High-Speed Reprogramming and Calibration with CAN FD: A Case Study

Armin Happel, Erik Sparrer, Oliver Kitt, Oliver Garnatz, Peter Decker
Vector Informatik GmbH

Reprogramming of ECUs as well as their in-vehicle calibration are typical and important automotive use cases requiring high data rates.
To meet the high timing requirements for reprogramming, techniques such as data reduction and parallelization have been used to optimize for CAN. Faster data protocols such as FlexRay and Ethernet have also been introduced. The first part of this case study compares these well-known and practice-proved measures with the capabilities of CAN FD. In particular its influences on the transport protocol and write/erase times of current hardware devices are demonstrated using a real environment.
For in-vehicle measurement and calibration the ASAM XCP Working Group already has extended the current version 1.2 to include the XCP transport layer for CAN FD. The second part of this case study shows the potential of increased data throughputs now possible with CAN FD due to the higher payload size of 64 bytes. Also shown are possible future XCP protocol enhancements which support simple portability of existing AUTOSAR ECU implementations of the XCP slave.

High Speed Flash Programming

Due to the continuous raising complexity of ECUs and its software sizes, a fast and efficient way to re-program In-Vehicle ECUs has become more and more important. The improvements in the past have been driven mainly by two factors: optimal performance of the flash download sequence, and introduction of newer and faster bus systems such as FlexRay, Ethernet or CAN FD.
As shown in Figure 1, the flash download can be divided into the three sequential phases, erase the flash memory, download and program segments of the software and afterwards verify if the data have been written successfully.

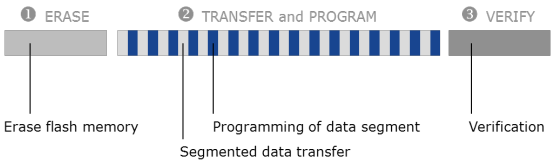

Figure 1: Programming phases

Optimizations have been concentrated to the download and programming phase. Since in most cases, the amount of data appears to be the bottleneck, the first approach was to apply a compression method which is adequate in performance and code reduction. In most cases, the LZSS[1] has shown a very good balance between compression rates compared to the resource utilization in small- and mid-sized microcontrollers. This can already provide a significant reduction in code size and download time of about 20-40% for little cost. The results are highly dependent on the entropy of data, but also on the performance of the microcontroller.
A further way to improve the download time is the introduction of “Pipelined Programming” (aka ‘Early-Acknowledge’ or ‘Double-Buffering’). The idea is to acknowledge the diagnostic service request “TransferData” immediately before the received data has been written to the flash memory. The message is then written to flash while the next one is received. Figure 2 shows the details of the diagnostic service flow.

[1] Lempel-Ziv-Storer-Szymanski-Algorithm, a method to mark redundancy in a data stream.

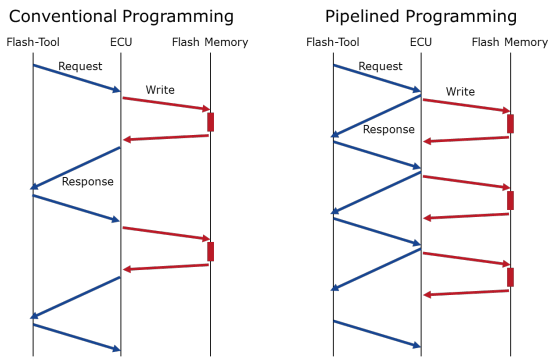

Figure 2: Optimize the download time with Pipelined programming.

Pipelined Programming shows the most beneficial effect when the data processing and programming time is smaller than the transmission time of a data segment. Even the combination with data compression provides an advantage when both can be done in parallel to the data reception. Since code can typically not be executed out of flash memory while it is programmed, the code parts for programming and communication must be executed out of RAM. In the following, an overview of the reprogramming time on different bus systems will be given, including CAN FD.

Programming with FlexRay

FlexRay is a time-triggered protocol with a bus speed of up to 10 Mbit/sec. All nodes on one network require a unique configuration for static and dynamic slots, slots per cycle, etc. The configuration is quite complex and essential to the performance of the system. On one hand the application requires space in the static slot for a reliable and time-triggered communication. On the other hand it is desirable to have several PDUs in one cycle with large payload for a fast diagnostic communication, e.g. for flashing, even though it's been used only in an exceptional case. A typical configuration uses 4-8 PDUs in one cycle each with payload from 42 to 255 bytes for diagnostic communication between the tester and the ECU. For more PDUs in one cycle the FlexRay schedule needs to be switched to a separate reprogramming cycle.

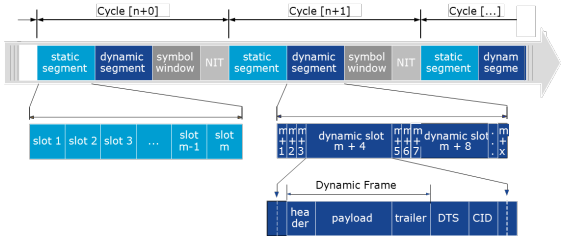

Figure 3: FlexRay configuration

For a fast reaction to a service request from the ECU, transmission of responses should be possible in subsequent cycles, e.g. in subsequent or periodically in every 4th cycle.

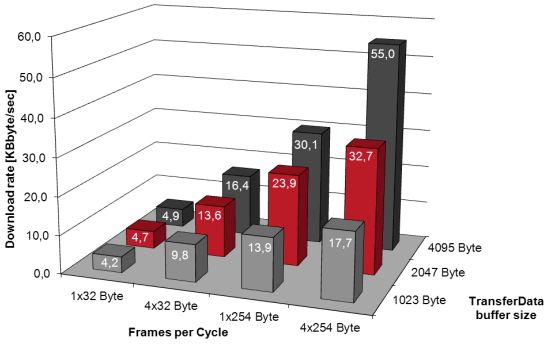

Figure 4: Transmission rate with FlexRay communication (without programming)

It can be seen in Figure 4 that the slot configuration has high influences to the data throughput, also the number of PDUs per cycle and the buffer size allocated for the TransferData service.
Table 1 shows the transfer and programming time that can be achieved on a typical FlexRay bus configuration. The measures were taken from a configuration with 8 PDUs per cycle and 42 bytes payload, and 6 PDUs per cycle and 255 bytes payload. The measure was taken with real programming sequentially and with pipelined programming.

Table 1: Transfer and programming time on FlexRay (kByte/sec)

FlexRay configuration	**Download rate (Kbyte/sec)**	
	Conventional Programming	**Pipelined Programming**
8 PDUs/cycle 42 bytes / PDU	32-34	~40
6 PDUs/cycle 255 bytes/PDU	40	60

Programming with Ethernet

ISO13400-2 specifies the communication for Diagnostics over IP (DoIP). This protocol is also used for re-programming ECUs over Ethernet. The relevant diagnostic communication is done over the TCP protocol. The next figure shows the architecture of an Ethernet Bootloader.

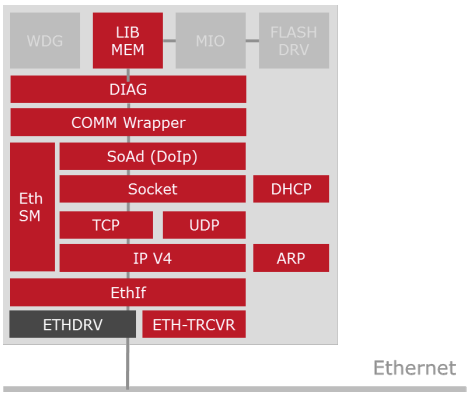

Figure 5: Architecture of an Ethernet Bootloader

This Bootloader has been implemented on a microcontroller where the pure flash write time takes about 180kByte/sec. Experimental downloads with a bus speed of 100Mbit were done with different buffer sizes for the TransferData-service to see the influence of this parameter. The results are listed in the table below.

Table 2: Transfer and programming time (kByte/sec) on Ethernet with different TransferData buffer sizes.

TransferData buffer size	Download rate (Kbyte/sec)
1 kB	95
4 kB	133
16 kB	150

Pipelined Programming has not been applied here. The size of the transfer segmentation has a direct influence on the download time. When using a 16kB buffer for one transfer segment, the download time reaches almost the throughput of the flash memory. The proportion of the data transmission time over Ethernet becomes relatively low compared to the write access time to the flash memory.

Programming with CAN and CAN FD

In this chapter measures of the rates with CAN and CAN FD are discussed. Since semiconductor manufacturers are still working on micros with an integrated CAN FD controller, an evaluation board was chosen where an FPGA with the BOSCH IP macros for CAN FD is connected to a microcontroller board.

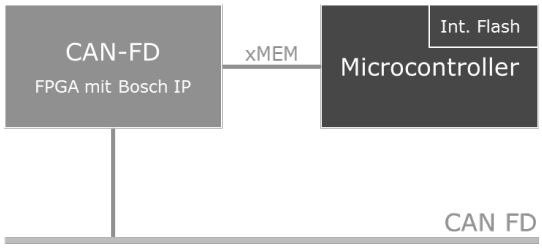

Figure 6: CAN FD Hardware environment for the evaluation project

A standard UDS CAN-boot loader-Software (FBL) has been used on the microcontroller. Only the communication specific layers needed to be adapted. The CAN-driver was exchanged to support the features of the CAN FD ASIC to transmit CAN-frames with up to 64 byte and bit rate switch (BRS) from 500kBaud up to 4 MBaud. The transport layer was based on an extension of the ISO15765-2 protocol as currently discussed in ISO. The segmented data transfer service has been kept to a maximum size of 4095 bytes. This reduced the necessary extensions to the transport layer to a minimum. CANoe has been chosen as the download tool to perform the software download. The flash programming sequence and the transport layer functionality were implemented in a DLL used by CANoe. The architectural overview of the measurement system is shown in the figure below.

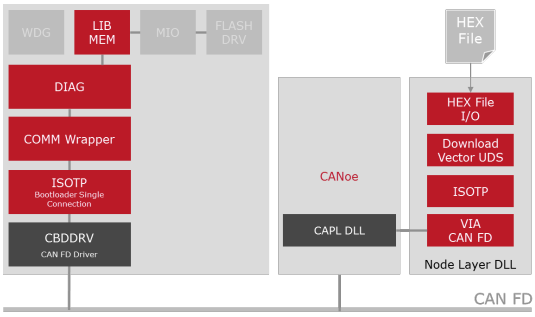

Figure 7: Architecture of the CAN FD Bootloader evaluation project.

The theoretical transmission rate on CAN of the ISO15765-2 transport layer with block size and STmin set to 0 provides an average transmission rate of 26-28 kByte/sec with 500 kBaud. On CAN FD, the DLC can be increased up to 64 which reduce the number of CAN-frames from 586 CAN frames down to 66 frames for a 4095 byte message. Using BRS with 4MBaud the transfer rate for 4095 bytes takes theoretically 270-370 kByte/sec. In practice the overall download was measured and programming time with DLC=8, without (500 kBaud) and with BRS (4 MBaud); and with DLC=64, without and with BRS were applied. The following figure shows the resulting transfer and programming rate for CAN and CAN FD.

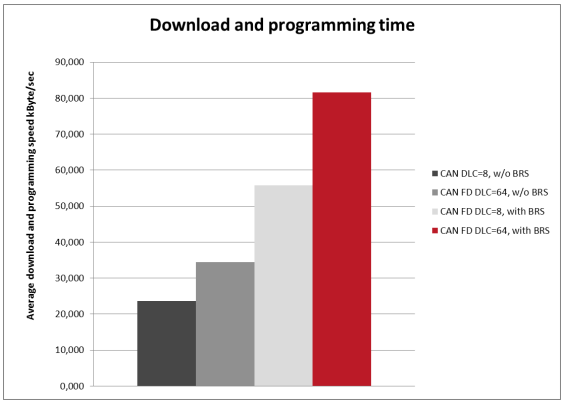

Figure 8: Transfer and programming time in comparison to CAN and CAN FD.

With DLC=8, without BRS, the overall programming time is getting close to the capability of the ISO15765 protocol on 500kBaud. But using CAN FD with BRS a download and programming time with more than 80kByte/s can be reached. This is far away from the theoretical capabilities of the bus, because now the flash programming time of the internal memory becomes a limiting factor. Faster flash programming time would be required to speed up the download time.
Even though these impressive data rates already exceed the capabilities of FlexRay, we want to further analyze the effect of compression and pipelining in combination with CAN FD.

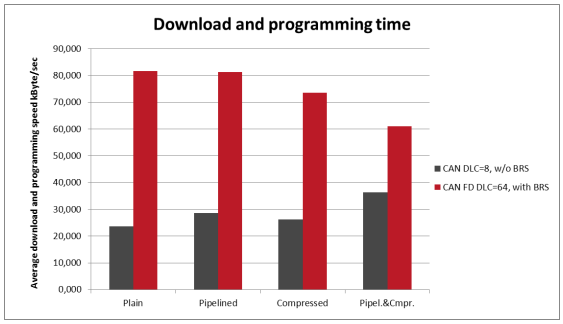

Figure 9: Download with CAN and CAN FD using plain, compressed, pipelined and combined programming.

The diagram shows (in blue) an advantage of pipelined and compressed download with 500kBaud. The combination of pipelining and decompression further increases the download time in this configuration to 38kB/sec. As long as the transmission of a message takes longer than the decompression and programming, both methods shows an advantage. With CAN FD the portion of time for data transmission is very low compared to the programming time (t_{tr} << t_{prog}). Therefore, the pipelining shows only minor improvements. It becomes even worse if compression is used. In this case, the CPU-time for decompression will be added to the download time and provides the contrary effect. The following figure illustrates the execution time and the delaying effect of the decompression.

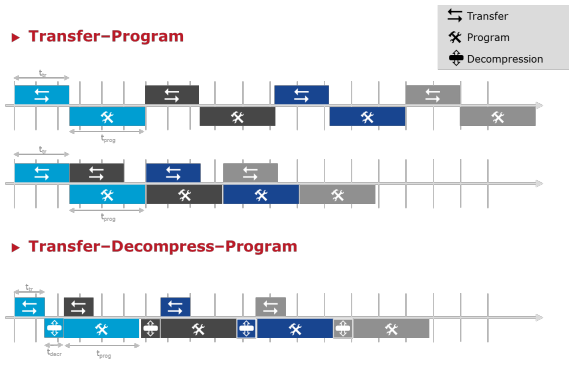

Figure 10: CPU dispatching for plain, pipelined and pipelined&compessed download

It must be noted, that the programming library and the CPU performance were not optimal for this evaluation. Further measurements must be taken on other microcontrollers to measure the beneficial factors of pipelining and decompression.
The measures show the capabilities of CAN FD to improve the download time. We can expect that now the programming time of the flash memory and the performance of the microcontroller will become the limiting factor for the overall programming time of an ECU.

Conclusion

Even though the measurements on the different buses have been taken on different controllers and therefore they are difficult to compare, a tendency can be seen that with Ethernet and CAN FD currently the bus speed exceeds the limits and the flash write time becomes now the limiting factor. This seems not to be the case for CAN and FlexRay.
Ethernet shows the best performance, especially on 100Mbit buses. However, the complexity of the communication stack and network configuration is high and also higher costs for the hardware are expected.
The FlexRay bus requires tricky configuration of the schedule to achieve a satisfactorily download performance, and this competes with the real-time data transmission in the static slot fields. The software complexity of the communication modules is higher than for CAN or CAN FD, but less than for Ethernet.
Pipelined Programming shows benefits in all configurations but is limited for CAN FD and Ethernet.

Outlook

The increasing complexity of in-vehicle functions will result in an increased amount of code size. Fast buses are required to provide a faster reprogramming time of ECU software during development and for series production.
CAN FD shows the capability to provide the performance in the future with adequate costs for software and hardware. Nowadays, the flash write time seem to become the limiting factor then, but steady improvements from chip manufacturers in flash erase and write time will then show benefits to the software download time.
Thorough software architecture and data handling will be needed in the future within the boot loader to fetch full performance.

In-Vehicle Measurement and Calibration

The second part of this paper focuses on the data transfer capabilities available for calibration proposes using XCP on CAN FD. First the interchangeability of CAN and CAN FD is explained by means of the Open System Interconnection Model. The subsequent analytical part provides a mathematical model to evaluate the payload throughput available at various transmission baud rates, which is validated by in-system measurements. Finally all results are summarized and an outlook on future developments regarding calibration over CAN FD is given.

The Open System Interconnection Model Applied to XCP over CAN FD

In state of the art automotive networks distributed service hosting and data sharing is a commonly applied technique. This requires a reliable, safe and efficient technical solution to facilitate the network communication. The Open System Interaction (OSI) Model introduces [5] a modular and maintenance friendly system architecture. The OSI model separates the various abstract communication requirements into 7 interaction layers. Each layer provides mechanisms for the interaction between layers and the logical connection between network nodes. These rules are known as protocol.

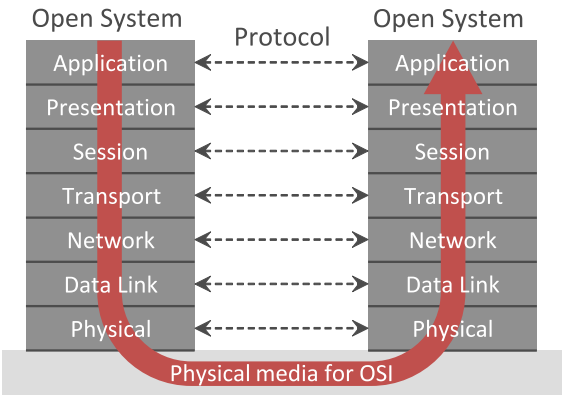

Figure 11: The Open System Interconnection Model

Since layers are independent of each other a system wide replacement of a single layer implementation does not affect other system layers.
With respect to the OSI model a calibration task using XCP over CAN (FD) maps protocols and the calibration use case to the OSI layers like presented in *Table 3*.
Like the CAN protocol, CAN FD applies to the basic two layers of the OSI model. Both protocols are based on a common physical layer. This enables CAN FD to reuse existing CAN transceiver hardware and to make use of the same voltage levels and bus topologies. The differences of CAN versus CAN FD do only affect the data link layer. According to the OSI model, a system wide replacement of CAN with the high performance protocol CAN FD does not affect XCP or the calibration application.

Table 3: Mapping of automotive protocols to OSI layers.

Protocol	OSI Layer
CAN (FD)	Physical Layer, Data Link Layer
XCP	Transport Layer
Calibration Use Case	Application Layer

To compare the performance of CAN and CAN FD a model for the data throughput is established and evaluated in the following section.

Evaluation of the Data Transfer Capabilities

The data throughput for CAN and CAN FD is estimated by a comparison of the frame size versus the payload size available for calibration purposes. A hypothetical busload of 100 % is assumed to calculate the available data throughput. The estimation is based on the size of every frame section provided in *Table 4* and *Table 5*. Since CAN does not provide an additional synchronization signal for signal sampling, the transition slopes of the data signals are used to synchronize recipients to the transmitter node. To ensure, that sender and receiver do not run out of synchronization a change of the logical level must be guaranteed within a defined limit. The CAN protocol specifies such a transition within 6 Bits. To ensure the transitions even in data that does not naturally provide them, CAN makes use of a bit stuffing algorithm and inserts a complementary bit after 5 equal bits. Therefore the actual size of a CAN frame depends on its content and cannot be predicted universally. Hence the throughput estimation for the payload applies a best and a worst case scenario where no respectively the maximum amount of stuff bits are inserted into the CAN (FD) frame.

Table 4: Field sizes of CAN Frames.

Name	Size [Bit]
Start Of Frame	1
Arbitration Field	12
Control Field	6
Data Field	≤ 64
CRC Field	15
Acknowledge Field	2
End Of Frame	10

Table 5: Field sizes of CAN FD Frames (applies to f_D).*

Name	Size [Bit]
Start Of Frame	1
Arbitration Field	12
Control Field (1st part)	4
Control Field (2nd part) *	5
Data Field *	≤ 512
CRC Field *	18 / 22[2]
Acknowledge Field	2
End Of Frame	10

The data throughput f_T is basically the quotient of the available payload b_D divided by the overall bit count of any frame fields b_F multiplied with the transmission frequency f (see (1)).

$$f_T = \frac{f \cdot b_D}{\sum b_F} \quad (1)$$

To serve the worst case scenario, the stuff bits are added to the divisor (see (2)).

$$f_T = \frac{f \cdot b_D}{\sum b_F - \left\lfloor \frac{\sum b_F - b_{EOF}}{5} \right\rfloor} \quad (2)$$

For the calculation of the data throughput of CAN FD it has to be taken into account, that major parts of the frame are transmitted with the frequency f_D (see (3) for the best case and (4) for the worst case scenario).

$$f_T = \frac{f \cdot b_D}{\frac{f}{f_D} \cdot \hat{b} + b} \quad (3)$$

$$f_T = \frac{f \cdot b_D}{\left\lfloor 1.2 \cdot \frac{f}{f_D} \cdot \hat{b} \right\rfloor + b + \left\lfloor \frac{\hat{b} - b_{EOF}}{5} \right\rfloor} \quad (4)$$

$$\hat{b} = b_{CF2} + b_D + b_{CRC}$$
$$b = \sum b_F - \hat{b} \quad (5)$$

[2] CRC polynoms with b_{CRC} = 17 Bit are used for payloads up to b_D ≤ 16 Byte, whereas b_{CRC} = 21 Bit apply to larger payloads.

Table 6: Definition of used terms

Term	Definition
f	Arbitration bit rate
f_D	Data bit rate
f_T	Bit rate of the average data throughput
b_F	Length of Frame/Bit
b_D	Length of Payload/Bit
b_{CRC}	Length of CRC/Bit
b_{CF2}	Length of CAN FD part of the control field

The calculation assumes a maximal payload for CAN and CAN FD. Using the later protocol the data bit rate f_D should be significant higher than the basic transmission frequency f. Hence, for the payload and the CRC polynomial the bit times are shorter and thus allow a higher throughput. In comparison to the payload transfer rate of CAN (see *Table 7*), this increase of the speed is reflected by the calculated data throughput band, as presented in *Figure 12* and *Table 8*.

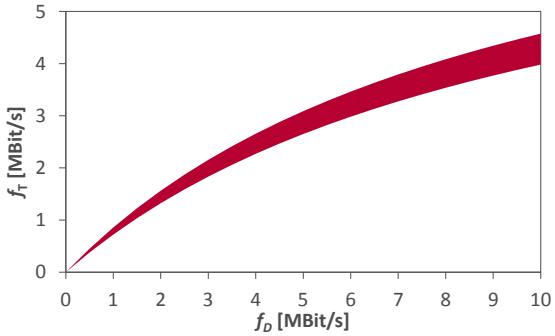

Figure 12: Visualization of the calculated throughput range.

Table 7: Calculated data throughput using CAN with a payload of 8 Bytes (f = 500 kBit/s).

Scenario	f_T [kBit/s]	Efficiency [% of f]
Best Case	288	58
Worst Case	244	49

Table 8: Calculated data throughput using CAN FD with a payload of 64 Bytes ($f = 500$ kBit/s).

f_u [kBit/s]	f_T [kBit/s]		Efficiency [%of$f$]	
	Best Case	Worst Case	Best Case	Worst Case
500	451	378	90	76
1000	858	721	172	144
2000	1563	1323	313	265
4000	2656	2271	531	454
5000	3088	2650	618	530
8000	4084	3537	817	707

Summarizing the above evaluation, the overall transfer rate boost of CAN FD is within a range of factor 1.5 up to 14.

Verification of the Model by Measuring XCP over CAN FD

To verify the evaluation provided in the last section, an in system calibration process is executed using XCP over CAN FD. The measurement environment consisted of Vectors measurement and calibration software, CAN/CAN FD hardware and a PC based Engine Control Unit Emulator with equivalent behavior to an embedded ECU. The transfer rate is examined by measuring the bus communication time. The timing of the data packages is measured between the in- and output of the CAN/CAN FD driver modules. An overview of the setup is given in *Figure 13*.
The time difference measured represents the time consumed to transmit the package over the bus plus an additional unknown delay caused by the CAN FD hardware driver and the transceiver hardware.

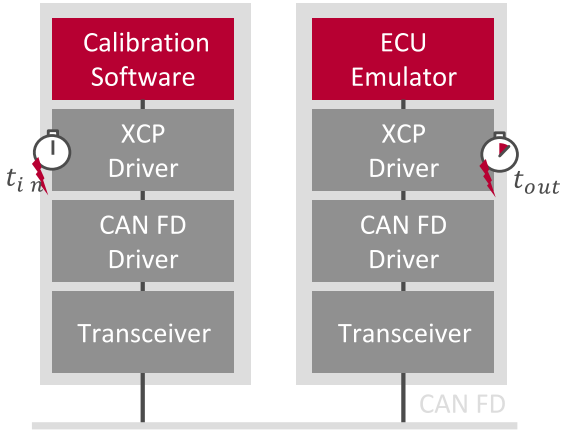

Figure 13: Setup for the verification measurement.

The measurements have shown that the delay time of the driver stack could be assumed to be constant. This leads to the equation used for the correction of the data transfer rate in (6) which is the quotient of the corrected transmission duration over the transferred amount of payload data.

$$f_T = \frac{\sum(t_{out} - t_{in} - t_{off})}{\sum b_u} \qquad (6)$$

To verify practical calibration use cases with the above stated model, the payload probability distribution $p(b_u)$, occurring at the investigated calibration process has to be taken into account.

$$\overline{f_T} = \sum p(b_u) \cdot f_T(b_u) \qquad (7)$$

Table 9*: Calculated data throughput for a realistic calibration process using CAN FD ($f = 500$ kBit/s).*

f_u [kBit/s]	$\overline{f_T}$ [kBit/s] Best Case	$\overline{f_T}$ [kBit/s] Worst Case
500	407	341
1000	753	635
2000	1318	1119
4000	2130	1825
5000	2438	2095
8000	3126	2707

The bandwidth corridor has been calculated using (7). The calculated results are shown in *Table 9*. Compared to the measured throughput (see *Table 11*) the calculated bandwidth corresponds to the presented model and confirms its correctness (see *Figure 14*).

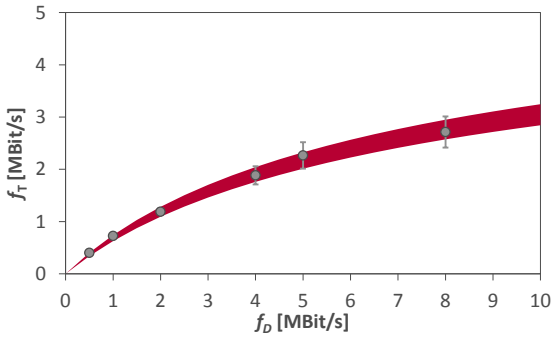

Figure 14: Data throughput range and measured data for the investigated calibration use case.

Table 10: Measured data throughput for a realistic calibration process using CAN ($f = 500$ kBit/s).

f [kBit/s]	306 ± 12

Table 11: Measured data throughput for a realistic calibration process using CAN FD ($f = 500$ kBit/s).

f_D [kBit/s]	f_T [kBit/s]
500	401 ± 21
1000	724 ± 46
2000	1189 ± 57
4000	1884 ± 172
5000	2316 ± 253
8000	2664 ± 298

Conclusion

Both CAN and CAN FD are restricted to the OSI layers 1 and 2. Hence it is not to be expected that an upgrade of OSI compliant automotive networks such as the improved CAN protocol will cause an extensive impact on higher protocol layers. On the basis of a calibration use case the upgrade feasibility for an established automotive network has been investigated and proven. The study focuses on the achievable data transfer rate for calibration using CAN and CAN FD respectively. The transfer rate is modeled mathematically and tested against an in-system measurement with a real system.
Depending on the selected data transfer frequency, the mathematical model predicts a payload bandwidth increase by a factor of 1.3 up to 9, which has been verified by the measured data (see *Table 9, Table 10*, *Table 11*).
This increase serves the requirement for high data transfer rates of state of the art automotive bus systems. It removes the existing bottle neck of the CAN protocol and hence enables higher level protocols to transfer huge amounts of data within short time. For this reason CAN FD is an elegant and simple way to boost the data transfer capabilities of established but maxed out CAN bus systems.

Outlook

Any ECU firmware implemented with respect to the ISO network layer model enables a transparent and rapid integration of a CAN FD driver. Thus features like the enhanced transfer rate can be used for any higher OSI layer communication with a minimum of integration effort. In practice a CAN FD driver for the calibration master is provided with the PC software which shifts the main effort towards replacing the embedded CAN driver in the ECU. The embedded XCP Driver however will further provide XCP packages of 8 Byte size only, which is the maximum capacity of CAN frames. Hence the XCP driver has to be extended to hand up to 64 Byte wide XCP-packages to the subjacent CAN FD Driver.

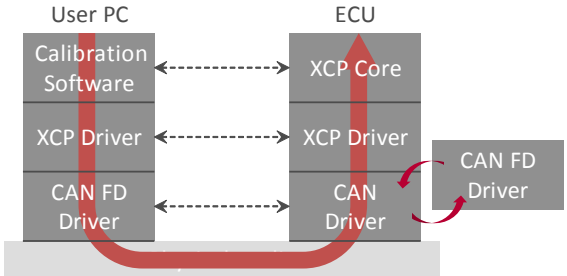

Figure 15: Least required ECU software changes accruing from an upgrade to CAN FD.

To upgrade an established network with minimal impact on the system and to achieve an optimal payload transfer rate a package concatenation feature should be introduced to XCP on CAN FD.

Established ECU firmware with CAN access is restricted to an 8 Byte wide data path. This allows an upgraded ECU to use the high speed data transmission frequency but not the full payload size of CAN FD. Since small payloads decrease the transmission efficiency due to multiple frame headers it is favorable to pool small XCP packages to fill a full size CAN FD payload. This optimizes the data transfer capabilities of the bus (see *Table 16*).

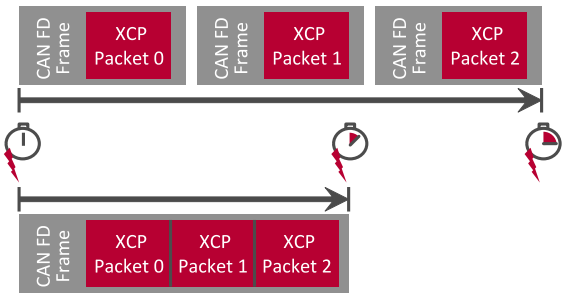

Figure 16: Bus occupation of small single packages (top) versus concatenated packages (bottom)

Package concatenation is not yet specified as a feature for XCP on CAN (FD). However a package concatenation feature would be an optional transport layer capability and would therefore further improve the XCP performance. To introduce package concatenation the XCP driver of any system participating on the bus has to support this feature.
Present research at Vector Informatik is focused on these optional capabilities. Vector is working on a proposal for an update of the XCP specification to introduce XCP package concatenation over CAN FD. Also a demonstration implementation as a proof of concept for XCP package concatenation over CAN FD is in preparation.

Armin Happel
Vector Informatik GmbH
Ingersheimer Str. 24
DE-70499 Stuttgart
Tel.: +49-711-80670-3624
armin.happel@vector.com
www.vector.com

Erik Sparrer
Vector Informatik GmbH
Ingersheimer Str. 24
DE-70499 Stuttgart
Tel.: +49-711-80670-3036
erik.sparrer@vector.com
www.vector.com

Oliver Kitt
Vector Informatik GmbH
Ingersheimer Str. 24
DE-70499 Stuttgart
Tel.: +49-711-80670-3027
oliver.kitt@vector.com
www.vector.com

Oliver Garnatz
Vector Informatik GmbH
Ingersheimer Str. 24
DE-70499 Stuttgart
Tel.: +49-711-80670-3615
oliver.garnatz@vector.com
www.vector.com

Peter Decker
Vector Informatik GmbH
Ingersheimer Str. 24
DE-70499 Stuttgart
Tel.: +49-711-80670-4805
peter.decker@vector.com
www.vector.com

References

[1] CiA DS 301, CANopen application layer and communication profile

[2] CiA DSP 302, Framework for CANopen managers and programmable CANopen devices

[3] Robert Bosch GmbH: "CAN FD Specification Volume 1.0"; http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can_fd_spec.pdf. (called 08/2013); 04/2012; 34 pages

[4] International Organization for Standardization: "Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signaling"; ISO 11898; 11/1999; 51 pages

[5] International Organization for Standardization: "Information technology – Open System Interconnection – Basic Reference Model: The Basic Model"; ISO 7498-1; 06/1996; 68 pages

[6] Decker, P(Vector Informatik GmbH): „CAN FD – Flexible Tools for Flexible Data Rates" at CAN FD TechDay; Detroit, USA; 10/2012; 26 slides

[7] Vector Informatik GmbH: "CAN FD Flexible Datarate CAN an Introduction"; CAN_FD_Introduction.pptx (internal source); Weilimdorf, Germany; 01/2013; 46 slides