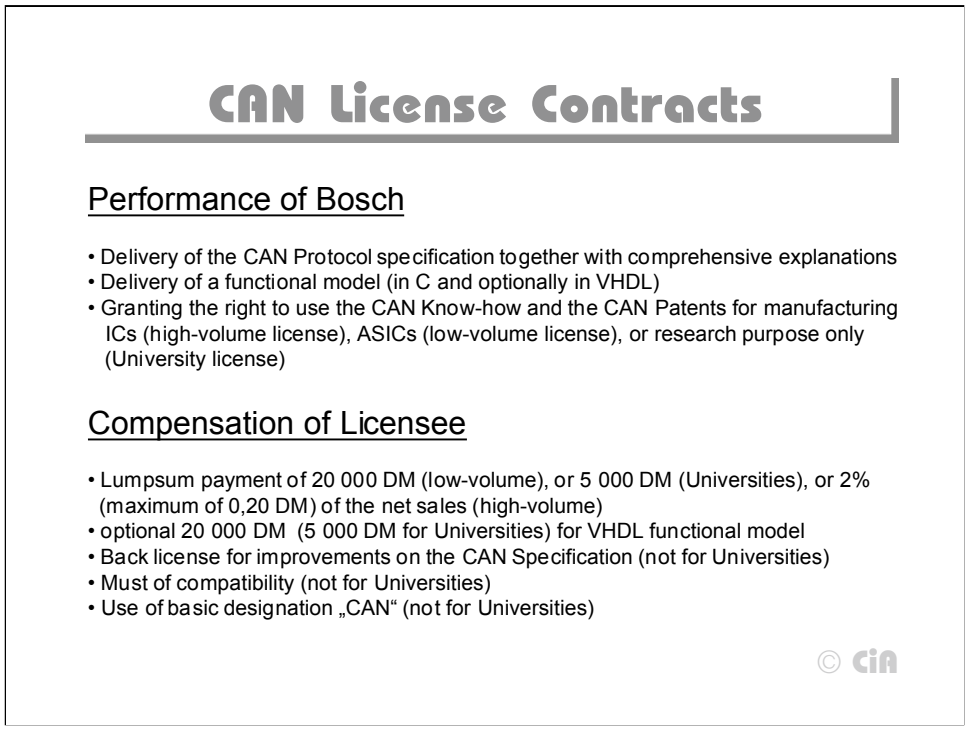
Arbitration Lost Capture. Using this feature in combination with the Single Shot option (is specified in the ISO 11898 frame scheduling option), the system designer can check the communication behavior.

Sleep mode for Power Reduction. During sleep mode the clock is switched off, only the wake-up logic is still active.

Time-stamp Capability and Frame Counting. This is especially useful in real-time applications. Some CAN chips implement 16-bit frame counters.




The Scalable Processor Architecture allows developing ECUs (electronic control units) suitable for different applications. They may be scalable in communication, functionality, and performance. They should provide several CAN interfaces for different purposes. Such an Standard Vehicle Processor should be able to handle about 15,000 interrupts per second, which is required when the SVP is connected to one high-speed, two low-speed, and one multi-media networks.




Bosch holds several patents on the CAN protocol, most of them valid for a further five to ten years. This means that it is not possible to implement the CAN protocol in hardware, firmware or software without a license from Bosch. Bosch provides licensees with the CAN functional model in C or VHDL and grants the right to use CAN know-how and CAN patents for the manufacture of integrated circuits or ASICs.

Bosch holds the following patents relevant for the CAN protocol:

| | |
|---|---|
| DE 35 46 684 | 22.2.2005 |
| DE 35 46 662 | 22.2.2005 |
| DE 35 46 664 | 22.2.2005 |
| US 500 16 42 | 19.3.2008 |
| US 530 33 48 | 12.4.2011 |
| US 552 42 13 | 4.6.2013 |
| JP 198 92 53 | 17.2.2006 |
| JP 204 11 07 | 17.2.2006 |
| JP 254 55 08 | 17.2.2006 |
| FR 851 7780 | 2.12.2005 |