Implementing the CAN Calibration Protocol (CCP) in an SAE J1939 Application

William B. Vlcek

This paper presents the implementation of the CAN Calibration Protocol (CCP) on an electronic control unit (ECU) using the Society of Automotive Engineers' (SAE) Recommended Practice J1939 multiplex communications protocol. SAE J1939 uses a CAN-based network protocol, to which we added CCP to support calibration and measurement activities during the development and test of new application software. The use of a commercially available PC-based tool for calibration and measurement provided a cost-effective solution to support these capabilities for this development program.

Introduction

The approach taken in this paper is to first define the overall task with which we were presented, then identify the options available to satisfy the specific challenge discussed here — providing a cost effective means to calibrate and measure an ECU in development. This will be followed by a brief review of the communications protocols used, before getting to the discussion covering the implementation of our selected solution. The basic task discussed in this paper is how we provided the customer with a solution to their requirement to calibrate the application software in their ECU.

Task Definition

Our customer decided to develop their next generation electronic control unit (ECU). They chose to move from a Motorola MC68HC11 microcontroller, with application software in Assembly language, to a Motorola MC68376 microcontroller with the software to be written in "C". We were approached to develop the software due to our familiarity with the Motorola family of microcontrollers, and experience with developing and maintaining similar automotive software applications.

The ECU under development is targeted to work on a variety of diesel engine applications, and requires flexibility in the number and type of the inputs and outputs to be supported. The application contains a large number of programmable parameters, or calibrations. There is also a requirement to provide the capability to set and change these parameters using a service tool. Since this ECU interacts with other ECUs in heavy-duty truck applications, it must support the Society of Automotive Engineers (SAE) specification J1939 multiplex communications protocol. Other requirements include: monitoring internal operations of the ECU; reprogramming the ECU during its development; and providing a way to change calibrations while operating the engine.

Available Options

In considering the alternatives that related to calibration we identified two options: develop a calibration system from scratch, or find a commercially available solution. The analysis of the first option showed the cost to design a calibration strategy, to develop the ECU's calibration subsystem and counterpart service tool system (PC-based), then implement and test it, could have easily exceeded the cost of developing the application software it would support. The continued maintenance of such a proprietary calibration tool could have been substantial and was not in the plans of this development program.

The second choice, to identify a commercially available solution, was then reviewed. To simplify the communications capabilities of the ECU, the solution would preferably work with an existing protocol used by heavy-duty truck applications, either SAE J1587 or SAE J1939. One desire was to identify a CAN-based solu-

tion, which could be made to work with J1939, and thus allow the ECU to support only one communications protocol.

SAE J1939 Protocol Background

Before discussing the task in more detail, let us review some background on the technologies used. The industries which use large diesel engines — heavy-duty trucks, buses, construction equipment, and agricultural equipment — participate in the SAE's Truck & Bus Electrical and Electronics Committee. The Truck and Bus Control and Communications Network Subcommittee has developed a family of specifications for serial data communication over multiplexed wiring. Commonly known by the SAE specification number, J1939, these specifications define the complete ISO OSI seven-layer communications model. The J1939 network environment operates with shielded, twisted pair wire and uses the 29-bit extended format CAN data frame. The J1939 family of specifications cover not only the physical layer (Part 11) and data link layer (Part 21). It also includes a network layer specification (Part 31), definition of all application messages and their data content (Part 71), and definition of all diagnostic messages, including emissions-related diagnostics (Part 73).

The J1939/Part 21 specification presents information on the data link layer. It describes the message format used by J1939, detailing each bit in the extended format identifier (priority, source ID, etc.). And it describes the transport protocol used on a J1939 network, covering the packetization and reassembly of messages containing more than 8 bytes of data, and the management of the virtual connections for transferring this large message. The Part 31 specification defines the services used on a multi-segment vehicle network. These segments may operate with different physical media, protocols or data rates. So this specification describes the use of repeaters, bridges, routers, and gateways to provide the connections between segments.

The application layer defined by Part 71 supports both point to point and broadcast communications. Message frames contain 8 data bytes, of which some or all may be defined. Data bits not currently defined are to be transmitted as "1" and received as "Don't Care." This is to support future definition without affecting existing applications. The data bytes tend to be grouped by function in the message frames. For example, the message "Electronic Engine Controller #1" (EEC1) contains Status_EEC1 (Engine/retarder torque mode), Driver's demand engine – percent torque, Actual engine – percent torque, Engine speed, Source address of controlling device for engine control (which could be a device other than the one transmitting this message), and 2 undefined bytes.

The diagnostic messages defined in Part 73 are recognizable to anyone familiar with On-Board Diagnostics, Phase II (OBD II) as implemented by SAE J1979, E/E Diagnostic Test Modes (ISO 15031-5).

CCP Protocol Background

The CAN Calibration Protocol (CCP) is a standard maintained by the ASAP task force (Arbeitskreis zur Standardisierung von Applikationssystemen; English translation of this is Standardization of Application/Calibration Systems task force). The ASAP task force is composed of vehicle manufacturers, automation and test equipment manufacturers, and ECU manufacturers. The mission of ASAP, as stated in the CCP standard "is to reach mutual agreement and standardization in

- automation, modularization and compatibility of all equipment to do measurement, calibration and diagnosis, and
- manage the creation of a cost reasonable and sensible tool supplier market."

In keeping with this mission statement, CCP provides a CAN-based strategy for the calibration of ECUs and the acquisition of data from ECUs. The ASAP task force describes a model involving three components: Measurement, Calibration, and Diagnostics.

The communication strategy described for CCP is to utilize two CAN data frame

identifiers, one for the Command Receive Object (CRO) and one for the Data Transmission Object (DTO). CCP uses these two messages to operate in a master-slave style of network communications. The master device will be a measurement system, calibration tool, or diagnostic tool, and initiates communication with the slave device.

A CRO is the message sent from a master device (calibration tool in this case) to the slave device (our ECU). The CRO message contains a command code, command counter, and any command-related parameters and data. Valid command codes include writing calibrations, programming the ECU, and requesting data. A DTO is the message sent from the slave device to the master device. The DTO will contain the response to a command, an event message reporting internal status of the slave device, or the data supporting a previous data acquisition request.

For this specific implementation of CCP, we are using CAN message identifiers reserved by the J1939 specification for manufacturer proprietary functions. The 29-bit identifiers used are low priority and contain the destination address for this ECU.

Selected Solution Implementation

Following our ISO-9001 certified software development process, we decomposed the requirements specification provided by our customer and produced a software requirements specification. In the review cycle for this software requirements specification with our customer, the decision to support only J1939 CAN communications was made.

At the same time, we were conducting a trade study of commercially available development tools for the Motorola MC68376, real-time operating systems, and calibration tools. A number of criteria

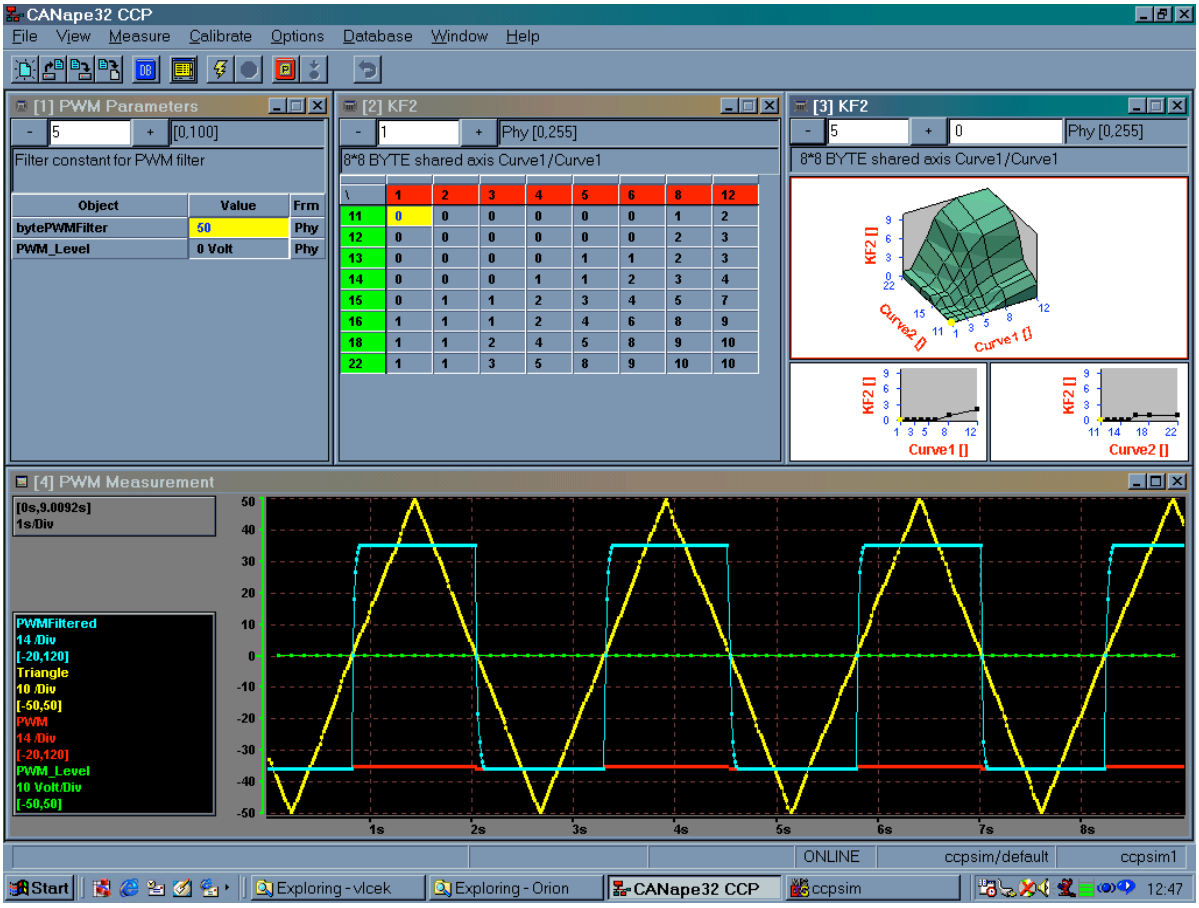
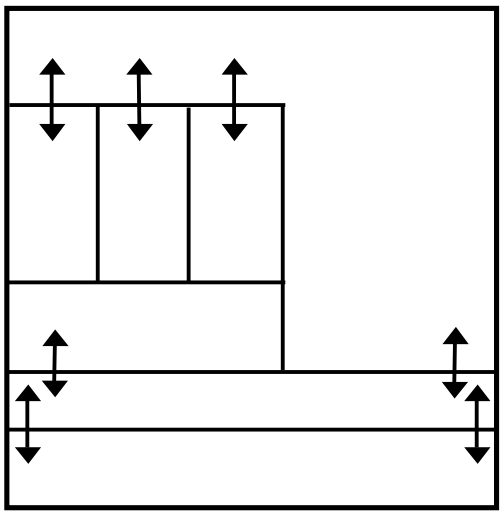


Figure 1 - Example CANape Screen

were considered when making our decision, including usability, suitability to task, follow-on technical support, and cost. After assessing the various options in the area of calibration tools, CCP was selected as the calibration methodology for this ECU development program. To work with CCP, a commercially available calibration tool was selected. At present we are using CANape from Vector Informatik, an example of the user interface is at Figure 1. However, by using CCP, the option to use a different PC-based calibration tool is possible.

Another goal desired by our customer was to have the flexibility to move the application software to a different microprocessor in the future. Such a move could result from cost considerations during production, or application software growth after adding new features. This goal fits well with our modular software development approach. A simple block diagram depicting the architecture of this application is provided at Figure 2.



To provide the capability to port the application to a new target processor, while keeping software rework to a minimum, we incorporated a hardware abstraction layer into our design. This approach is implemented as a series of functions which handle hardware specific actions, for example reading A/D registers, and placing the resultant data into data objects to be used by the application software. When the application software needs to perform a hardware action, it calls the appropriate function in the hardware abstraction layer, identifying the appropriate data object. The hardware function then performs the action, such as writing to a register, or driving a serial port. Within our communications subsystem, initializing the CAN controller, interacting with the receive and transmit buffers, and servicing all CAN controller interrupts, are similarly handled in the hardware abstraction layer. CAN message information becomes a data object handled by the communications subsystem. In the Motorola MC68376 microcontroller the on-chip CAN controller is known as the TouCAN. This CAN controller supports 16 transmit or receive buffers with three receive acceptance filters, and performs internal transmit arbitration to select the next message to attempt to transmit on the CAN bus.

One aspect of using CCP affected the development of our application software. In order to have data values available from the ECU for monitoring, and to provide the calibration flexibility required during development test, it was necessary to include in the detailed design widespread use of global variables in the software. To calibrate the software and support the data acquisition process, the CCP tool needs to know the memory address of these data values. So the tool reads the software memory map, and links the calibration and data acquisition objects to the address location of the appropriate global variable. Even though this action may be contrary to good design and programming practices, it is necessary in order to use CCP to satisfy development system requirements.

The J1939 communications subsystem software was designed and coded at Ascent Technologies, so as a result we could easily incorporate CCP into the structure. A portion of the high-level design is depicted in Figure 3. The path taken when a CAN message is received is as follows: the CAN message data object is provided to the Manage_Data_Link_Layer by the TouCAN-specific receive message

function. The Manage_Data_Link_Layer then performs the actions described in J1939/Part 21, determining if the received object is part of a multi-frame message, and reassembling the complete message if required. It then identifies where to send the received object. If it is a CCP message it is passed to Manage_CCP, if it is an application message it goes to Manage_J1939_Part71, or if it is a diagnostic

trouble codes (DTCs). And Manage_CCP operates similar to Manage_J1939_Part73, responding to DTO messages from the off-board tool, or directing a CRO to be transmitted in response to a previously scheduled data acquisition request. Manage_CCP, Manage_J1939_Part71, Manage_J1939_Part73 provide the transmit message data object to Man-

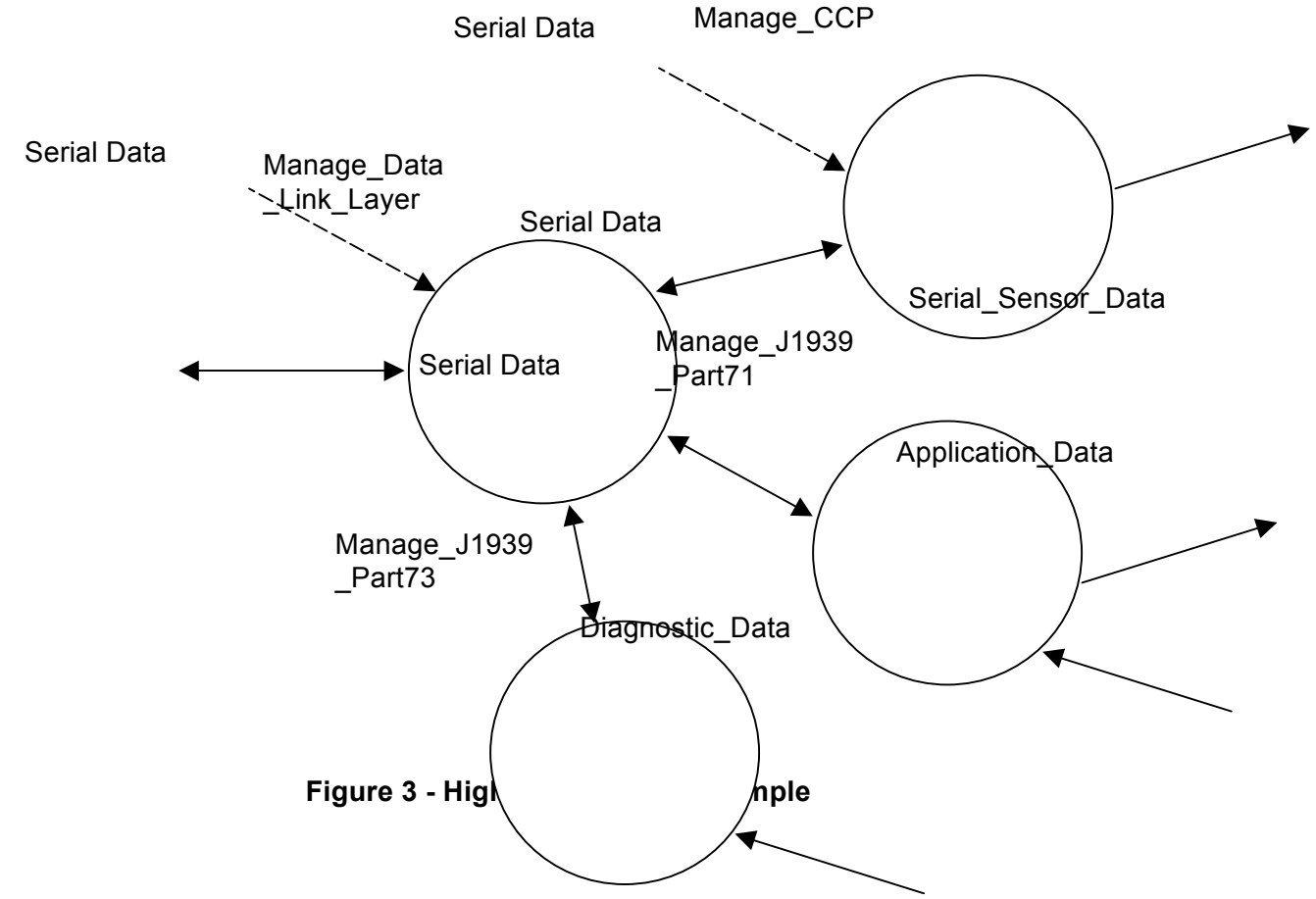


Figure 3 - High nple

message it will be sent to Manage_J1939_Part73.

Transmitting a message functions in the same fashion, only in reverse. For application layer messages, the application software will call Manage_J1939_Part71 to transmit the message. For diagnostic messages, Manage_J1939_Part73 will respond to diagnostic requests, or direct a diagnostic message to be transmitted announcing currently active diagnostic

age_Data_Link_Layer. The process, Manage_Data_Link_Layer, will determine if the data object requires a multi-frame message, preparing multiple CAN message data frames if necessary. It will then send the data object(s) to the transmit message function handling the TouCAN transmit buffers.

While the J1939 software was designed and coded in-house, we have benefited from the work at Vector Informatik with

CCP. They have generously made available to the public an example implementation of CCP on their Web site. We have incorporated this example software into our J1939 communications subsystem, modifying it to work with our operating system and adding the functionality which was absent, such as FLASH programming support for the FLASH memory part used in this ECU.

Conclusion

Use of the CAN Calibration Protocol has provided a cost-effective solution for calibration and measurement in this ECU development program. An off-the-shelf tool solution, offering more flexibility and capability than likely with a proprietary development, was identified and satisfies all the needs of the development and test engineers. Potential customers for this ECU application have indicated their interest in our use of CCP, and the opportunities it offers within their development and test environment. The approach to calibration and measurement utilized by CCP makes this suitable for any ECU with CAN as a communication link and requiring calibration, measurement, or both.

References:

1. ASAP Standard – CAN Calibration Protocol (CCP), version 2.1
2. SAE J1939 – Recommended Practice for a Serial Control and Communications Vehicle Network
3. CCP Driver, Implementation in Electronic Control Units, Vector Informatik GmbH (version 1.01)

Acknowledgements

Many thanks to my colleagues at Ascent Technologies and Jacobs Vehicle Systems, and for the assistance provided by the folks at Vector CANtech and Vector Informatik.

Ascent Technologies, Inc.

525 Avis Drive, Suite 15

Ann Arbor, MI 48108 USA

Phone 01-734-668-4035

FAX 01-734-668-2735

vlcek@asc-tech.com