II INTERNATIONAL CONFERENCE
TRANSPORT SYSTEMS TELEMATICS TST`02

ZESZYTY NAUKOWE
POLITECHNIKI ŚLĄSKIEJ 2002
TRANSPORT z.45, nr kol. 1570

CANBUS,VHDL,IP Core,
electronics, microprocessor systems,
traffic control, data transmission

Włodzimierz WRONA[1]
Wojciech SAKOWSKI[2]
Tomasz JAKÓBIEC[3]
Paweł JEŻ[3]

CANBUS CONTROLLER IMPLEMENTATION AS AN EXAMPLE OF IP CORE MODELLING FOR TRANSPORT SYSTEMS

The paper presents the Controller Area Network (CAN) controller, which has been implemented as an virtual component (IP core) in VHDL simulation environment. CAN is a serial bus system designed for networking "intelligent" devices as well as sensors and actuators within a system. CAN was originally developed for passenger car applications. Nowadays, the majority of European carmakers are using CAN networks at least for the engine management.

IMPLEMENTACJA KONTROLERA MAGISTRALI CANBUS JAKO PRZYKŁAD MODELOWANIA W JĘZYKU HDL ELEKTRONICZNYCH URZĄDZEŃ STEROWANIA I KONTROLI PRACUJĄCYCH NA RZECZ TRANSPORTU

Referat przedstawia szkic sprzętowego kontrolera magistrali transmisji szeregowej CAN, zaimplementowanego w języku opisu sprzętu VHDL. Komponent przeznaczony jest do zastosowań w systemach SoC i pSoC.

CAN jest szeregową magistralą danych zaprojektowaną z myślą o zastosowaniach motoryzacyjnych, pozwala tworzyć zintegrowane i niezawodne systemy zarządzania pracą silnika, hamulców i innych systemów pojazdu. W referacie oprócz opisu kontrolera znajduje się krótka charakterystyka magistrali CAN.

1. INTRODUCTION

CAN or Controller Area Network is a serial bus system designed for networking "intelligent" devices as well as sensors and actuators within a system. CAN was originally developed for passenger car applications. Nowadays, the majority of European carmakers are using CAN networks at least for the engine management. Since 1992, Mercedes-Benz has been using CAN in their upper-class passenger cars. Other car manufacturers have followed the example of their peers from Stuttgart and now usually also implement two CAN networks

[1] Department of Electrical Engineering, Technical University of Bielsko-Biała, Willowa 2, 43-300 Bielsko -Biala, Poland, wwrona@aristo.pb.bielsko.pl
[2] Institute of Electronic, Silesian Technical University, Akademicka 16, 44-100 Gliwice, Poland
[3] Evatronix S.A., 1 Maja 8, 43-300 Bielsko-Biala, Poland, ipcenter@evatronix.com.pl

in their passenger cars. After Volvo, Saab, Volkswagen and BMW, now also Renault and Fiat use CAN in their vehicles.
CAN networks used in engine management connect several ECUs (electronic control units). Mercedes-Benz was the first manufacturer who implemented CAN. Most of the other European automobile manufacturers also have implemented a CAN high-speed network (e.g. 500k bit/s) in their power-engine systems.
Some passengers cars are equipped with CAN-based multiplex systems connecting body electronic ECUs. These networks running at lower data-rates, e.g. 125k bit/s. Most of them are using not the high-speed transceivers compliant with ISO 11898-2, but fault-tolerant transceivers compliant with ISO 11898-3. These multiplex networks link door and roof control units as well as lighting control units and seat control units.

In this paper we present the Controller Area Network (CAN) model, which has been implemented as an virtual component (IP core) in VHDL simulation environment.

2. AN IP CORE DEVELOPMENT METHODOLOGY

Over last three decades integrated circuits of ever growing complexity were used as components of electronic systems that were assembled on the printed circuit boards. Now such systems may be integrated into a single silicon die. This dramatic increase of complexity of integrated circuits, that can be manufactured today with state-of-the-art semiconductor technology, challenges the designer productivity. The only way to take advantage of what technology offers is to reuse existing designs of functional blocks in order to create complex systems.
The data format for description of such reusable designs may be different, but they share a common characteristic: from a point of view of their users they are *components*. But as they are just sets of design data rather than actual pieces of hardware they are referred to as *virtual components*.
One of the possible forms of virtual components is functional description of hardware blocks in hardware description languages. These descriptions are *synthesizable*, which means that an appropriate design software tool may automatically synthesize logical circuit structures that implement the function described.
Basic development steps in the creation of a virtual component include [1, 2, 3]:

- development of the macro specification,
- partitioning the macro into sub-blocks,
- development of the testing environment and test suite,
- design of sub-blocks,
- macro integration and final verification,
- prototyping the macro in FPGA,
- productization.

We use the documentation of an original algorithm as a basic for specification of the core modeled after it. The documentation usually does not contain all the details that are necessary to gain its full functionality. Therefore analysis of the documentation results in a list of ambiguities that have to be resolved. At a later stage of specification development we use an Excel spreadsheet to document all operations and data transfers that take place inside the core. A dataflow spreadsheet makes it easier to define proper partitioning of the macro into sub-blocks. The crucial issue in this process is distribution of functions between the sub-blocks,

definition of the structural interface and specification of timing dependences between them. We check the code with VN-Check tool from TransEDA to ensure that the rules are followed, violations are documented.
A testing environment is developed in VHDL simulator. Aldec's ActiveVHDL simulator that proved to be very effective in model development and debugging phase.
To completeness of the test suite is checked with a code coverage tool (VN-Cover from TransEDA). The tool introduces monitors into the simulation environment and gathers data during a simulation run. Incompleteness of the test suite may be a reason for leaving bugs in untested part of the code. Therefore we pursue 100% statement coverage during RTL simulation (i.e. each statement must be executed at least once during simulation of the complete test suite). Code coverage also helps to reveal redundancy of the test suite and same times the redundancy in the hardware under test. We can also use more sophisticated coverage measures like branch or path coverage.
The next step in the core development process is building a real prototype that could be used for testing and evaluation of the core. For the moment we can propose two popular technologies: Altera and Xilinx. The tests are run again on the SDF-annotated structural model.
The methodology was developed over the last few years during design of several algorithms, one of them – the CANBUS controller we present in this paper.

3. CAN BUS – SHORT INTRODUCTION

CAN is a serial bus system with multi-master capabilities, that is, all CAN nodes are able to transmit data and several CAN nodes can request the bus simultaneously [4]. The serial bus system with real-time capabilities is the subject of the ISO 11898 international standard and covers the lowest two layers of the ISO/OSI reference model. In CAN networks there is no addressing of subscribers or stations in the conventional sense, but instead, priorities messages are transmitted. A transmitter sends a message to all CAN nodes (broadcasting). Each node decides on the basis of the identifier received whether it should process the message or not. The identifier also determines the priority that the message enjoys in competition for bus access (Figure 1).
The relative simplicity of the CAN protocol means that very little cost and effort need to expended on personal training; the CAN chips interfaces make applications programming relatively simple. Introductory courses, function libraries, starter kits, host interfaces, I/O modules and tools are available from a variety of vendors permitting low-cost implementation of CAN networks.
The use of CAN in most of European passenger cars and the decision by truck and off-road vehicle manufacturers for CAN guarantees the availability of CAN chips for more than 10 years. Other high volume markets, like domestic appliances and industrial control, also increase the CAN sales figures. Up to spring 1999 there are more than 150 million CAN nodes installed.

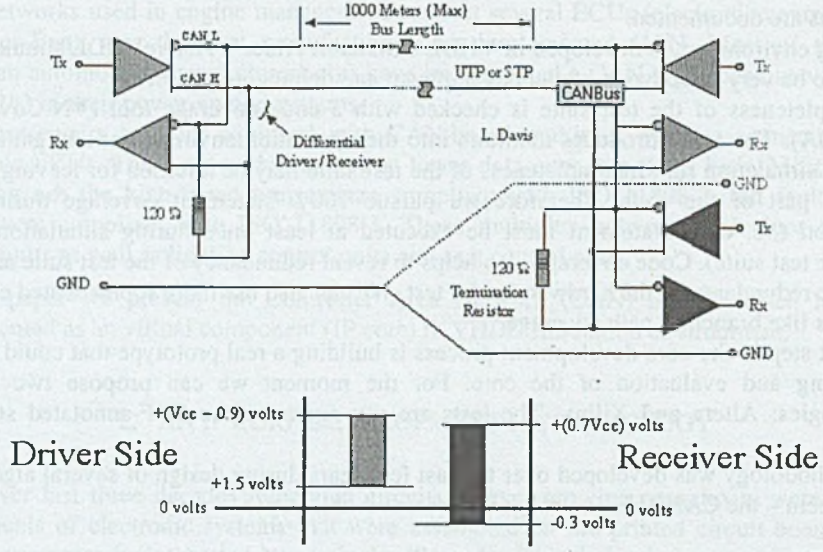

Fig.1. Electrical diagram

One of the outstanding features of the CAN protocol is its high transmission reliability. The CAN controller registers a stations error and evaluates it statistically in order to take appropriate measures. These may extend to disconnecting the CAN node producing the errors. Each CAN message can transmit from 0 to 8 bytes of user information. Of course, you can transmit longer data information by using segmentation. The maximum transmission rate is specified as 1 Mbit/s. This value applies to networks up to 40 m. For longer distances the data rate must be reduced: for distances up to 500 m a speed of 125 Kbit/s is possible, and for transmissions up to 1 km a data rate of 50 Kbit/s is permitted.

The acceptance and introduction of serial communication to more and more applications has led to requirements that the assignment of message identifiers to communication functions has to be standardized for certain applications. In standard format the identifier is 11 bits long and extended format 29 bits long.

CAN has the following properties:

Prioritization of messages, guarantee of latency time, configuration flexibility, multicast reception with time synchronization, system wide data consistency, multimaster, error detection and signaling, automatic retransmission of corrupted messages as soon as the bus is idle again, distinction between temporary errors and permanent failures of nodes and autonomous switching off of defect nodes.

4. CAN BUS – MESSAGE TRANSFER

Message transfer of CANBUS is manifested and controlled by four different frame types:

- data frame carries data from a transmitter to the receivers.
- remote frame is transmitted by a bus unit to request the transmission of the data frame with the same identifier.
- error frame is transmitted by any unit on detecting a bus error.
- overload frame is used to provide for an extra delay between the preceding and the succeeding data or remote frames.

Data frames and remote frames are separated from preceding frames by an interframe space (Figure 2).

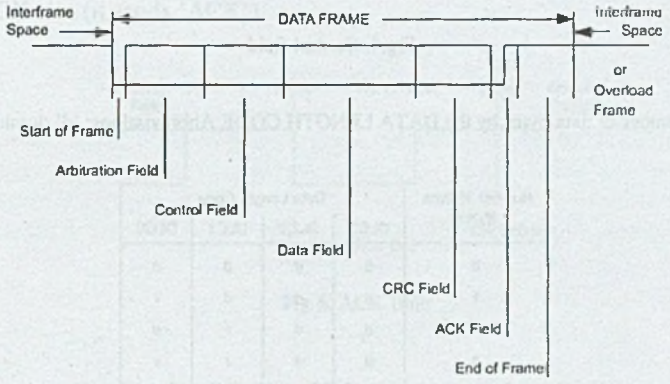

Fig.2. Data frame diagram

Start of frame marks the beginning of data frames and remote frames. It consists of a single 'dominant' bit.

The arbitration field consists of the identifier and the RTR-bit (Figure 3).

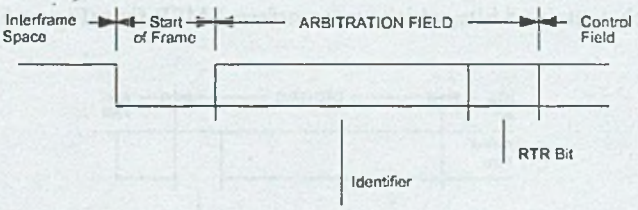

Fig.3. Arbitration field

The identifier's length is 11 bits. These bits are transmitted in the order from ID-10 to ID-0. The 7 most significant bits (ID-10 – ID-4) must not be 'recessive'. RTR bit (Remote Transmission Request Bit) in data frame has to be 'dominant'. Within a remote frame the RTR bit has to be 'recessive'. Control field includes the data length code and two bits for future expansions. The number of bytes in the data field is indicated by the data length code (Figure 4).

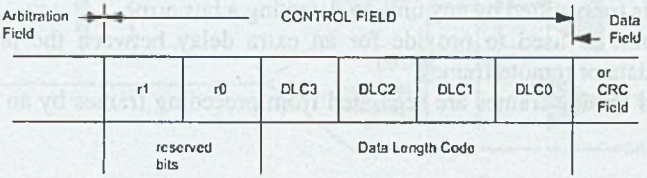

Fig.4. Control field

Table 1

Coding of the number of data bytes by the DATA LENGTH CODE Abbreviations: 'd' dominant, 'r' recessive

Number of Data Bytes	Data Length Code			
	DLC3	DLC2	DLC1	DLC0
0	d	d	d	d
1	d	d	d	r
2	d	d	r	d
3	d	d	r	r
4	d	r	d	d
5	d	r	d	r
6	d	r	r	d
7	d	r	r	r
8	r	d	d	d

Data field consist of the data to be transferred within a data frame. It can contain from 0 to 8 bytes, which each contains 8 bits, which are, transferred MSB first (Figure 5).

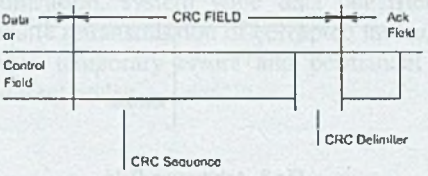

Fig.5. CRC field

CRC contains the CRC sequence followed by a CRC delimiter. In order to carry out the CRC calculation the polynomial to be divided id defined as the polynomial, the coefficient of which are given by the destuffed bit stream consisting of start of frame, arbitration field, control field (if present) and, for the 15 lowest coefficients, by 0. This polynomial is divided by the generator-polynomial:

$$X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1.$$

The reminder of this polynomial division is the CRC sequence transmitted over the bus. CRC delimiter always consists of a single 'recessive' bit.
ACK field is two bits long and contains the ACK slot and the ACK delimiter. In the ACK field the transmitting station sends two 'recessive" bits (Figure 6). A receiver that has received a valid message correctly, report this to the transmitter by sending a 'dominant' bit during the ACK slot (it sends 'ACK').

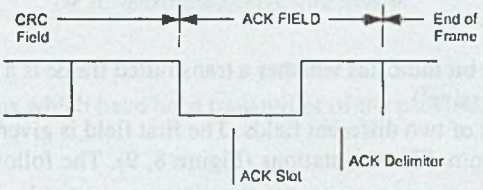

Fig.6. ACK field

All stations having received the matching CRC sequence report this within the ACK SLOT by rewriting the "recessive' bit of the transmitter by a 'dominant' bit. The ACK delimiter is the second bit of the ACK field and has to be a 'recessive' bit. Each data frame and remote frame is delimited by a flag sequence consisting of seven 'recessive' bits.
A station acting as a receiver for certain data can initiate the transmission if the respective data by its source node send a remote frame.

Contrary to data frames, the RTR bit remote frames is 'recessive'. There is no data field (Figure 7).

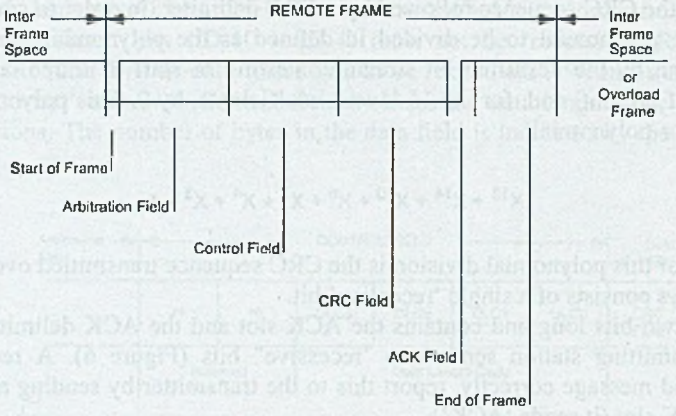

Fig.7. Remote frame

The polarity of the RTR bit indicates whether a transmitted frame is a data frame ('dominant') or a remote frame ('recessive').
The error frame consists of two different fields. The first field is given by the superposition of error flag contributed from different stations (Figure 8, 9). The following second field is the error delimiter.

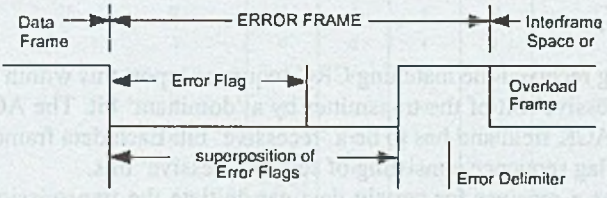

Fig.8. Overload frame with error delimiter

The overload frame contains two bit fields overload flag and overload delimiter.

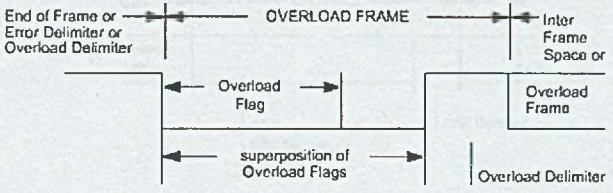

Fig.9. Overload frame with flag and delimiter

Data frames and remote frames are separated from preceding frames whatever type they are by a field called interframe space. In contrast, overload frames and error frames are not preceded by an interframe space and multiple overload frames are not separated by an interframe space.
Interframe space contains the bits fields' intermission and bus idle and, for 'error passive' stations, which have been transmitter of the previous message, suspend transmission (Figure 10, 11).
For stations which are not 'error passive' or have been receiver of the previous message:

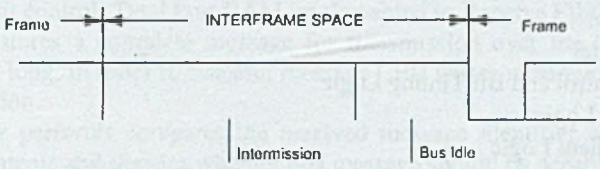

Fig.10. Interframe spacing – for receiver

For 'error passive' stations which have been transmitter of the previous message:

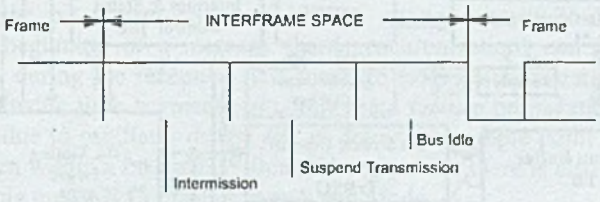

Fig.11. Interframe spacing – for transmitter

5. IP CORE CAN CONTROLLER

The CAN is a stand-alone controller for a Controller Area Network conforming to the CAN 2.0B specification. It provides an interface between a microprocessor and a CAN bus. The controller implements the BOSCH CAN 2.0B Data Link Layer Protocol. The CAN is compatible with a Philips SJA1000 working in its PeliCAN mode.
The project was preformed to take consideration the more important features as:

1. Supports Standard Frame Format (11-bit identifier) and Extended Frame Format (29-bit identifier);
2. Error Confinement functions, to identify error code, number of error appearance, and determine error warning limit.
3. Self Test Mode with Self Reception Request, this feature allows to check controller by self, without any other controller or tool.

4. Compound programmable Acceptance Filter to assure that's data on buss will be received by only addressee device.
5. Data Overrun control to protect overwriting data buffer
6. Sleep Mode, in this work mode is reduced power consumption.

The block diagram of the CANBUS controller is presented in Figure 12. The main elements of this controller are as follows:

- CPU Interface Control
- MUX and Decoder
- FIFO Control Unit
- Transmit Buffer
- Acceptance Filter
- Bit Stream Control and Bit Timing Logic
- CAN Interface Logic
- Error Management Logic
- Interrupt & Status Control unit.

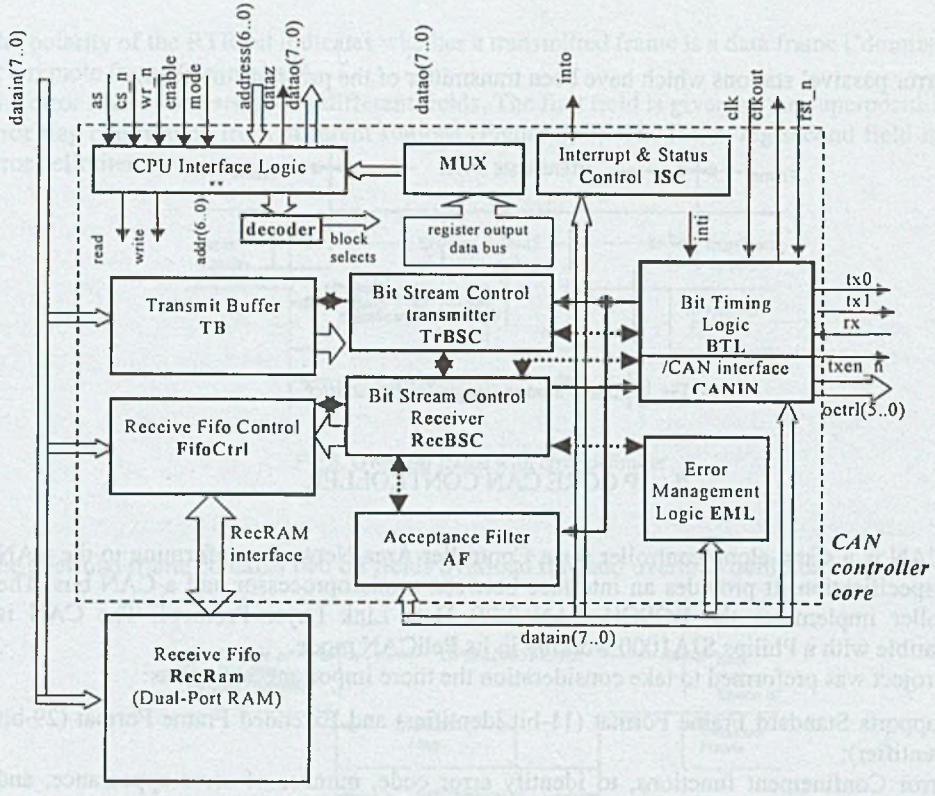

Fig.12. CAN Controller Block Diagram

CPU Interface Control controls write and read access to CAN controller registers from CPU. Functionality depends on selected CPU interface mode.
MUX controls addressing of the CAN registers within CAN blocks. It is entirely combinational circuit. It multiplexes data buses from CAN blocks into output data bus.
Decoder works out register select signals from latched address.
Receive FIFO stores received and accepted messages from CAN-bus. The Receive FIFO has a total length of 64 bytes. With the help of this FIFO the CPU is able to process one message while other messages are being received. As Receive FIFO there is implemented Dual Port RAM.
FIFO Control Unit controls Dual Port RAM implemented as Receive FIFO.
Transmit Buffer stores a complete message for transmission over the CAN network. The buffer is 13 bytes long. In order to transmit message CPU writes message into this buffer and starting transmission.
Acceptance Filter performs compares the received message identifier with the acceptance filter registers contents and decides whether this message should be accepted and written into Receive FIFO or not.
Bit Stream Control - Receiver monitors CAN bus line and controls message reception. This block is responsible for performing BOSCH CAN 2.0B protocol. It performs bus arbitration, CRC calculation and CRC checking, error detection, error signaling and bit destuffing.
Bit Stream Control – Transmitter - its main function is message transmission. It also forms frames, sends acknowledgements, Error and Overload Flags, and performs bit-stuffing.
Bit Timing Logic monitors CAN-bus line and handles the bus line-related bit timing. It is synchronized to the bit stream on the CAN-bus on a 'recessive-to-dominant' bus line transition at the beginning of a message (hard synchronization) and re-synchronized on further transitions during the reception of a message (soft synchronization). The BTL also provides programmable time segments to compensate for the propagation delay times and phase shifts (e.g. due to oscillator drifts) and to define the sample point and the number of samples to be taken within a bit time. Within this component there is also implemented logic that controls waking up CAN Controller from Sleep Mode.
CAN Interface Logic is an interface between CAN-bus line transceiver and CAN controller.
Error Management Logic is responsible for the error confinement of the transfer-layer modules. It receives error announcements from Receiver and then informs the ISC about error statistics.
Interrupt & Status Control Unit is responsible for interrupt generation, establishing requested CAN controller status and interpreting commands in Command Register. This block controls all other CAN Controller blocks.

6. CONCLUSIONS

Despite the vast literature found about the Controller Area Network, a lot of detailed problems (rarely discussed in the papers) had to be solved to achieve a viable implementation of the IP controller.
CAN has the following properties: prioritization of messages, guarantee of latency time, configuration flexibility, multicast reception with time synchronization, system wide data consistency, multimaster, error detection and signaling, automatic retransmission of corrupted messages as soon as the bus is idle again, distinction between temporary errors and permanent

failures of nodes and autonomous switching off of defect nodes.
The Controller Area Network module is implemented as separate virtual component (soft macro). Several versions could be available for different applications. This macro may be used as peripheral unit for any microcontroller with dedicated software implementation.

BIBLOGRAPHY

[1] HASE J., Virtual Components – from Research to Business, Proceedings of FDL'99 Conference, Lyon 1999
[2] BANDZAREWICZ M., SAKOWSKI W., Development of the configurable microcontroller core, Proceedings of FDL'99 Conference, Lyon 1999
[3] STUART M., DEMPSTER D., Verification Methodology Manual, Teamwork International, Hampshire UK, 2000
[4] CAN specification, Version 2.0, Bosh Documentation, Stuttgart 1991

Reviewer: Ph. D. Jerzy Mikulski