CCP

A CAN Protocol for Calibration and Measurement Data Acquisition

Rainer Zaiser
Vector Informatik GmbH
Friolzheimer Strasse 6
70499 Stuttgart,Germany

1 Contents

2 Introduction

The CCP (CAN Calibration Protocol) is a CAN (Controller Area Network) based application protocol for calibration and measurement data acquisition of electronic units. It has been developed by the European ASAP task force. The actual version 2.0 was released in June 1996.

2.1 The ASAP Task Force

The ASAP task force (**A**rbeitskreis zur **S**tandardisierung von **Ap**plikationssystemen; English translation: Task Force for the Standardization of Calibration Systems) has been founded by the German companies Audi AG, BMW AG, Mercedes-Benz AG, Porsche AG and Volkswagen AG. European manufacturers of automation, test and development systems for the automotive industry, as well as manufacturers of electronic control units have joined this task force.
Up to now, tools and systems used in automotive electronics development and production are not compatible and exchangeable. Similar technical components and data are involved in the development of hardware and software, in the calibration and test cycle, in manufacturing and last but not least, in service. The goal of the standardization done in ASAP is the exchange of data and the compatibility of hardware and software systems at all levels. A concept was developed which comprises stand alone sub systems, that can be linked by means of defined interfaces.

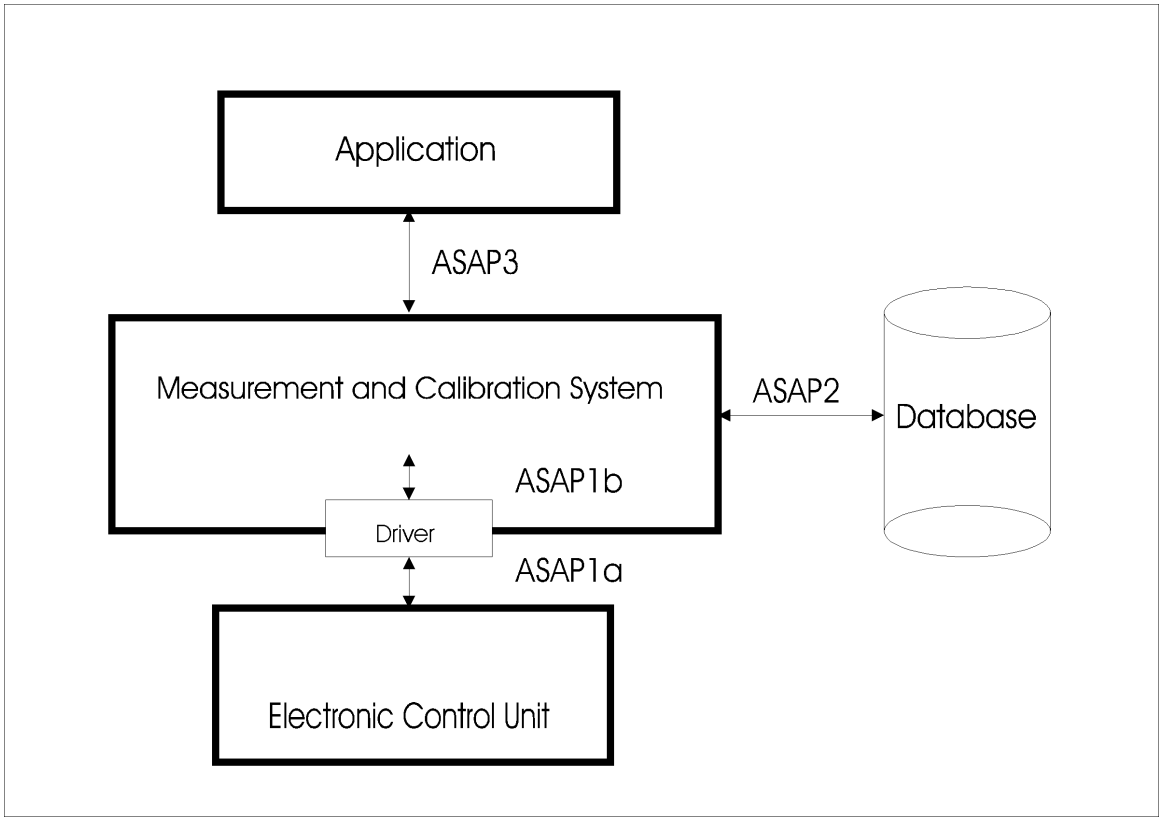


Fig.1: ASAP Interface Definitions

The model shows the interfaces ASAP1, ASAP2 and ASAP3. An Electronic Control Unit (ECU) is connected to a Measurement and Calibration System (MCS) via Interface

ASAP1. The required description database is provided by interface ASAP2. The interface ASAP3 connects the MCS to an automation system (for example a motor test stand). The ASAP1 interface definition is split into ASAP1a, the physical and logical interface to ECUs connected to the MCS and ASAP1b, the software driver interface into the MCS.

2.2 The CAN Calibration Protocol (CCP)

The CCP (CAN Calibration Protocol) is an ASAP1a interface using CAN 2.0B (11-bit or 29-bit identifiers) for communication between the MCS and the ECU.
The Controller Area Network CAN is a joint development of Robert Bosch GmbH and Intel Corporation. CAN has become the standard field bus in most automotive European and some North American powertrain control systems, as well as in body electronic systems. CAN is used in many industrial control systems. Controllers for CAN are available from various semiconductor manufacturers as standalone chips or integrated into many popular microcontroller chips.

CCP offers the following functionality:

- Read and write memory access to arbitrary locations in the ECUs RAM and ROM.
- Synchronous cyclic or event driven Data Acquisition.
- Simultaneous Calibration and Measurement Data Acquisition.
- Simultaneous Handling of Multiple ECUs.
- Flash Programming.
- Plug and Play.
- Calibration RAM Handling.
- Access Protection Security.

CCP has been designed to meet the requirements and restrictions of electronic control units. It is highly scaleable to fit into small 8 bit control units on low speed body electronic buses as well as providing high speed data acquisition in high performance powertrain control units.
No additional hardware has to be connected to the control unit. The CCP driver is just a small piece of software in the control unit. A basic implementation of CCP needs only a few control unit resources such as RAM, ROM and CPU time. The CCP driver in the control unit is simple and easy to implement. CCP occupies only two CAN identifiers, which may be assigned low bus priority to avoid influencing other bus traffic.
PC based measurement and calibration systems may use the same simple and cheap CAN interface, provided by various manufacturers, for CAN bus monitoring, measurement and calibration.

3 Protocol Description

The CAN Communication Protocol for Calibration and Data Acquisition is based on a master-slave type communication. A CCP master device is connected with one or more CCP slave devices on the CAN:

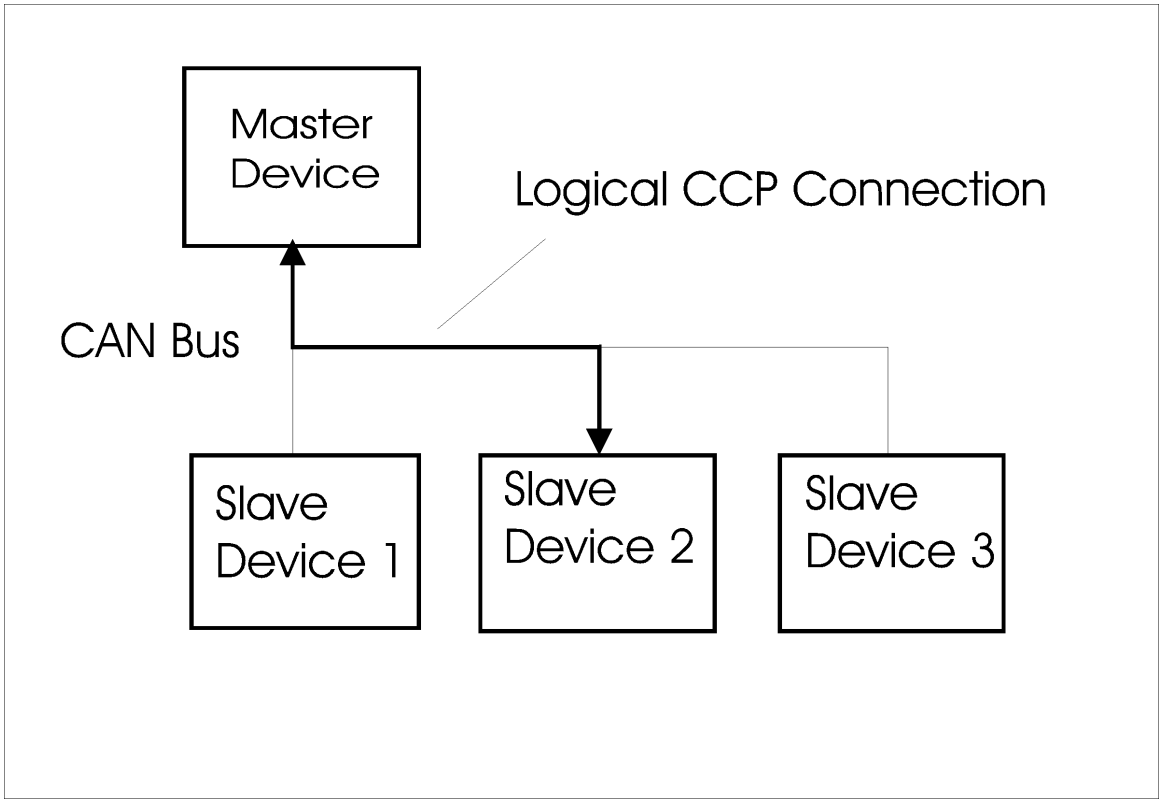


Fig.2: Master/Slave Device Configuration

The master device may be a MCS initiating the data transfer on the CAN by sending commands to the slave devices. The CCP implementation supports commands for generic control which includes primitive memory transfer functions and commands for synchronous data acquisition. These two parts (function sets) of the communication protocol are independent and may run asynchronously, depending on the implementation in the slave controller. It is also possible that messages of both parts are transmitted in nested order.

3.1 Generic Control Commands

These commands are used to carry out various functions with and within the slave device. For this purpose a continuous logical connection is established between the master device and any slave device. This logical connection is valid until another station is selected or the current station is explicitly disconnected via command.
After the logical connection has been made, data transfers from the master device to the slave device and from the slave device to the master device are controlled by the master device. All commands from the master device have to be prompted by the selected slave device with a return message, which contains the command return values or an error code.

3.2 Data Acquisition Commands

These protocol commands are used for continuous synchroneous data acquisition from a slave device. Any slave device may periodically transmit internal data, corresponding to a list that has been configured by control commands from the master device. Data transmission is initiated by the master device and executed by the slave device and may be dependent on a fixed sampling rate or may be driven by events (ex. crank position).

3.3 Organization of CAN Message Objects

According to the definition of the CAN, all messages and data to be transferred are packed into CAN message objects, containing up to 8 bytes of data.

The CCP requires two CAN message objects, one for each direction:

1. **C**ommand **R**eceive **O**bject (Master->Slave): **CRO**
2. **D**ata **T**ransmission **O**bject (Slave->Master): **DTO**

The **CRO** is used for the reception of command codes and their related parameters to carry out internal functions or memory transfers between the master device and the logically connected slave device. The reception of a command has to be prompted with a response message using the DTO, and in this case a DTO is called a **C**ommand **R**eturn **M**essage (CRM). The return code in this CRM is used to determine whether the corresponding command has been successfully completed or not.

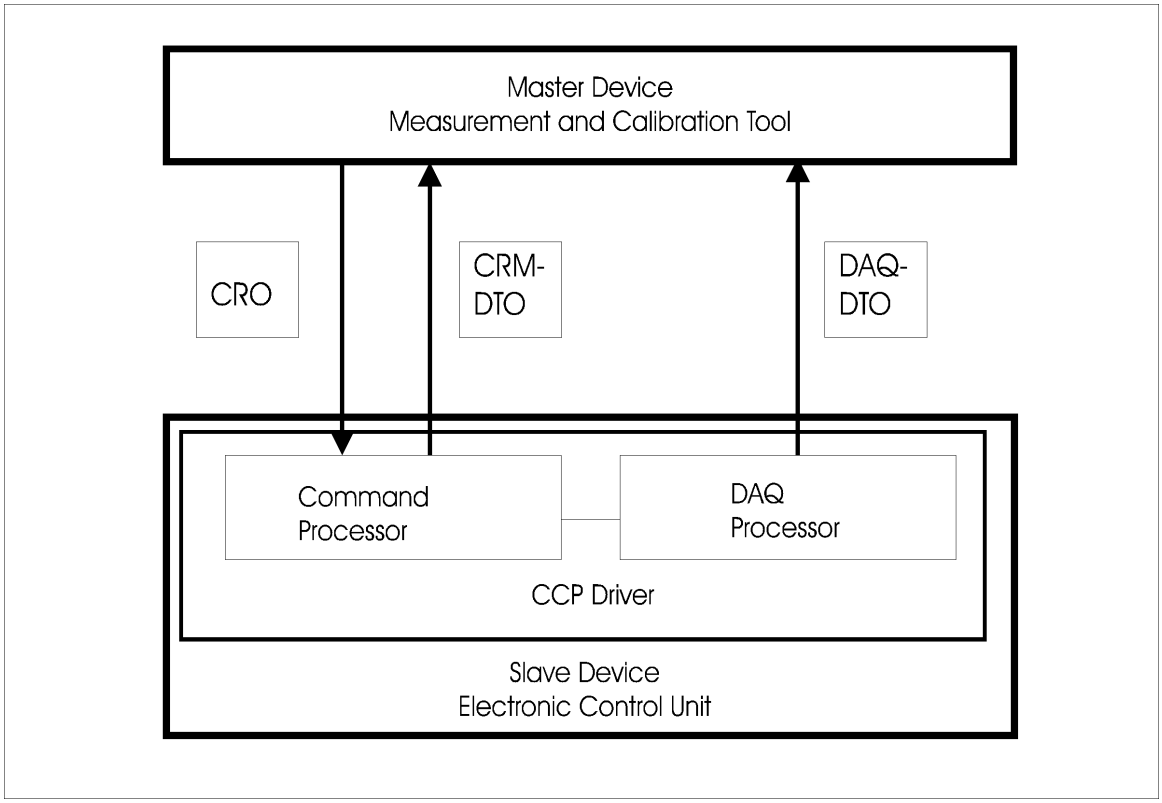


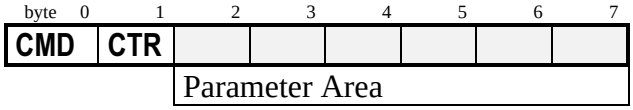
Fig. 3: Communication flow between CCP master and slave devices.

3.4 Description of Message Objects

3.4.1 Command Receive Object (CRO)

A Command Receive Object CRO is sent from the master device to one of the slave devices. It carries a command and the appropriate parameters to be executed in the slave device.

Structure of the CRO:



CMD: Command Code.
CTR: Command Counter.

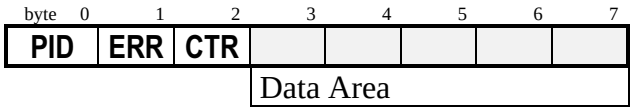
3.4.2 Data Transmission Object (DTO)

The Data Transmission Object DTO is sent from the slave device to the master device. It carries all outgoing slave device command responses or measurement data. The first byte of a DTO is used as the Packet Id PID to distinguish between different types of DTOs.

The DTO is a:

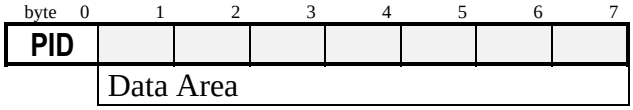
1. **Command Return Message** CRM, if the DTO is sent as an answer to a CRO from the master device.
2. **Event Message**, if the DTO reports internal slave status changes in order to invoke error recovery or other services.
3. **Data Acquisition Message**, if the DTO contains measurement data.

Structure of the Command Return- or Event Messages:



PID: PID=255: Command Return Message.
PID=254: Event Message.
ERR: Error code.
CTR: Command counter as received in CRO with the last command.

Structure of the Data Acquisition Message:



PID: Packet Id

PID=n: DTO contains data corresponding to an Object Descriptor Table.

3.5 Synchronous Data Acquisition

The master device can initiate a slave device to send event driven or periodically sampled measurement data in Data Acquisition Messages. Data Acquisition Messages are configured by setting up an Object Descriptor Table ODT in the ECU. The ODT contains up to 7 memory addresses, which point to measurement values in the ECU memory. Each ODT is assigned a unique Packet Id PID to identify the corresponding Data Acquisition Message (DAQ-DTO).

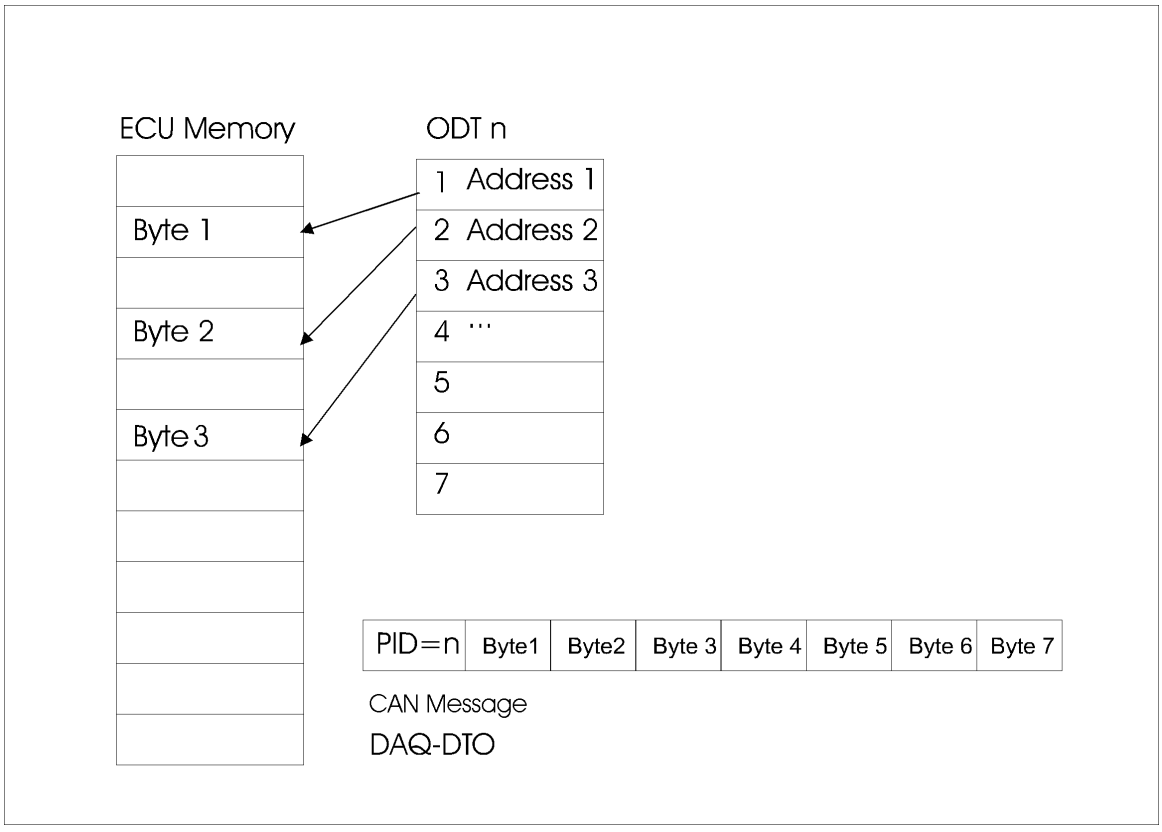


Fig.4: Object Descriptor Table

The contents of each element defined in a ODT are transferred into a DAQ-DTO to be sent to the master device. Multiple ODTs form a DAQ list.

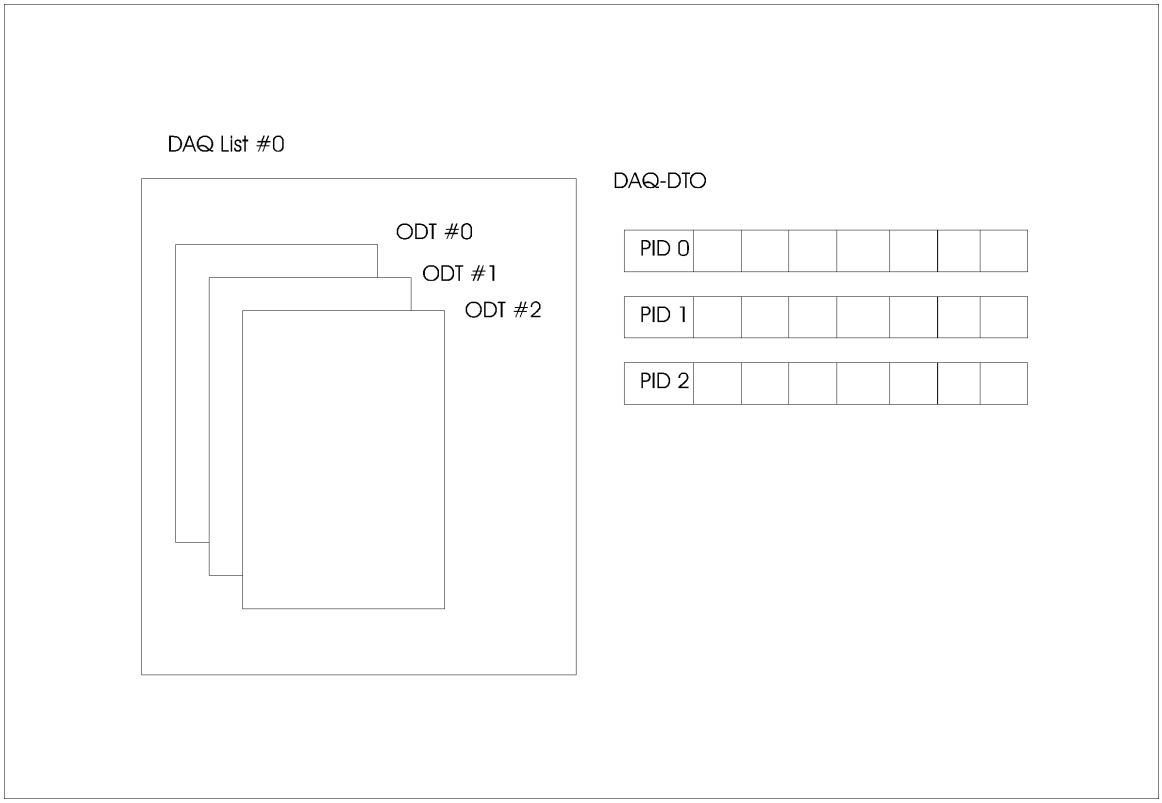


Fig.5: DAQ List with 3 ODTs

CCP allows the setup of a number of DAQ lists, which may be simultaneously active. The sampling and transmission of the DTOs of each DAQ list is triggered by individual events in the ECU. When a DAQ list is triggered, the data for all or one ODT (depending on the ECU implementation) is sampled in a consistent way and send over the bus.

3.6 Command Overview

The following table shows an overview of all CCP command:

Command	Description
CONNECT	Establish a logical connection to a slave device.
EXCHANGE_ID	Get the ECU identification.
TEST	Test for presence of a slave device.
SET_MTA	Set memory transfer address (MTA).
DNLOAD	Download up to 5 bytes into the ECU memory.
DNLOAD_6	Download 6 bytes into the ECU memory.
UPLOAD	Upload up to 5 bytes from ECU memory.
SHORT_UP	Upload up to 5 bytes from ECU memory (no MTA).
GET_DAQ_SIZE	Get the size of a DAQ list.
SET_DAQ_PTR	Set the ODT entry pointer.
WRITE_DAQ	Write an entry in a ODT.
START_STOP_ALL	Start or stop all DAQ lists.
START_STOP	Start or stop a specific DAQ list.
DISCONNECT	Finish a logical connection.
SET_S_STATUS	Get the ECU session status (optional).
GET_S_STATUS	Set the ECU session status (optional).
BUILD_CHKSUM	Calculate a memory checksum (optional).
CLEAR_MEMORY	Clear a memory range (optional).
PROGRAM	Download up to 5 program bytes (optional).
PROGRAM_6	Download 6 program bytes (optional).
MOVE	Move a memory range (optional).
GET_ACTIVE_CAL_PAGE	Get the active calibration page (optional).
SELECT_CAL_PAGE	Select the active calibration page (optional).
UNLOCK	Unlock the access protection (optional).
GET_SEED	Get the access protection code (optional).

Many of the commands refer to the special CCP features:

- Synchronous Data Acquisition
- Flash Programming
- Access Protection
- Calibration Page Switching

These features and their commands are optional.

4 Multi Slave Applications

Simultaneous Calibration and Measurement Data Acqusisition from multiple slave devices becomes more and more important in automotive systems as functionality will be distributed into multiple ECUs. Instrumentation input devices for analog and digital data may be treated as CCP slave nodes.
Multi slave applications require to carry out commands in more than one slave device simultaneously, for example starting the measurement data acquisition. Although the CCP has been designed to work point-to-point with only one slave device connected at a time, there is an approach for broadcasting protocol commands to more than one slave station:
All slave devices use the same message identifier for the CRO, and different message identifiers for the DTO. All concerned slave devices recognize a common broadcast slave address (in addition to their dedicated slave address), so that the CCP master device may multi connect to those slaves.

5 Plug and Play

5.1 Automatic Slave Device Detection

A calibration tool may automatically determine which slave devices are present in the system.
The CCP command. TEST polls for the presence of a specific device without changing the state of this device. The calibration tool scans a defined range of slave device addresses for reponses to detect all active control unit and instrumentation devices. A slave device node may transmit an event message on startup to indicate configuration and status changes.

5.2 Automatic Database Selection

Accessing the ECUs measurement and calibration objects is done by specifying their address in the ECUs memory. A measurement and calibration tool may use an ASAP2 database containing the descriptions of all objects including their address, datatype and conversion formula. Each version of the ECUs software will be described by a different database. To avoid assigning the correct database manually, CCP provides a command (EXCHANGE_ID) which identifies the ECUs software version. A measurement and calibration tool may use this identification to select the database automatically.

6 Resource Consumption

The ECUs CCP driver will consume resources such as RAM, ROM and CPU time. The code size of the CCP driver depends on the implemented optional features.
The following table shows values for a typical CCP driver running on a 16 bit microcontroller at 16 Mhz (Siemens C167). The driver was compiled from ANSI-C source code.

Resource	
RAM	1 DAQ list with 3 ODTs (for 21 measurement data bytes per cycle) **60 Bytes**
ROM	Without Flash Programming **2 Kbyte**
CPU time	Command Processor (per Command): 100-200 usec depending on the command DAQ Processor (per ODT): **50-100 usec**

The code size may be significantly reduced by implementing the driver in assembler language.

The amount of ECU RAM needed in an implementation of the CCP depends on the number of DAQ lists and their ODTs supported by the CCP driver.
The number of ODTs in a DAQ list determines the maximum number of data acquisition bytes in one measurement cycle. Each ODT specifies 7 measurement data bytes, as described in chapter 3.5.

Bytes per Measurement Cycle = NUM_ODT * 7

The number of DAQ lists determines the number of simultaneously possible data acquisition cycle rasters or events. Each DAQ list will need additional RAM space for some runtime informations.

The amount of RAM needed may be estimated by the following formula:

Memory Bytes need = NUM_ODT*14 + NUM_DAQ*4

The number of DAQ lists and their ODTs should be carefully chosen by the ECU developer according to the demands on cyclic data acquisition.

7 Performance Ratings

The performance ratings of the CCP are strongly depending on the response latency times, the bus baudrate, the bus load situation and the bus priority level of the message objects.
As only small portions of memory may be transferred at a time, the latency time of the involved network nodes is the most restrictive factor regarding burst memory transfers (Download or Upload).

With a baudrate of 500 kbit/s and a typical bus load of 20% the following ratings have been achieved:

Type	Data rate (approx.)
Burst Memory Transfer	4-6 Kbyte/s
Data Acquisition	20 Kbyte/s

Due to the nature of the CAN, the latency time for transmission of a specific message may vary strongly, depending on the message priority and the bus load situation. In many implementations of the CCP, the message objects are assigned to low bus priorities to avoid an impact on the overall system behavior. In these cases, the CCP driver may run into a situation, where a new acquisition cycle has started but the transmission of messages from the previous cycle is not yet completed. To avoid skipping the actual cycle, the CCP driver will need additional RAM space to buffer the messages in a FIFO.

The CCP driver configuration for a certain ECU implementation should be investigated by tests using the system on real bus situation to find an appropriate compromise on the following properties:

- Maximum Number of Bytes per Acquisition Cycle
- Message Object Priority
- FIFO Size
- Cycle Loss Probability

8 References

1. CCP CAN Calibration Protocol
   ASAP Standard
   Version 2.0
   14.June.1996
2. CCP Driver Implementation
   Version 1.0
   Vector Informatik GmbH, 1998
3. ISO/DIS 11898
   Road vehicles - Interchange of digital information
   Controller Area Network (CAN) for high speed communication
   August 1992
4. ISO/DIS 11519-1
   Road Vehicles - Interchange of digital Information
   Low speed serial data communication,
   Part 1: Low speed Controller Area Network (CAN)