ijecs open access www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 6 June, 2013 Page No. 2072-2085

Development & Calibration of CAN for Designing Vehicle Electronic Architecture

Rahul B. Adsul[1], Prof. Mrs. S.D. Joshi[2]

[1]AM, CIS Technologies (India) Pvt. Ltd., Pune, Maharashtra, India
rahulbadsul@gmail.com

[2]Professor, Dept. of Electronics & Telecommunication, PVPIT,
University of Pune, Maharashtra, India.

s.d.joshi@hotmail.com

Abstract: ***This literature is in the field of communication networks where different Electronic Control Units (ECUs) communicate with each other over Controller Area Network (CAN) protocol. Typically these types of CAN networks are used in automotive vehicles, plant automations, etc. This proposed method is applicable in all such applications where controller area network is used as backbone electrical architecture.***
This literature proposes a new method of CAN signal packing into CAN frames so that network bus-load is minimized so that more number of CAN signals can be packed and more number of ECUs can be accommodated within a CAN network. The proposed method also ensures that the age of each CAN signal is minimized and all CAN signals reach the intended receiving ECUs within their maximum allowed age. Typically network designers are forced to design and develop multiple sub-networks and network gateways to get rid of network bus-load. As the proposed method intends to minimize network bus-load, the requirement of gateways just to reduce bus-load will be avoided.
The implementation of CAN messages has been a critical aspect of the ECU development process in recent years. The traditional approach generates more inconsistencies between the specification and the software coding (implementation), probably coding errors, and variable reproducibility of the implementation, depending on the ECU platform. Therefore in order to achieve an efficient development, the use of a high abstraction level of the CAN protocol is essential.
The proposed method also proposes method for assignment of CAN Identifiers in each CAN frame by introducing different sub-fields within the CAN Identifiers. This will help to reduce the delay in the CAN frames due to arbitration loss in the network. The proposed methods will minimize the ECU loads network due to CAN frame reception for all ECUs in the CAN by minimizing the number of CAN frames to be received by each ECU in the network and also by enabling hardware filtering of CAN frames by receiving ECUs due to different sub-fields within the CAN Identifier.

Keywords: Electronic Control Unit, Automotive Domain, CAN Identifiers, Frames.

Introduction

This literature relates to communication networks where different electronic control units (ECUs) communicate with each other over Controller Area Network (CAN) protocol. The present paper talks about a new design method for CAN signal packing into CAN frames, assigning CAN Identifiers into CAN frames so that network bus-load and ECU load can be minimized.
Typically these types of CAN networks are widely used in automotive vehicles, plant automations, etc. The proposed method is applicable in all such applications where controller area network is used as backbone electrical architecture for sharing electrical signals. With the advent of distributed real-time control functionalities the need has arisen for reliable communication network systems where different Electronic Control Units (ECUs) are connected with each other through Controller Area Network (CAN network) buses for sharing hundreds of critical signals. These CAN signals are packed into CAN frames. These CAN frames in turn are transmitted and received by different ECUs in the network. Typical applications of such network systems are widely seen in automotive domain, where in-vehicle networks are used as the backbone of all distributed control functions.
Existing methodologies for CAN signal packing mainly group the signals based on the periodicity of these signals; so that the signals with same periodicity and transmitted by same ECU are grouped together. Then signals which are in the same group are packed into a CAN frame until the frame is fully packed. If the first CAN frame is fully packed, then

another CAN frame is created and packed with remaining signals and so on. After all of the CAN frames are created, lower value of "Frame Identifier" is assigned to CAN frames with lower periodicity (or more frequent CAN frames) to allocate higher priority for that frame.
The problem associated with existing practice of CAN signal packing is that the bus-load of the network and ECU load are not optimized. So, the increase in number of CAN signals results in increase in network bus-load and ECU load. This results in unnecessary increase in sub-networks and gateways in the network architecture which in turn increases the cost.

The CAN Calibration Protocol, which is commonly referred to as CCP, is essentially a software interface used to interconnect a development tool with an Electronic Control Unit (ECU). The interface defines methods to handle module calibration, measurement data acquisition, and flash programming activities.
Whether the complete CCP interface or a portion of it is supported by the tool or implemented in the ECU depends upon the module developer's needs. Based on the Controller Area Network (CAN) protocol, CCP places no limitations on the choice of physical layer and the system-selected communication bit rate.
CAN Calibration Protocol is capable of supporting both a single point-to-point connection and a networked connection to an entire distributed system. This means that any combination of calibration, measurement data acquisition, and flash programming activities are possible for a single module or for any portion of modules across a CAN network.

There is yet another problem associated with existing method. As the signal packing is not optimized, the bus-load is increased. This increase in bus-load results in higher latency for the CAN signals. Higher latency value for CAN signals affects the performance of the distributed functions which leads to customer dissatisfaction. Existing methods for assignment of CAN Identifiers into different CAN frames are not optimized. It classifies the available range of CAN Identifiers based on the periodicity of CAN frames and then randomly assigns the CAN Identifiers based on the periodicity of CAN frames. These methods leads to arbitration loss of CAN frames in CAN network and thus leads to message loss.
In recent years, the Commercial Vehicle OEM has been offering many technical solutions to satisfy customer needs and to increase the efficiency and productivity of trucks. The new solutions have been realized by using Electronic Control Units. All ECUs are linked together via a unique communication interface. The information which each ECU needs to function correctly is received from other system components and other ECUs. In addition, each ECU puts data about its own status on the network for reception and evaluation by other control units. For example, a *vehicle computer* can read the current engine speed from the *engine controller* and provides the relevant data to a different ECU. The OEM that elaborates the ECU network has the big task of requiring all suppliers to provide a common and shared standard protocol in order to improve the system's robustness and diagnostics.
The CAN standard is well established and has been refined as the *standard de facto* for communication between ECUs. The CAN network implementation typically includes a microprocessor with an integrated CAN protocol controller. Usually, each supplier provides a CAN Driver, based on OEM-specific requirements in terms of CAN Controller Initialization, Reception and Transmission of CAN messages, CAN Messages scheduling, CAN Gateway functions and more (in general functions relevant to Network Management).

Detailed Description:

Typically, different Electronic Control Units (ECUs) are connected with each other over CAN network. Different ECUs share their data with each other over this CAN network in form of CAN signals. For example, Engine Control Module (ECM) is connected in the CAN bus. ECM has the data for current engine speed. This data needs to be shared with other ECUs over CAN network. Then ECM needs to transmit current engine speed as one of the transmit signals over CAN bus and other ECUs need to receive this CAN signal.
Typically, ECUs transmit and receive CAN frames over the network. So, CAN signals need to be packed in CAN frames such that the ECUs can share the signals over CAN network. This literature proposes optimized method for packing these CAN signals into CAN frames.

Terminologies -
Following terminologies are defined to explain the proposed method.
Age of a signal:
Age of a signal is defined as the time between the following two events:
• The time instant when a new value of the signal is available with the application in transmitting ECU.
• The time instant when signal is consumed by the application in receiving ECU.

As shown in Fig. 1, the Age of a signal is the summation of the following:
• Time delay in the transmitting ECU after generation of new signal value and before queuing of CAN frame for transmission
• Time required to transmit the frame in the CAN network (considering time delay due to arbitration loss in the CAN network)
• Time delay in the receiving ECU after reception of CAN frame and before consumption of signal value by the application in receiving ECU

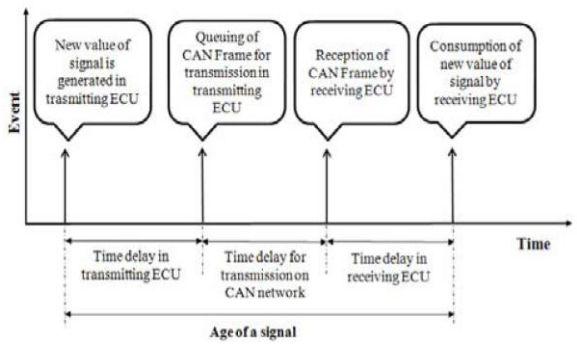


Figure 1: Age of a signal in CAN network

Max allowed Age of a signal:
It is the maximum allowed delay time by which the latest value of a signal generated in transmitting ECU can be consumed by the application in receiving ECU without any degradation in distributed functionality.
OBJECTIVES: The proposed method aims to pack the transmit signals into CAN frames such that the following two objectives are satisfied:
Objective 1: Network average bus-load is minimized.
Every transmit CAN frame contributes to the bus-load. Average bus-load contributed by one transmit CAN frame is

$$Average\ busload = \cdots$$

$$= \frac{(Total\ number\ of\ bits\ in\ one\ Frame) \times \ldots}{(Periodicity\ of\ CAN\ Frame)}$$

$$\times (Bit\ time) \times 100\% \quad \text{.....(1)}$$

Average bus-load contributed by every individual CAN Frame needs to be added for all the Frames in the network to arrive at the average bus-load of the network.
So, we can conclude that the average bus-load of a network is directly proportional to the number of CAN Frames in the network. More the number of Frames more is the bus-load. We can also say that lower the periodicity of a CAN Frame higher is the average bus-load of network.

Objective 2: ECU load due to CAN frame reception in the network is minimized.
ECU load due to CAN frame reception is directly proportional to the number of CAN frames which that ECU receives.
For example, if any ECU needs to receive 7 different CAN signals from the network then
- The ECU load due to CAN frame reception will be maximum if the ECU needs to receive 7 different CAN frames for 7 different CAN signals (i.e. all different signals are packed in different frames)
- The ECU load due to CAN frame reception will be minimum if the ECU needs to receive only 1 CAN frame (i.e. all 7 signals are packed within 1 CAN frame).

So, we can conclude that ECU load due to CAN frame reception is maximum if all the intended receive signals are packed in different CAN frames. To minimize ECU load number of receive CAN frames by any ECU in the network needs to be minimized. Also, the CAN Identifiers shall be assigned in the CAN frames in such a way that any receiving ECU is able to filter the intended receive CAN frame correctly through hardware filtering.

Proposed Method:

The proposed method is explained step by step to meet the objectives of this literature.

Step 1
Names of all CAN signals transmitted by different ECUs in the network need to be listed first. Each signal which is transmitted by any ECU in the network will be received by one or more ECUs in the network. If one signal is received by more than one ECU, then there is possibility that the functional requirement for the same signal may differ among different receiving ECUs. So, the following signal properties for each transmit signal need to be defined in the following manner:
• **Physical range of transmit signal**: This shall be defined as the maximum of all "physical range" requirements from all receiving ECUs for this signal.
• **Physical resolution per bit**: This shall be defined as the minimum of all "physical resolution per bit" requirements from all receiving ECUs for this signal.
• **Maximum allowed Age**: This shall be defined as the minimum of all "max allowed Age" requirements from all receiving ECUs for this signal.
So, if the same signal is needed by different receiving ECUs and functional requirements are different for these receiving ECUs, the transmit signal shall not be duplicated to meet separately different receiving ECU's requirements; the properties of transmit signal shall be chosen such that it meets requirements of all receiving ECUs to avoid duplication.

Step 2
The length and periodicity of each transmit signal needs to be derived as follows. This is applicable for all periodic transmit signals.

$$Length\ of\ CAN\ signal\ (in\ bit) = \cdots$$

$$= \frac{Physical\ range\ of\ signal\ including\ offset}{Physical\ resolution\ per\ bit} \quad \text{...(2)}$$

$$Periodicity\ of\ CAN\ signal = \cdots$$

$$\frac{Max\ allowed\ Age\ of\ the\ signal}{Factor\ of\ safety} \quad \text{.... (3)}$$

Factor of safety is a configurable value, by default it shall be 3. For critical signals, Factor of safety shall be 5. So, the proposed method suggests periodicity (or cycle time) of each transmit signal to be at most one third the max allowed age for that signal. This will ensure that each transmit signal will get consumed by receiving ECU within max allowed age of the signal.
As CAN is real-time in nature, the Transmit-Signal Model of any ECU may demand for Event-driven or Mixed (Eventdriven + Periodic) CAN signals to be transmitted by that ECU. For such cases of Event driven or Mixed CAN signals, Virtual Periodicity shall be calculated from the max allowed age of these signals, as mentioned in Eq. (3). This method of computing Virtual Periodicity will simplify the modeling of Event-driven and Mixed CAN signals in real-time distributed control systems.

Step 3
Assign priority for each transmit signal. Higher priority shall be given to critical signals. Typically 4 different levels of priority need to be assigned where priority 1 is the highest priority and priority 4 is the lowest priority. Following are the different priorities for CAN signals:
• **Priority 1:** All safety critical signals, security related signals, interface torque control signals and associated health status signals conveying plausibility or implausibility of different control signals shall be given the highest priority.

• **Priority 2:** All body function related control signals and associated health status signals shall be given next higher priority (i.e. priority level 2).
• **Priority 3:** All comfort and convenience function related signals and associated health status signals shall be given next higher priority (i.e. priority level 3).
• **Priority 4:** All other signals (for example information related signals, network management related signals, diagnostic signals, proprietary signals, development signals, etc) shall be assigned the lowest priority.

Step 4

Packing of CAN signals needs to be done separately for different priority groups of signals. First all CAN signals from highest priority group need to be packed in the CAN frames. Refer Fig. 2. This figure shows a 3 dimensional plot of which X axis represents the list of transmitting ECUs, Y axis represents the list of receiving ECUs and Z axis represents the list of signals of same priority. List of all transmitting ECUs shall be written in X axis and list of all receiving ECUs needs to be written in Y axis in any order. Then list of all signals from highest priority group needs to be written in Z axis in any order. Then all signals in Z axis needs to be mapped with the transmitting ECUs in X axis and the receiving ECUs in Y axis as per this figure. It may so happen that a given signal is received by more than one receiving ECUs. This mapping of each CAN signal with its transmitting ECU and receiving ECUs may be represented with the help of a symbol (say a Dot).

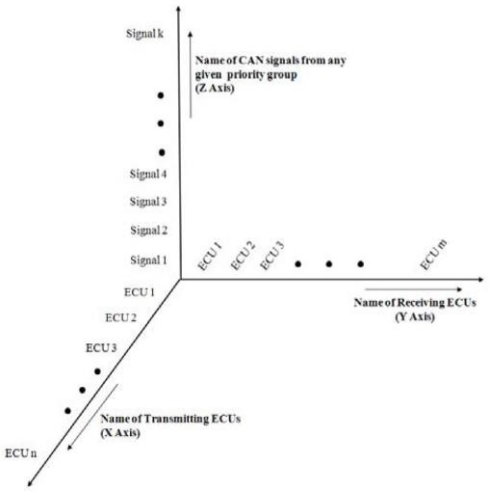


Figure 2: Mapping of CAN signals of same priority with Transmitting ECU and Receiving ECUs

Step 5

After each signal from highest priority group is mapped with transmitting ECU and receiving ECUs, all the Dots will be distributed in the 3 D space in the plot. The names of the signals in Z axis and the name of the receiving ECUs in Y axis were written in any random order in the previous step. Now, the order of the signals along the Z axis and the order of ECUs in Y axis shall be re-arranged in such a way that all the Dots in the 3 D space form different distinguishable clusters of Dots. Different Dots within any cluster are close together, however Dots in different clusters are much distant from each other; so that different clusters are distinguishable from each other.

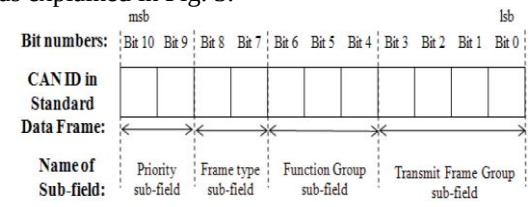
All signals from any cluster of Dots shall be packed together in the same CAN Frame. Signals shall be packed in a CAN frame until the frame is fully packed (or available data field is too small to accommodate any signal further). If the number of signals are more and if it is not be possible to accommodate all the signals within one CAN Frame, then the remaining signals need to be packed in another CAN Frame, and so on. All signals from different cluster of Dots shall be packed in different CAN Frames. At this stage, only different serial numbers need to be assigned to different CAN Frames to differentiate one CAN Frame from the other. Actual value of CAN Frame Identifier will be assigned for each CAN Frame at a later step as per proposed method.

Once all signals from the highest priority signal group are packed, then signals from second highest priority group shall be packed. So, step 4 and step 5 shall be repeated for all the signals in second highest priority group and so on for other priority groups. It is possible to have a partially packed CAN frame with higher priority level (say priority level 'x') CAN signals with sufficient data field available to pack any other signals within this frame. However, all the signals of priority level 'x' are already packed into CAN frames. Then signals from lower priority level (say priority level 'y'; where value of 'y' is greater than value of 'x') can be packed into this partially packed CAN frame with higher priority signals to optimize the bus load.

Step 6

After signals are packed in different CAN frames and different serial numbers are assigned in different CAN frames, it is recommended to assign the value of CAN Identifier for each CAN frame. As per CAN protocol specification (version 2.0), CAN Frames have either 11 bit CAN Identifier or 29 bit CAN Identifier. CAN frame with 11 bit Identifier is called Standard Frame and CAN frame with 29 bit Identifier is called Extended Frame. This literature suggests method for assigning CAN Identifier for Standard Frame (i.e. CAN frame with 11 bit CAN ID).

This paper suggests defining sub-fields within the available 11 bits for CAN ID. Following sub-fields shall be defined within 11 bit CAN ID before assigning the value of CAN ID as explained in Fig. 3.



lsb: Least Significant bit
msb: Most Significant bit

Figure 3: Sub-fields in CAN Identifier of a Standard CAN Data Frame with 11-bit CAN Identifier

Priority sub-field: This is a 2 bit sub-field within the 11 bit CAN ID in the highest significant part of CAN ID. As this is 2 bit sub-field, priority can take value range from 0 to 3.

Priority levels 1 to 4 was assigned for each transmit signal in step 3. Refer the definition of priority level of transmit signal in step 3. Now, value of this sub-field shall be according to the priority level of the signals within the CAN frame, if all the signals within the CAN frame are of same priority level. Else if all the signals within the CAN frame are not of same priority level then value of this sub-field shall be according to the highest priority signal within the CAN frame.

Table 1: Priority sub-field

Value of Priority sub-field		Condition / Description
Integer value	Binary value	
0	00	This state corresponds to the highest priority. At least one highest priority signal (example: safety critical signal, security related signal, etc with priority level 1) is packed in the CAN frame
1	01	This state corresponds to higher medium priority. At least one higher medium priority signal (example: body function related signal with priority level 2) is packed in the CAN frame
2	10	This state corresponds to lower medium priority. At least one lower medium priority signal (example: comfort and convenience function related signal with priority level 3) is packed in the CAN frame
3	11	This state corresponds to lowest priority. All signals in the CAN frame are lowest priority with priority level 4

Frame type sub-field: As shown in Fig. 3, the length of this sub-field is 2 bits and placed just adjacent to priority subfield. This sub-field can take value range from 0 to 3. Different sub-field values are assigned depending on different type and periodicity of frames. This is shown in the Table 2.

Table 2: Frame Type sub-field

Value of Frame type sub-field		Condition / Description
Integer value	Binary value	
0	00	This state shall not be used
1	01	Periodic CAN frame with cycle time (or periodicity) less than 30 ms
2	10	1. Periodic CAN frame with cycle time (or periodicity) within 30 ms to 250 ms. Or 2. Event driven CAN frame (cyclic on change with periodicity less than or equal to 100 ms)
3	11	1. Periodic CAN frame with cycle time (or periodicity) greater than 250 ms. Or 2. Event driven CAN frame (cyclic on change with periodicity greater than 100 ms) Or 3. Network Management frames Or 4. Diagnostic frames Or 5. Other frame types

Function group sub-field: As shown in Fig. 3, the length of this sub-field is 3 bits in the lower significant part of CAN ID. So, this sub-field can take value range from 0 to 7. All individual transmitting ECUs in the network shall be classified under different function group according to criticality of the function groups and shall be assigned function group ID in this sub-field of CAN Identifier for every transmitting CAN frame. Table 3 provides different group IDs for different functions as commonly used in automotive domain.

Value of Function Group sub-field		Name of function group
Integer value	Binary value	
0	000	Power-train domain
1	001	Chassis domain
2	010	Safety and security domain
3	011	Body domain
4	100	Infotainment domain
5	101	Comfort and Convenience domain
6	110	Luxury function domain
7	111	Other domains

Table 3: Function Group sub-field

Transmit frame group sub-field: As shown in Fig. 3, the length of this sub-field is 4 bits in the lowest significant part of CAN ID. So, this sub-field can take value range from 0 to 15. For a given priority, frame type and function group it is possible to have 16 different CAN frames with different CAN IDs. This sub-field shall act as a serial number field. Different serial number from this sub-field shall be assigned starting from integer value 0 (or binary value '0000') for different CAN frames under same priority, frame type and function group to generate different CAN Identifiers. This will result in 16 different CAN Identifiers for same value of priority sub-field, frame type sub-field and function group sub-field.

For example, if any receiving ECU wants to filter all CAN frames with safety critical signals with 20 ms periodicity from chassis domain function group then as per proposed method, the value of different sub-fields within CAN ID will be as specified in Table 4.

Table 4: Sub-fields in CAN ID

Name of sub-fields within CAN ID	Length of sub-field (in bits)	Value of sub-field	
		Integer value	Binary value
Priority sub-field	2	0	00
Frame type sub-field	2	1	01
Function group sub-field	3	1	001
Transmit frame group sub-field	4	Don't Care	XXXX

From the Table 4, we can derive the 11 bit CAN ID range as '0001001XXXX', where X represents Don't Care condition. So, in this example receiving ECU shall configure Acceptance Mask and Acceptance Code Registers as per Table 5.

Table 5: Acceptance Mask and Code Register

CAN ID range to be filtered (in binary)	Value of Acceptance Mask Register (in binary)	Value of Acceptance Code Register (in binary)
0001001XXXX	11111110000	00010010000

Step 7

The last step is to distribute all the CAN frames transmitted by different ECUs with time in the network in such a way that the peak bus load is minimized. If an ECU transmits more than one CAN frames, then the

ECU shall not transmit all the CAN frames in burst. This is because if any ECU transmits all of its different transmit CAN frames in burst, then the peak bus load will increase and also there will be a possibility that the receiving ECUs may be able to receive all the consecutive CAN frames in a bust resulting in loss of CAN frames. Hence this literature proposes the following.

Each ECU shall transmit the highest priority CAN frame first. The time instant when the first message is transmitted shall be considered as zero reference. With respect to this zero reference, other CAN frames shall be transmitted with decreasing priority allowing sufficient time-gap between two different priority frames (with different CAN ID) transmitted by the same ECU. This shall be achieved by specifying offset time for every CAN frame transmitted by any ECU with respect to zero reference.

Let us consider the following example. One ECU which shall transmit three CAN frames M1, M2 and M3. All the three frames are 20ms periodic. The highest priority CAN frame is M1 and the lowest priority CAN frame is M3 for the ECU under consideration. Then the instant when frame M1 is transmitted is the 'zero reference' for frames M2 and M3. If the offset time for M2 is defined as 5ms and offset time for M3 is defined as 10ms, then the ECU under consideration shall transmit the frames M1, M2 and M3 as shown in Fig. 4.

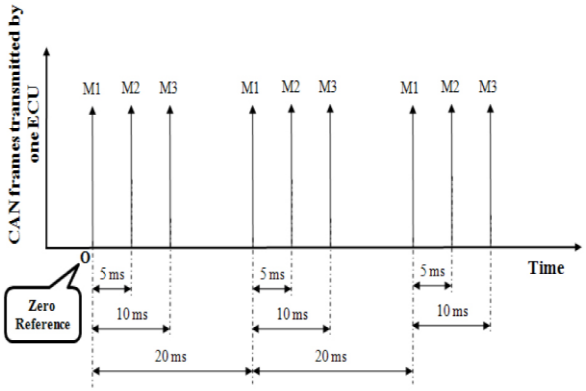


Figure 4: Offset Time between different CAN Frames (with different CAN IDs) to reduce peak bus load

So, offset time aims to specify the time between different CAN frames with different CAN IDs transmitted by the same ECU to avoid burst transmission of all different CAN frames by the same ECU.

CAN Calibration Protocol (CCP):

Basics -

In its most elemental form, the CAN Calibration Protocol is a monitor program. Similar to many earlier serial RS232-type monitors and bootstrap loaders that provide basic read and write memory capabilities, CCP provides the same functionality using a standard protocol rather than a company-specific proprietary protocol. However, when one is using a rather high-speed CAN bus in comparison to some previous 9600 baud UART-based monitor, CCP provides the ability to access data at such a fast rate that it is possible to run an application at the same time. Using the right tool, developers now have a significant advantage over the earlier monitor methods. When one examines the dialog used by most monitor programs, it is the tool or PC that is the master of the commands sent into the ECU. For CCP there is no difference. The ECU does nothing without the master (Tool) initiating commands. Using the appropriate CCP messages, a CCP-compliant tool can read data from the ECU and can write data into the ECU.

However, this is only CAN Calibration Protocol's minimum capability. CCP includes several additional monitor commands, and provides several new features including automatic data acquisition processing based on events or periodic updating, flash programming and data security. Because there is no requirement to use all its features, CCP is a scalable protocol.

CCP Communication:

Using only two CAN identifiers for message transfers, CCP uses a specific conversation or dialog to accomplish each designated function. Each dialog is a collection of exchanged messages between a master, the calibration or development tool, and a slave ECU. Most CCP dialog always uses a master/slave form of conversation. The tool (or master) always initiates the conversation with a single CAN message, and once received, the ECU (or slave) is then responsible for responding with a single CAN message.

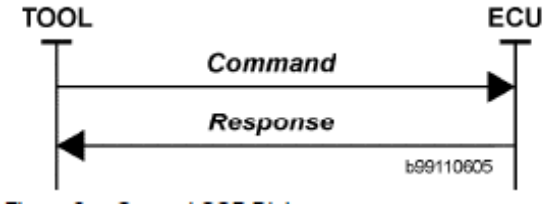


Figure 5: General CCP Dialog

Basic CCP Commands – One of the CAN Identifiers is used to send information from the tool to the ECU. This command is defined from the ECU's point of view as the Command Receive Object (CRO). The CRO contains command information and the related parameters needed for the command. After the ECU receives the CRO, the CCP driver processes the command code within the CRO. Then, internal functions or data transfer between the tool and the ECU can occur.

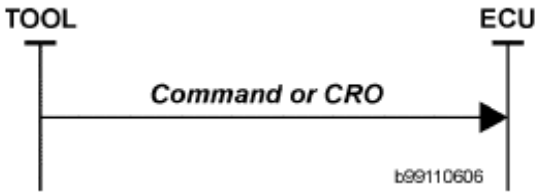


Figure 6: Command Receive Object

CCP Commands:

The following table shows an overview of all CCP commands used in the CRO:

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	Set the ECU session status (optional)
GET_S_STATUS	Get 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)

CRO Message – The data field for each CAN Identifier used with CCP is limited to 8 data bytes because CCP is based on CAN. The CRO uses each of the data bytes for specific items of information based on the command.

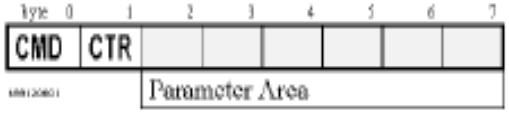


Figure 7: Structure of the CRO Message

The first byte of the CRO Message is a number that corresponds to the command as defined in the CCP specification. The second byte is a counter from the tool to track the current command that was issued. This same value will also be used in the response from the ECU to the tool.
Basic CCP Response – The other CAN Identifier is used to send information from the ECU to the tool. This command is defined from the ECU's point of view as the Data Transmission Object (DTO).

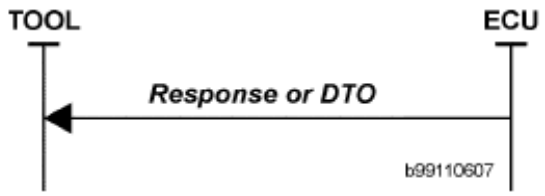


Figure 8: Data Transmission Object

Three types of DTOs are defined by the CCP specification. These three types are the Command Return Message (CRM), the Event Message, and the Data Acquisition Message (DAQ).
CRM-DTO Message – The Command Return Message (CRM) is a message sent by the ECU in response to a CRO. The CRM can be a simple acknowledgement to the CRO, or it can contain actual requested data. The CCP specification describes in detail the content of each CRM for each CRO.

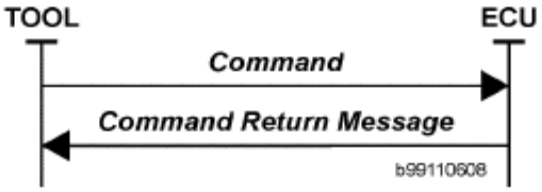
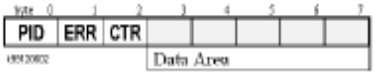


Figure 9: Command Return Message

Event Message – The Event Message is a specific type of DTO used to inform the tool of the internal status changes of the ECU. This information can then be used to invoke error recovery or other services. The Event Message allows the ECU to report any errors to the tool that have occurred since the last CRO was sent by the tool. The CCP specification describes the available error codes.

Figure 10: Event Message

The CRM and the Event Message have the same structure for the first three bytes of the message. The first byte is the Packet Identifier (PID). A CRM has the value 0xFF, and the Event Message has the value 0xFE. The second byte is the error code, and the third byte is the command counter value sent by the tool in the last CRO message. The remaining bytes are used for data relating to a particular response.



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

Figure 11: Structure of CRM and Event Message

DAQ-DTO Message – The Data Acquisition (DAQ) message is a specific type of DTO used to send measurement data to the tool. Before the ECU can send DAQ messages, the tool must send initialization messages. The ECU is informed in the initialization process of which measurement data needs to be sent to the tool. The tool also sends information on whether the measurement data is event driven or periodically sampled. Then the measurement data values can be sent to the tool without the tool first having to send a CRO to request the information.

Figure 12: Data Acquisition Message

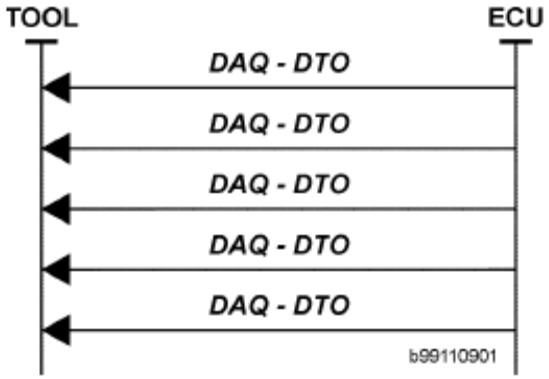


Figure 13: Synchronous Data Acquisition

During the initialization process tables are setup in the ECU to identify the location of the measurement data. Each table can contain up to seven memory addresses. The table is then assigned a unique packet identifier PID that is used for the first byte of the DAQ message. The other seven bytes are used to send the requested data to the tool.

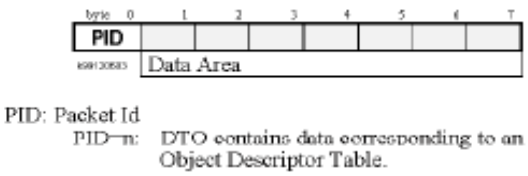


Figure 14: Structure of Data Acquisition Message

Object Descriptor Tables – The tables that are used to organize the location of the measurement data are called Object Descriptor Tables (ODTs). An ODT describes the contents of a single CAN message for data acquisition. Each ODT stores up to seven address locations where the measurement data is stored. The unique Packet Identifier PID is then assigned to the ODT to identify the data. Since a unique PID is needed to identify the measurement data for each ODT, and the number of data bytes that is sent back per ODT is limited to seven data bytes, multiple ODTs may be necessary to store all the requested measurement data. A DAQ list contains all the ODTs for a particular event or time period. Multiple DAQ Lists are needed when data needs to be acquired based on different events or time periods.

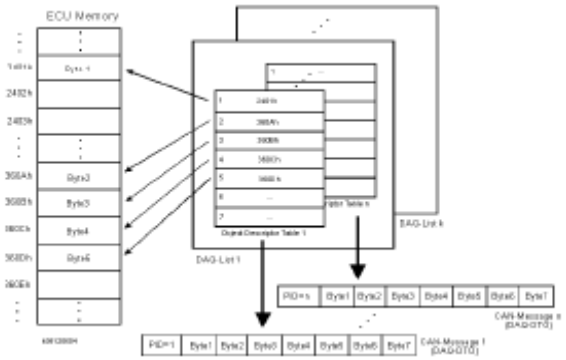


Figure 15: Overview of DAQ Lists with Multiple ODTs

The ECU receives the CAN message with the SHORT_UP command. The ECU then responds to this message with the appropriate CRM. Figure 14 shows the data needed for the SHORT_UP response. The first byte (byte 0) of the response is a Packet Id of 0xFF. This indicates that the DTO is a CRM. If the ECU is able to respond successfully to the request, the second byte (byte 1) is the command return code with the value 0x00. (The complete list of command return codes are listed in the CCP specification.) The third byte (byte 2) is the command counter value received in the CRO. The last five bytes contain the actual requested data. In the above example, only four data bytes were requested by the CRO; therefore, the tool disregards the data in the last byte (byte 7).

The CCP driver is developed for the ECU to support the commands described in the CCP specification. The commands must be received from the CAN bus, and processed in the ECU to provide the appropriate response which must be sent back out on the CAN bus. Two methods of obtaining measurement data are possible. A simple polling method can be implemented that sends data only after a request message from the tool. The DAQ list can be implemented when more throughput of data is needed or when the data needs to be obtained synchronously.

The CCP driver can be implemented with all the commands described in the CCP specification, or it can be scaled down to include only the commands needed for a particular implementation. The CCP implementation in the ECU can be broken down into two parts. The first part is a command processor, which is able to receive the required CRO commands and send the appropriate CRM. The second part involves the DAQ processor, which is responsible for sending the required DAQ list information at the appropriate time. Figure 16 illustrates the two main components of the CCP Driver.

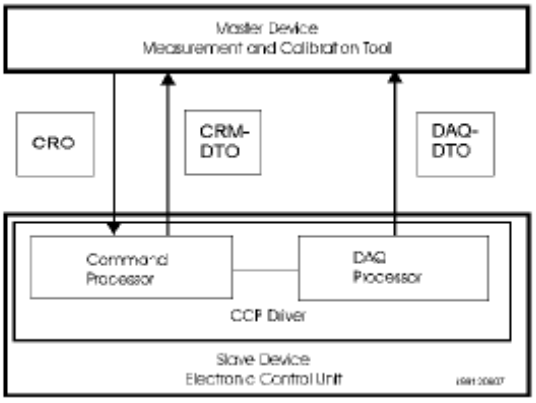


Figure 16: CCP Communication

In summary, the ECU receives commands codes and the related parameters in the Command Receive Object (CRO) to carry out internal functions or memory transfers. The Data Transmission Object (DTO) is used by the ECU to respond to the CRO, to indicate any errors conditions, and to transmit measurement data to the tool.

CCP Applications:

Basic development uses for CCP include
• Real-time ECU information (basic read and write function).
• Real-time access of ECU parameters (data acquisition).
• Real-time adjustment of ECU process algorithms (Calibration).
• In system or in-vehicle evaluation of design concepts.
• Evaluation of engineering design modifications.
• In system (or in-vehicle) Flash Programming.
• Emulation-type operation beyond the lab bench.

Coupled with the right tools, CAN Calibration Protocol is suitable for several module development activities. CCP allows development outside of the traditional software engineering environment. Beyond the engineer's desk and the engineering lab, module development on the road or on the test track is not only possible but several companies are already at this advanced stage.

CCP Driver Implementation:

The CCP Driver must be implemented in the ECU before any interaction with the tool can occur. The CCP commands will be sent in a particular order by the tool to elicit the appropriate information from the ECU. The ECU only needs to implement the correct responses to each command as required by the CCP specification. The CONNECT command must be processed like other CCP commands; however, the ECU must also store additional information about the status of the connection.
As required in the specification, the ECU must not respond to any CCP messages unless it has processed a CONNECT command with the proper station address. Therefore, the CCP driver must store the current status of the CCP connection, so that commands can be ignored or acknowledged as required by the CCP specification.

Initialization Sequence – In a typical tool application, the tool sends the connect command with the station address of the ECU. The ECU responds with the correct CRM-DTO if the station address is correct. Next, the GET_CCP_VERSION command can be issued to allow the tool to determine if the ECU is compatible with CCP implementation 2.0 or 2.1. Then, the EXCHANGE_ID command can be issued for automatic session configuration. The ECU responds by sending the Station Identification name's length and setting the Memory Transfer address to the appropriate memory location with the Station Identification name. The tool can then request the Station Identification name. Each version of the ECU software can be described by a different database. The EXCHANGE_ID can be used to make sure the tool database version matches the ECU software version. Figure 17 illustrates this sequence.

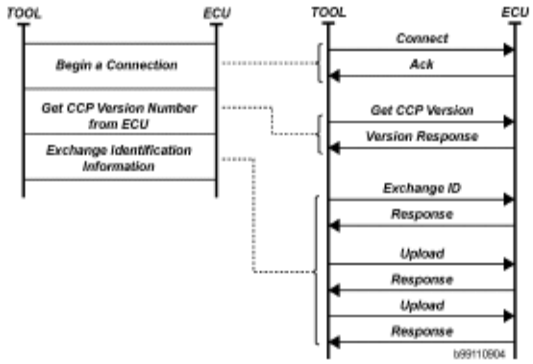


Figure 17: Example Initialization Operations

DAQ Operations – If the DAQ list is used for data acquisition in a typical tool application, the tool must first inquire about the DAQ storage information in the ECU. For example, the user may request information based on two DAQ lists, but the ECU has only been configured to store information for one DAQ list. Once the tool determines that the setup in the ECU is sufficient for the current requested measurement data, the DAQ lists can be configured. Then the command to start the DAQ information transfer is sent by the tool. The ECU must continue to send the required measurement data until the DAQ STOP command is received from the tool. Figure 18 illustrates an example of the DAQ operations sequence.

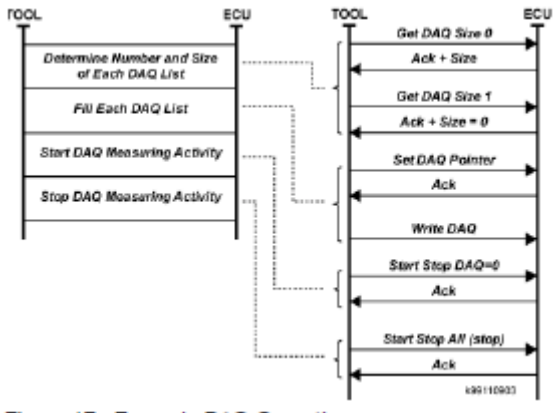


Figure 18: Example DAQ Operations

Disconnect – When the connection from the tool to the ECU is no longer needed, the tool can issue the DISCONNECT command. The CCP specification allows for either a temporary disconnect or for a complete termination of the calibration session.

CCP Implementation Requirements:

To implement CCP, the software developer needs the following three items:

• CCP specification document.
• CAN bus connection to the ECU.
• CCP Software Driver.

Figure 19 illustrates the functionality needed for CCP CAN message reception.

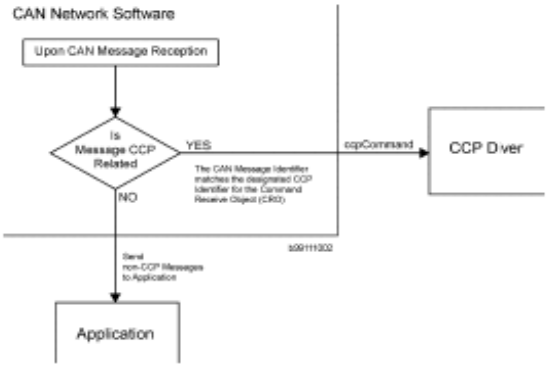


Figure 19: CCP CAN Message Reception

The CCP driver code must also be able to interface to the CAN driver code for the transmission of CCP messages. When the CCP driver calls a function in the CAN driver to transmit the CCP message, the CCP driver needs to be informed when the message is transmitted successfully. When the CAN driver calls a function (ccpSendCallBack) to inform the CCP driver that the last CCP message was transmitted successfully, the CCP driver is then able to call the CAN driver function to send another message.

This confirmation process prevents the CCP Driver from overloading the CAN transmit buffer with CCP messages. This also allows the application some control over how often the CCP messages are sent if the reception and transmission is integrated into the operating system of the application. Figure 20 illustrates the functionality need for CCP CAN message transmission.

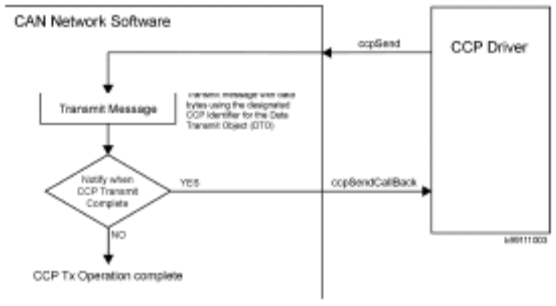


Figure 20: CCP CAN Message Transmission

Figure 21 gives an overview of the interfaces needed to implement the sample CCP driver. For more information on implementing the free CCP driver provided by Vector, please refer to the CCP Driver Implementation in Electronic Controls Units in the Reference section.

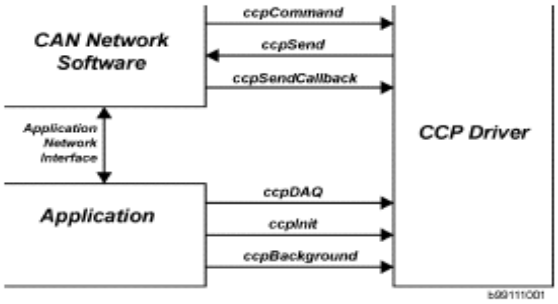


Figure 21: CCP Sample Driver Interface

CCP Resource Requirements – The CCP software driver will consume resources such as RAM, ROM and CPU time in the ECU. The code size of the CCP software driver depends on which optional features are implemented. The following list indicates the resource requirements for an implementation with 1 DAQ List and 3 ODTs. This allows for storing of up to 21 bytes of data in an intermediate buffer.

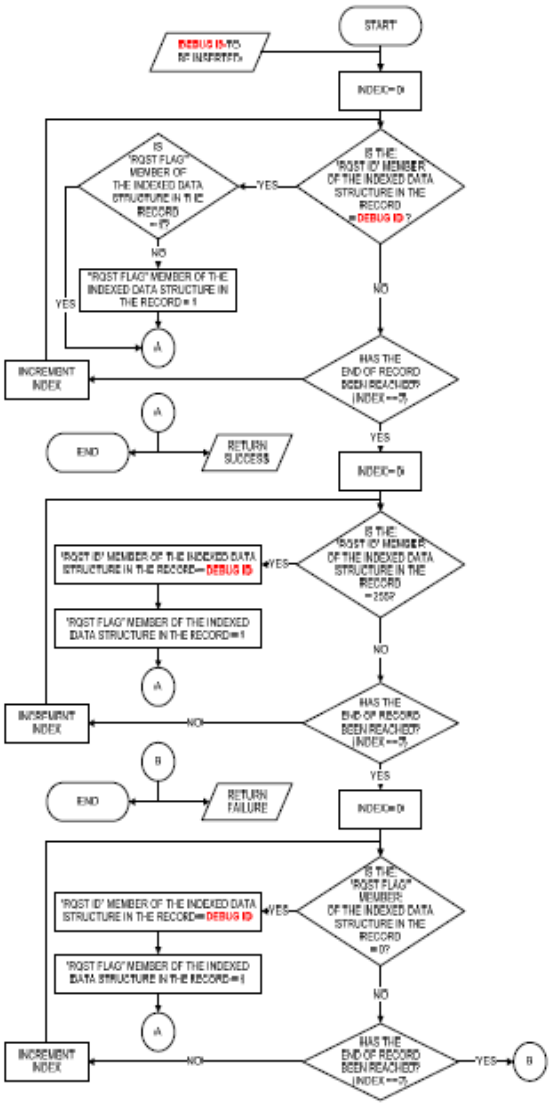
CCP Performance Ratings – CCP performance depends on a number of factors. The response latency time of the services in the ECU affects the CCP performance. In other words, the amount of time allocated by the ECU operating system for the CCP driver to perform its functions greatly affects the performance of the CCP driver. In addition, CAN bus conditions such as bit rate, busload, and bus priority level of the CCP message also affect the performance of the CCP driver. With a bit rate of 500 kBit/sec and typical load conditions, a burst memory transfer of ~5-10 kBytes/s and data acquisition rates of ~25 kBytes/s were obtained. A burst memory transfer would include uploading calibration values and flash programming. The data acquisition rate is for the synchronous data acquisition of 100 values every 10 msec. Each of the 100 values were two bytes in length. These results were achieved with an implementation in a Siemens 80C176 16 MHz processor.

CAN Messages Handler – Development Process:

Traditional Development Approach – In the early phase of a typical Automotive ECU software development project, it was necessary to make the initial analysis on the vehicle ECU's CAN network. The above phase is characterized by a large effort to write a specification to describe the CAN interfaces for the new ECU that must be in compliance with other connected ECUs. Therefore, the System Engineer prepares a specification document for the CAN Interface based on the vehicle's pre-existing CAN network database.

Document–based specifications are shared among different teams, each for that team's ECU. Errors due to misunderstanding between them may have impacts up to the Implementation phase and then only be detected during the System Testing and Validation phase. Based on system engineering documentation, the software engineers manually write the code relevant to the *CAN Messages Handler*, increasing the risk of propagating errors throughout the next process phases and of having additional

implementation errors. This process suffers from a variety of drawbacks, including the difficulty of keeping the documentation updated. For the OEM, this process has the disadvantage that the software has to be newly developed for each ECU development project, both if the supplier changed and if the OEM requires adding a new ECU.



FLOWCHART 1: REQUEST HANDLING

Efficient Development Process - One objective of the Embedded Software Development is to participate to the development of the ECU's application software, with particular emphasis on OEM strategic functions. A second objective is to apply an efficient development process for software development and validation. The approach used implied the development and early validation of the embedded software in a graphical design and subsequent use of automatic C code generation and its reuse in different ECUs with minor migration effort.

Vehicle ECU Network Database – During the vehicle's evolution from a simple ECU network to a very complex ECU network, the OEM system engineers are defining the Network Nodes and storing all information in a specific network database, to be updated in case of change on the vehicle network. Usually the CAN Network Database, specific for each vendor, contains CAN message definitions for an entire vehicle in terms of message identification, start bit, number of bits for each signal within a given message, byte order (Intel/Motorola), data type (signed, unsigned, etc..), conversion rules, applicable range, default value, comment. Commercial tools, such as Vector products, facilitated the system engineers' work. The system engineers use the database to generate the specification document and to perform the System Test, where they verified and validated the Vehicle ECU's network.

The CAN-DB Import Tool that creates a configuration file in a specific MATLAB format from a standard CAN database file which includes all the communication information (for example the bit meaning for each byte of each CAN Message). Offline importing is scalable, supports multiple platforms and it is independent of the standard (for example J1939).
The CAN Frame Configurator allows the system software engineers to configure (through a user-friendly GUI) the process-chain (filter, fault check, recovery) applied to each signal coming from (sent to) the CAN bus. In other words, the system software engineer can easily use the data (bytes) coming from the Rx CAN messages and create the data for the Tx CAN messages. Individual words, bit strings, and word segments within a CAN message can be defined as data fields, allowing the binary data passed onto the bus to be converted into a meaningful format. Data fields can be named (e.g. vehicle speed or engine speed) and relevant unit measurements can be specified and used when requested by the application software.

The adaptation process required an appropriate CAN interface implementation, both in terms of CAN Driver and CAN Message Handling, according to the target platform's basic software.
Its complete application to our scenario should have involved the following development phases:

- ECU network redefinition in terms of CAN network database adaptation.
- CAN messages specification for the new ECU.
- Manual implementation (manual coding, unit test and SW documentation preparation).
- System testing using Hardware-In-the-Loop.
- System testing on the vehicle.

Timing Aspects of CAN: CAN uses *non-destructive bitwise arbitration* process to select a message when message collision occurs. The simultaneously transmitted messages are allowed to enter the bus, while the process monitors their identifiers bit by bit. Message transmissions can go on, as long as the messages' identifier bits are the same. Once the difference is detected, the message with a passive bit (1) will have to stop, while the message with dominant bit (0) continues. The message which is backed off will be transmitted again as soon as the CAN bus is free.

By giving each data a unique identification number, same priority message collision where all the messages have to be taken off line, can be avoided. In practice, more important or urgent messages are given lower identification numbers, which give them higher priorities to be transmitted. Under the CAN protocol, the message with the highest priority

always get the first access to the bus, which effectively guarantees its delivery time. It can be seen that a problem can arise when a large number of high priority messages are transmitted, and lower priority messages will have to keep backing off transmission. This may cause long delays before lower priority messages can be sent.
The vehicle dynamic control systems considered here are safety related. Hence it is vital that their control related messages, for example a wheel speed data, arrive in time. It is, therefore, important that the CAN message delays of the above vehicle electronic architectures are examined. An excessive delay of signal transmission is a delay longer than the period of the transmission, causing the delayed data to be obsolete. The results of message missing or late arrival, forcing ECUs to use previous, out-of-date data could be hazardous. Thus, a simulation of CAN data transfer between ECUs had been carried out.

Simulation Data

In order to realistically simulate this operation, real information on data signalling in the target vehicle is needed, including all the other messages that would be using the CAN bus. The signals include all classes (Class A, B and C), characterised as low, medium and high speed messages, respectively.

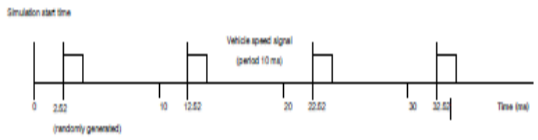


Figure 22: Timing diagram of a 10 ms period signal generation

From ECUs linked by arrows to Bus Queue represents, data queuing to be transmitted from ECUs onto the CAN bus. An ECU will let its signals join a queue one by one, equivalent to an ECU attempting to transmit one signal at a time. More than one signals in the Bus Queue at a time symbolises message collision. The Bus Queue arranges incoming signals in order according to their priority, equivalent to CAN message contention. The highest priority signal is put in front of the queue. The Bus Queue then let the highest priority signal onto the CAN bus (displayed as a door image) once the bus is free. Each signal occupies the CAN bus for 64 ms, equal to the time taken to transmit the 8 byte message.

Simulation Run

A simulation was run for an equivalent of 1 real-time second at time. One second covers the periods of all the signals except for signal no. 117, whose period is 10s and hence of little significant to the CAN bus load. Since all the signals are assumed periodic, any longer simulation run would give a repetitive result to the 1 second run.
For each architecture, the simulation was run 100 times with different sets of random numbers. Each simulation was run for 1 real-time second. A number of simulations were run in order to simulate different possibility of messages arriving on the CAN bus at different times.
Each simulation involves 4800-6800 messages getting access to the CAN bus. The time which the two groups of class C signals (of period 5 and 10ms), which are for real-time control, wait in the CAN Bus Queue plus the transmission time was collected. This is equivalent to the signal time delay associated with CAN in real applications.

Working with P-CAN:

The PCAN-USB adapter from Peak-System Technik enables simple connection to CAN networks. Its compact plastic casing makes it suitable for mobile applications.
The opto-decoupled version guarantees galvanic isolation of up to 500 Volts between the PC and the CAN side.
The package is also supplied with the CAN monitor PCAN-View for Windows® and the programming interface PCAN-Basic.

Figure 23: CAN Protocol Adapter

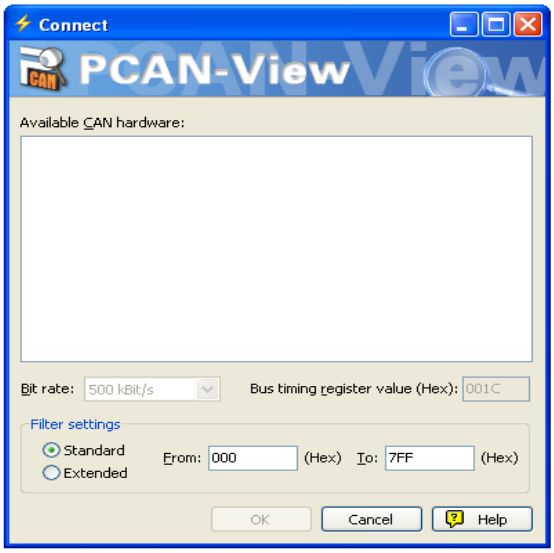


Figure 24: PCAN View 1.

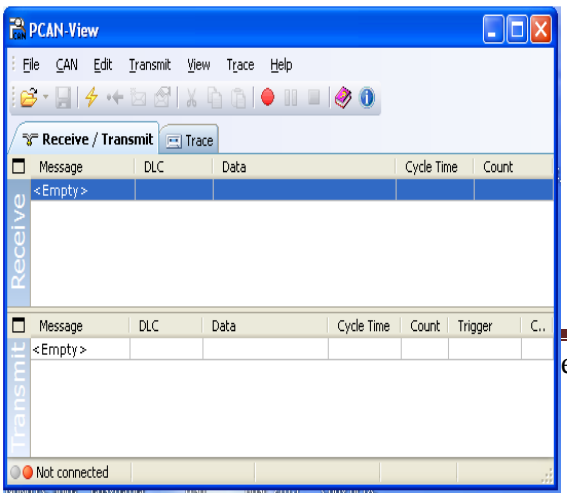


Figure 25: PCAN View 2. The PCA

N-USB adapter provides one CAN channel at computers with USB ports. Device drivers and programming interfaces exist for different operating systems, so programs can easily access a connected CAN bus.

Results and Analysis

The frequency distribution of the CAN delay of the two groups of class C signals are shown in Figures 26 and 27.

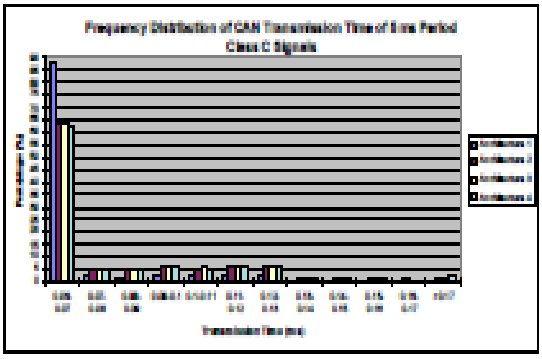


Figure 26: Frequency Distribution of CAN Transmission Time of 5 ms Period Class C Signals

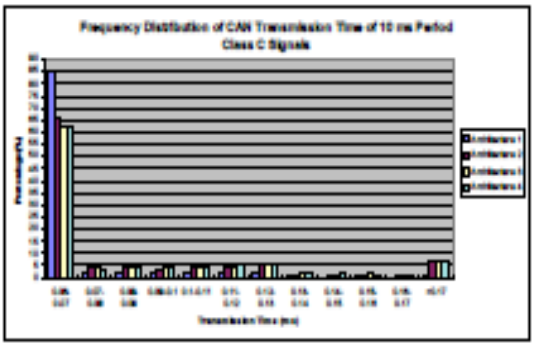


Figure 27: Frequency Distribution of CAN Transmission Time of 10 ms Period Class C Signals

From Figures 26-27, it can be seen that the majority of the two groups of class C signals are transmitted within 0.7 ms. The minimum possible CAN transmission time (no collisions) for each signal is 0.64 ms. This indicates that those signals are transmitted virtually without delay.

This can be seen from the higher percentage of signals with transmission time longer than 17 ms (the last bars on the chart) in Figure 6 than those in Figure 26. Also the 10 ms period signals have experienced longer worst case delay than the 5 ms signals. This could be expected, since the 10 ms period signals have lower priority than the 5 ms signals.

Conclusion:

The main object of this literature is to optimize CAN signal packing into CAN frames, to assign CAN Identifiers into CAN frame and distribute these CAN frames in the network with time in such a way that bus-load of CAN network as well as ECU load for all receiving ECUs in the CAN network are minimized.

It can be concluded that the proposed method increases the number of signals which can be transmitted and received by different ECUs in the network. This reduces the need to introduce another sub-networks and network gateways just to manage the bus-load. This results in reduction of engineering effort and cost for network development.

Aside from advantage of using a standard protocol rather than using a company-specific proprietary solution, CCP provides a wide range of functionality to help both the OEM and the module supplier in the development of electronic modules.

Having a complete set of tools to handle module calibration, test, measurement, diagnostic, and flash programming activities all within one protocol is a big technical advantage for the software engineer.

Another advantage can be concluded that proposed method ensures that all signals will reach receiving ECU from transmitting ECU within maximum allowed latency time for these signals. This improves performance of the distributed functionalities and customer satisfaction.

References:

1. Cortese, D., “Efficient CAN Protocol Development Process,” SAE Technical Paper 2009-01-1607, 2009, doi:10.4271/2009-01-1607.
2. Di Natale, M., “What CAN Go Wrong in CAN (Timing Analysis),” SAE Technical Paper 2009-01-1378, 2009, doi:10.4271/2009-01-1378.
2. CCP CAN Calibration Protocol, ASAP Standard, Version 2.1, February 1999
3. CCP Driver Implementation in Electronic Control Units, Version 1.18, Vector Informatik GmbH, 1999.
4. CCP, A CAN Protocol for Calibration and Measurement Data Acquisition, Rainer Zaiser, Vector Informatik GmbH.
5. B. Upender, “Analysing the Real-Time Characteristics of Class C Communications in CAN Through Discrete Event Simulation”, SAE 940133,1994.
6. J. Fenton, “Focus on Networking of On-Board Vehicle Electronic Systems”, Automotive Engineer, June/July 1996.

Author Profile:

Rahul Adsul: He is working as Assistant Manager with CIS Technologies (India) Pvt. Ltd. He is Doctoral candidate in Management from SIU, Pune and received MBA degree from YCMOU. He has completed B.E. from DYPIET, Pune University. He is lifetime corporate member of Institute of Engineers (India) under international body IEEE.

His research interests include fault diagnosis, ECU architecture management, vehicle health management and development and testing of ECUs.

Prof. Mrs. Sneha D. Joshi: She is Research scholar in COEP Pune. She has received B.E. and M.E and is pursuing Ph.D. at College of Engineering Pune. She is having 16 years of Teaching Experience. Her fields of interests are Modeling & robotics. Presently she is working as head of the E & TC department in PVPIT, Pune.