

Automotive embedded systems software reprogramming

A thesis submitted for the degree

of Doctor of

Philosophy

by

Ralf Schmidgall

School of Engineering and Design

Brunel University

May 2012

Abstract

The exponential growth of computer power is no longer limited to stand alone computing systems but applies to all areas of commercial embedded computing systems. The ongoing rapid growth in intelligent embedded systems is visible in the commercial automotive area, where a modern car today implements up to 80 different electronic control units (ECUs) and their total memory size has been increased to several hundreds of megabyte.

This growth in the commercial mass production world has led to new challenges, even within the automotive industry but also in other business areas where cost pressure is high. The need to drive cost down means that every cent spent on recurring engineering costs needs to be justified. A conflict between functional requirements (functionality, system reliability, production and manufacturing aspects etc.), testing and maintainability aspects is given.

Software reprogramming, as a key issue within the automotive industry, solve that given conflict partly in the past. Software Reprogramming for in-field service and maintenance in the after sales markets provides a strong method to fix previously not identified software errors. But the increasing software sizes and therefore the increasing software reprogramming times will reduce the benefits. Especially if ECU's software size growth faster than vehicle's onboard infrastructure can be adjusted.

The thesis result enables cost prediction of embedded systems' software reprogramming by generating an effective and reliable model for reprogramming time for different existing and new technologies. This model and additional research results contribute to a timeline for short term, mid term and long term solutions which will solve the currently given problems as well as future challenges, especially for the automotive industry but also for all other business areas where cost pressure is high and software reprogramming is a key issue during products life cycle.

Content

1	Introduction.....	1
1.1	Vehicle's life cycle.....	2
1.2	Reprogramming within an ECU's life cycle.....	4
1.3	Aspects of software reprogramming	7
1.3.1	Automotive innovation vs. software size.....	8
1.3.2	Automotive development focus and priority.....	9
1.3.3	Automotive system complexity and compatibility.....	10
1.3.4	Automotive network aspects	12
1.3.5	Summary	14
1.4	Scope of the thesis.....	15
1.5	Organisation of the thesis	16
2	Background.....	18
2.1	Embedded Systems	19
2.2	Electronic Control Unit	21
2.2.1	Microcontroller	21
2.2.2	Memory.....	22
2.2.3	ECU Software Components Overview	24
2.3	Programming Control Unit (Test system).....	28
2.4	Programming Sequence	29
2.5	Communication Stack.....	33
2.5.1	Field bus systems	35
2.5.2	Media Access Control Overview	36
2.5.3	Transport Layer Protocol.....	36
2.5.4	Application Protocols	38
2.6	Network	39
2.7	Summary	40
3	Double buffered data transfer	41
3.1	Reprogramming Protocol	42
3.2	Double buffered data transfer	42
3.3	Method's utilisation.....	49
3.3.1	Mapping to Diagnostic Protocol ISO-14229 – UDS	49

3.3.2	Mapping to other application protocols.....	51
3.3.3	Mapping to multi controller systems	51
3.4	Conclusion	52
4	Field bus system protocol stacks.....	53
4.1	Controller Area Network	54
4.1.1	CAN bus protocol according to ISO 11989.....	54
4.1.2	CAN-TP according to ISO 15765-2	58
4.1.3	Complete reprogramming process based on UDS	70
4.1.4	Conclusion	71
4.2	FlexRay.....	73
4.2.1	FlexRay (FlexRay Specification 2.1)	73
4.2.2	FlexRay Transport Protocol (ISO 10681-2)	84
4.2.3	Complete reprogramming process based on UDS	91
4.2.4	Conclusion	93
4.3	Summary	94
5	Data size reduction	96
5.1	Partitioning.....	97
5.1.1	Analysis	97
5.1.2	Discussion	98
5.2	Fill byte skipping	99
5.2.1	Analysis	99
5.2.2	Discussion	100
5.3	Data compression	103
5.3.1	Analysis	103
5.3.2	LZ77 and LZSS Algorithm.....	104
5.3.3	Discussion	106
5.4	Differential file.....	107
5.4.1	Analysis	108
5.4.2	Discussion	111
5.5	Conclusion	112
6	Microcontroller Hardware Optimisation.....	116
6.1	Memory status information.....	116
6.1.1	Analysis	117

6.1.2	Discussion	119
6.2	Doubling interrupt vector tables	121
6.2.1	Analysis	121
6.2.2	Discussion	122
6.3	Conclusion	123
7	Network architecture	125
7.1	Introduction.....	127
7.1.1	Networking issues.....	127
7.1.2	Network types.....	128
7.2	Routing nodes (Gateways)	129
7.3	Routing strategy	130
7.3.1	Analysis	130
7.3.2	Discussion	131
7.4	Conclusion	135
7.4.1	Routing strategy.....	135
7.4.2	Network design.....	136
7.4.3	Summary	139
8	Reprogramming in parallel.....	141
8.1	Introduction.....	142
8.2	ECU schedule calculation.....	143
8.3	Discussion	147
8.4	Conclusion	149
9	Magnetoresistive RAM	151
9.1	Introduction.....	152
9.2	Discussion	153
9.3	Case study to the differential file approach	154
9.4	Conclusion	156
10	Case study – Software reprogramming	157
10.1	Software reprogramming via CAN	157
10.1.1	ISO15765-2 (CAN-TP) model evaluation	158
10.1.2	ISO14229 (UDS) on CAN model evaluation.....	162
10.1.3	CAN bus baud rate optimisation.....	166
10.1.4	ISO 15765-2 (CAN TP) Flow Control parameter Block size	168

10.1.5	ISO 15765-2 (CAN TP) FlowControl parameter STmin	170
10.2	Application Protocol ISO 14229 (UDS) Optimisation.....	172
10.3	Gateway optimisation	178
10.3.1	Buffer for the partly store and forward routing strategy.....	179
10.3.2	Increasing gateways clock frequency.....	181
10.3.3	Summary	182
10.4	Software reprogramming via FlexRay	182
10.4.1	Vehicle access by CAN bus system.....	182
11	Conclusion and Outlook	188
11.1	Summary	189
11.1.1	Method's performance potential.....	190
11.1.2	Method's potential vs. effort and costs	194
11.1.3	Utilisation in practice	196
11.1.4	Further work.....	197
11.2	Outlook.....	199
11.2.1	Automotive Ethernet	199
11.2.2	MRAM technology.....	199
11.2.3	Wireless access.....	200
11.3	Conclusion.....	200
12	Figures.....	202
13	Tables	205
14	Bibliography.....	207
A	Journal Paper – IEEE TVT	A-1

Acknowledgement

I thank my supervisor Dr. Ian Dear from the Brunel University for enabling and supporting the research presented in this thesis. The long time we discussed about the reprogramming topics have been always inspired. He found always the right words for motivation also if research progress was not given. Thank you, Ian!

Thanks are owned to Professor Dr.-Ing. Werner Zimmermann from the HS Esslingen - University of applicant science. He supported me since my first university studies and found always time to discuss new trends, protocols or technologies within the automotive diagnostic area. The author work to our book was the base to my interests in automotive communication systems.

I thank the students I supervised during their master thesis for their contributions to several data transfer acceleration topics (in alphabetical order): Samir Karic (ODX-V Development), Susann Kunde (FlexRay introduction), Rolf Molzahn (CAN network analysis), Sascha Neumann (network analysis with Symta/S) and Alexander Stock (FlexRay test system).

Many thanks to my colleges of the department for diagnostic development (GSP/OVE) at the Daimler AG (in alphabetical order): Dennis Artz, Viktor Brester, Gunnar Gaisser, Stefan Glattes, Michael Hiljegerdes, Andreas Kopf, Stephan Römer and Tobias Tetzlaff. Thank you for your support and the good technical discussions during the last years.

Thanks to Andreas Theissler for the good and inspired discussions about software reprogramming as well as for the support during writing the papers or the thesis. Also many thanks to Professor Dr. Joachim Goll for his support.

Lastly, I would like to thank my family for all their love and encouragement. Thanks to my parents who always supported me in all my pursuits. Thanks to my children Salome and Miriam for the understanding that her father was always working at the weekends.

And most of all for my loving, supportive, encouraging, and patient wife Kerstin for the support during all the years and especially during the final stages of this Ph.D. Thank you!

Acronyms, terms and definitions

ALU	Arithmetic Logic Unit
Application	The term “application software” represents the compiled binary code of an ECU.
ASAM	Association for Standardisation of Automation and Measuring Systems
AUTOSAR	Automotive Open Software Architecture
BDC	Binary Delta Compression
BC	Bandwidth Control (refer to ISO10681-2)
BS	Block Size (refer to ISO15765-2)
BSW	Basic Software
CAN	Controller Area Network
CCP	CAN Calibration Protocol
cf.	Latin “confer” – compare
CiA	CAN in Automation
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA/CA	Carrier Sense Multiple Access / Collision Avoided
CSMA/CR	Carrier Sense Multiple Access / Collision Resolution
e.g.	Latin “exempli gratia” – for example
et al.	Latin “et alii” – and others
ECM	Engine Control Module
ECU	Electronic Control Unit
E/E	Electric and electronic
EEPROM	Electrical Erasable and Programmable Read Only Memory
EMC	Electromagnetic compatibility
EOP	End of production
EPROM	Electrical Programmable Read Only Memory
etc	Latin “et cetera” – and so on
FIFO	First in first out
HIS	German “Hersteller Initiative Software”
HMI	Human - Machine Interface
ISA	International Society of Automation
i.e.	Latin “it est” – that means
ISO	International Standardisation Organisation
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LIN	Local Interconnect Network

LTE	Long Term Evolution
MCD	Measurement, calibration and diagnostics
MOS-FET	Metal oxide semiconductor field-effect transistor
MOST	Media Oriented Systems Transport
kb	Kilobit
kBit	Kilobit
kB	Kilobyte (1024 byte)
kByte	Kilobyte (1024 byte)
Mb	Megabit (1024 kb)
MB	Megabyte (1024 kB)
msec	Millisecond
MRAM	Magnetoresistive Random Access Memory
MTJ	Magnetic Tunnel Junction
NVM	Non Volatile Memory
ODX	Open Diagnostic Data Exchange
OEM	Original Equipment Manufacturer
OICA	Organisation internationale des Constructeurs d'Automobiles
OPEN	One Pair Ethernet
PC	Personal Computer
PCI	Protocol Control Information
PCU	Programming Control Unit Initiates and controls a reprogramming process.
PDU	Protocol Data Unit
PLL	Phase-locked Loop
PROM	Programmable Read Only Memory
RAM	Random Access Memory
ROM	Read Only Memory
RTE	Runtime Environment
SAE	Society of Automotive Engineers
SDU	Service Data Unit
SID	Service Identifier
SOP	Start of Production
STmin	Minimum Separation Time (refer to ISO15765-2)
TTCAN	Time Triggered Controller Area Network
VCO	Voltage-controlled Oscillator
XCP	Universal Measurement and Calibration Protocol
\$01	“\$” indicates a hexadecimal (hex) nomenclature.

1 Introduction

Content

1.1 Vehicle's life cycle	2
1.2 Reprogramming within an ECU's life cycle	4
1.3 Aspects of software reprogramming.....	7
1.3.1 Automotive innovation vs. software size.....	8
1.3.2 Automotive development focus and priority.....	9
1.3.3 Automotive system complexity and compatibility.....	10
1.3.4 Automotive network aspects	12
1.3.5 Summary	14
1.4 Scope of the thesis	15
1.5 Organisation of the thesis	16

Ever since the invention of the car by Carl Benz (1844-1929) 125 years ago engineers have been striving to improve performance, increase reliability and reduce costs. Early innovations were based around purely mechanical leap forwards in technology. However, over the last 30 years electronic systems have been rapidly taking over the technology advances to improve functionality, performance, reliability and reduce costs. Currently, more and more functionality, which was implemented in hardware in the past, is implemented in software today. In-addition, what was futuristic driver assist or even replacement technology is now a reality due to this computing power. If we consider the mid to high range end of the market it is now warrantable to talk about highly complex computer systems on wheels. This revolution is primarily due to the rapid trends in more powerful microcontrollers and communications technology. Complex mathematical algorithms, which required complete computer centres in the past, can be calculated on a single powerful microcontroller today. This added to the revolution in real-time sensor technology, and has created a new era for personal transport technology. A consequence of this

trend in rapid expansion of software quantity in embedded systems is the need to consider the hidden cost of increasing software quality and maintainability in embedded systems. With focus on the automotive industry, one of the most fitting statements about the automotive future is:

The majority of all automotive innovations will be within the electrics and electronics (E/E) area. The vast part will be software.

This prediction is supported by nearly all vehicle manufacturers [Dra11], suppliers [Hau11] and scientists [Bro11]. Of course, the percentage values vary but the basic statement is equal: The quota of electronics (hard- and software) and as a result complexity will continuously increase. This thesis is also supported by the Oliver Wyman Automotive's study "Car Innovation 2015" given by J. Dannenberg and J. Burgard et al. They predict that "electrics and electronics will remain the most important enabler of automotive innovations through 2015 and beyond, and will grow by six percent annually" [Dan07].

These prospects have a simple consequence: increasing amounts of software results in longer software reprogramming time for those embedded systems. That was not a problem so far but today software size has been increased in such a way that fundamental activities within the product life cycle, e.g. initial programming of the Electronic Control Units (ECUs) during production process or software updates within service / after sales can no longer be handled within adequate time windows. Expanding reprogramming time finally results in economical and therefore financial disadvantages for the automotive industry. This aspect raises some crucial questions:

- 1) Why is software reprogramming such an important issue?
- 2) Why is software reprogramming process acceleration necessary?
- 3) Is there a basic approach to solve the reprogramming challenge?
- 4) Are there any other industries that have the same essential problems?

Within this chapter an overview is given to software reprogramming aspects with the aim to provide answers to the crucial questions according to ECUs' or embedded systems' software reprogramming process.

1.1 Vehicle's life cycle

Compared to other electronic systems, vehicles and therefore automotive ECUs have a quite different life cycle. As depicted in figure 1.1-1 a car is typically developed between 6 and 8 years. The model line is manufactured and sold also between 6 and 8 years. After the vehicle is manufactured the maintenance time period starts where the first 2 -3 years a

warranty time is given. The challenge within the automotive industry is the combination of a mass product with a long time life cycle combined with the high dynamic in electronic development and technology's evolution.

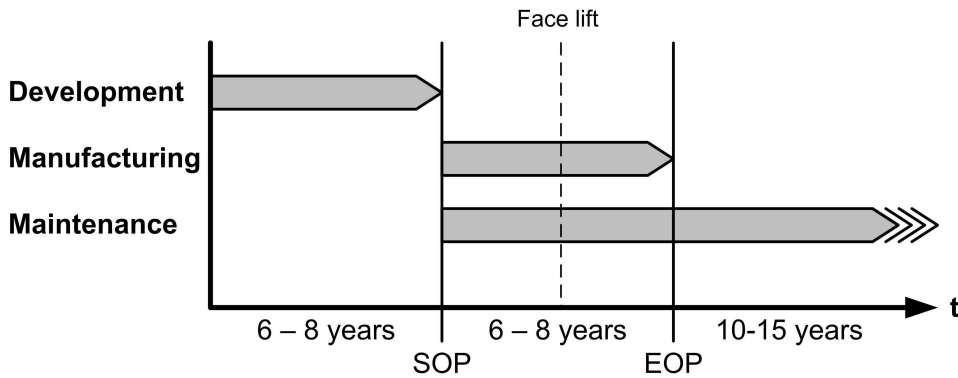


Figure 1.1-1: Vehicle model line life cycle

It is a common intention of all vehicle manufacturers to reduce the development period as well as the manufacturing time (hours per vehicle) for new model lines. Hence, several model lines will derive from a basic platform. This allows the reuse of electrical and electronic components (e.g. sensors, actuators and ECUs) to reduce costs. Exact values for the different periods in figure 1.1-1 as well as numbers of derived model lines can not be given because these are commercial sensitive data, and these vary for different vehicle manufacturers.

Figure 1.1-1 also depicts, that an electrical design decision (e.g. ECU functionality, communication network architecture etc.) have been made a long time period before the model line is initially produced (Start of Production – SOP). These decisions will also influence the maintenance processes for that model line several years after model line's end of production (EOP) time. Typically significant changes of such design decisions can not be modified so easily. It might be possible to make some new design aspects in the middle of a model line production period, but typically significant changes are not intended e.g. bus architectures, network design.

The thesis is looking at vehicles that are now in the development or production stage and identifies the current problems associated with the embedded system's software reprogramming process. The aspects above also make a contribution to the discussions within this thesis.

1.2 Reprogramming within an ECU's life cycle

Today electronic control unit's application software reprogramming is an important issue. At the very least, if the ECU is manufactured the application software still has to be initially programmed. Depending on the final usage and the final product's life time, an ECU could be reprogrammed several times during its life cycle. Figure 1.2-1 depicts an abstract overview. Below the typical reprogramming stages within a vehicle ECU's life cycle are described with focus on automotive ECUs.

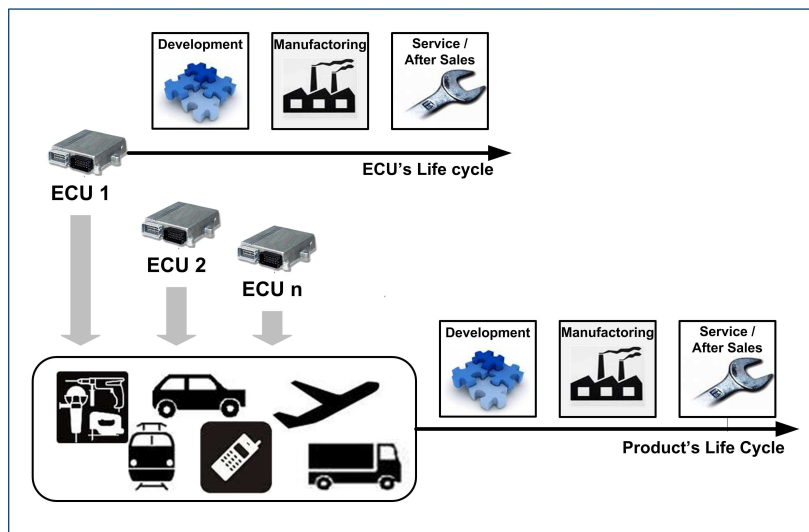


Figure 1.2-1: Reprogramming stages within an ECU's life cycle

Software reprogramming during the development process

During the ECU's development process the software may be reprogrammed several times because new functionality is developed or bugs are fixed. Especially in an early development state only a few ECUs are available for testing. Within typical ECU development processes the different features and the complete functionality are not available on the first (early) sample. The functionality increases step by step and therefore reprogramming of an ECU's application software is an essential issue. Figure 1.2-2 depicts the current typical software volumes for automotive ECUs depending on their assignments.

If the ECU is part of a more complex system (e.g. vehicle, plane, train, machine etc.) functionality could be distributed over several ECUs. In that case it might be necessary to reprogram application software of more than one ECU, e.g. for bug fixing purposes or to have the latest software version for testing. Especially the last aspect has a strong correlation to software reprogramming time: with focus on the automotive industry, a modern high-end class car includes up to 80 different ECUs where each ECU provides several functionalities. If such a vehicle-in-development is prepared for test drives (e.g. winter tests in Scandinavia or heat tests in the USA etc.) the most recent software for each ECU

should be programmed. Hence, the total amount of time per vehicle should be as small as possible, especially if a fleet of several vehicles is prepared.

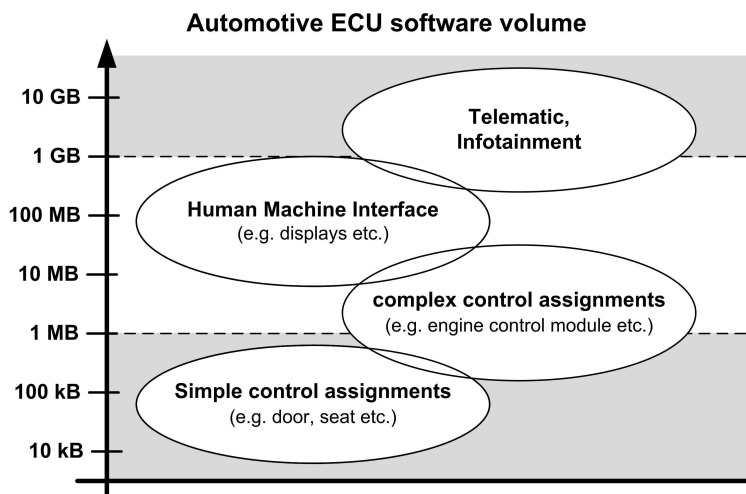


Figure 1.2-2: Automotive ECU software volume

Another reprogramming scenario during the development process is the adaptation of software to a vehicle, e.g. parameterisation of an engine control unit to the engine. In that case the parameter set has to be reprogrammed several times until the final parameter set is found. Until the engine control unit is reprogrammed the ignition is off. Hence, the engine is not running and the electrical power for all ECUs is supplied completely by the vehicle battery. To guarantee a correct process without low system voltage interruption the programming time should be as fast as possible.

Software reprogramming during manufacturing process

ECU's application software programming process is an essential part of the value chain. During the ECU's production process the final ECU software is programmed into the target system. Two different scenarios could occur: 1) ECU's application software is completely programmed during the ECU manufacturing process within the ECU assembling line or 2) one or more software fragments are programmed later within the vehicle assembling line. Of course, in both cases the final application software is programmed, but due to the different reprogramming places the total reprogramming time has different consequences.

If the ECU is programmed completely within the ECU's manufacturing process (scenario 1), the ECU could be programmed before packaging. Thus the microcontroller's internal interfaces¹ are available and usable. The software is programmed very fast but

¹ refer to chapter 2

typically without standardised protocols. This strategy is preferred if only one software variant of the ECU exists and no specific software adaptation for the final control assignment is necessary.

If the ECU has a specific software part depending on the final control assignment it might provide some benefits if the final software is programmed at the vehicle assembly line. For example, an engine control module (ECM) could have different parameter sets depending on the number of the engine's cylinders. If vehicles with different engines are produced on the same assembly line and the ECMs have been delivered finally programmed, each ECM variant allocates storage place. Late programming at the vehicle assembly line (scenario 2) provides a) economic benefits because of the smaller and less complex part storage requirements and b) assembly complexity is decreased because the manufacturer has only one device for selection. These economical benefits will not be achieved if the total software programming process time significantly increases the assembling line clock. Here a strong necessity is given to quantify and reduce the reprogramming time.

Software reprogramming within service or after sales market

Software reprogramming is an important repair method for the vehicle manufacturers in the aftersales or service market. If customer's complaints could be solved by a new software release reprogramming is the preferred repair method. As discussed above, ECU software sizes vary in a range of several kByte up to several MByte depending on ECU's control assignment (refer to figure 1.2-2). As a consequence today's reprogramming process time is in regions of several minutes up to hours based on the currently given automotive bus systems (discussed later in section 1.3.2).

The total time for the software reprogramming process has an immediate economic impact. The more time required the higher the costs are. For the garage the equipment (e.g. Programming Control Unit (PCU), power supply etc.) and the working area are occupied during that time. For the customer the vehicle is not usable. If the total time for software updates enlarges up to hours economical and therefore financial disadvantages occur, e.g. if a truck requires a longer garage time. In case of a software bug the vehicle manufacturer has to pay the down time costs.

Another aspect is to decrease the risk of process errors. Depending on the existing environmental conditions, a software reprogramming process is more or less stable. The more time a reprogramming process requires the higher the risk of interruption. The acceleration of a software reprogramming phase reduces this risk and provides a more stable and reliable process.

Table 1.2-1: World automotive production [Vda11]

	2010	2011	change in %
Passenger vehicles	63.377.724	66.237.761	+ 4.5
Commercial vehicles	14.180.650	14.033.714	-1.0
Total	77.558.374	80.271.475	+ 3.5

The increasing software sizes in future vehicles will increase the software reprogramming times, too. If it is assumed that only one ECU of each new produced vehicle will be reprogrammed within the warranty period, the increased costs will be enormous. As depicted in table 1.2-1 the world automotive production was up to 66 million passenger cars in 2011 [Vda11]. A reprogramming process cost reduction, e.g. by reprogramming time reduction etc., of 1 € provides world wide potential up to 80 million € per year.

Summary

The necessity to reprogram ECU application software is given during the complete product life cycle. The increasing software size provides new problems which have never existed before. The result is an economical disadvantage that could be solved by decreasing software reprogramming time. Generally this topic is not only relevant for the automotive industry. The increasing software reprogramming time of a plane requires a longer down time in the hangar or machines are longer non-productive. However, the cost market of the business areas is completely different. Compared to the automotive industry, where vehicle piece costs are responsible for the cost pressure, in other industries the maintenance costs are important.

Hence, approaches to accelerate the software reprogramming process in the automotive industry are necessary, especially if the predictions and forecasts as mentioned above become true.

1.3 Aspects of software reprogramming

Today ECU application software reprogramming is an important issue within an ECU's life cycle. Especially in the after market business software reprogramming is a powerful repair method to solve software errors and in some cases it is the only repair method. Of course, the reprogramming process was established years ago, but with focus on the current automotive industry software reprogramming and the required time for this process becomes continuously more important. Also environmental aspects of software reprogramming are explained as well as an overview of history and reasons are given why the current situation is as it is.

1.3.1 Automotive innovation vs. software size

The automotive market is highly competitive. Innovations and new technologies are influencing customer's decisions when they select a new car. As already mentioned, the majority of all automotive innovations will be within the electrics and electronics (E/E) area. Today a significant part of vehicle's characteristics is made by software. This trend was already recognized in 2006 by W. Huhn and M. Schaper as they wrote in the McKinsey on IT report: "the focus and value in engineering products is shifting from chips to code" [Huh06]. Today the amount of software is up to 100 million lines of code and thousands of functions are controlled by software [Bro11]. M. Broy has analysed the vehicle software ratio during the last 40 years and identified an exponential increase [Bro11]. This statement is supported by the analysis of the software ratio within Mercedes-Benz vehicles since 2004 (refer to figure 1.3-1). Starting in 2004 the amount of embedded systems' software has doubled every 2.5 years. The tendency will be supported by the next S-Class generation in 2013 where again doubling of the software size is predicted.

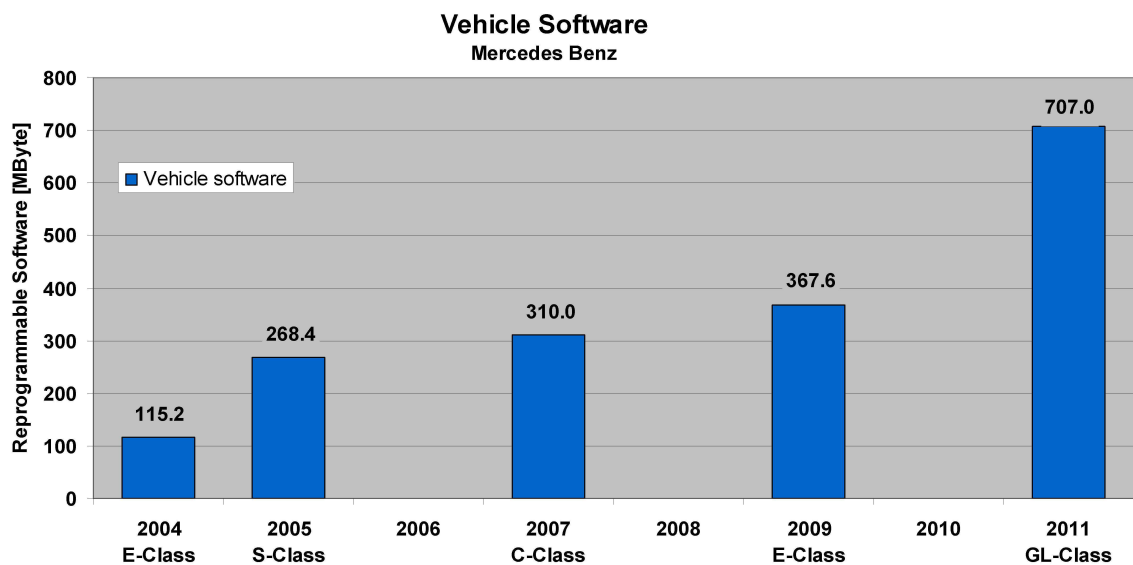


Figure 1.3-1: Amount of vehicle software of Mercedes-Benz

The exponential growth has now reached the boundary where the amount of software is so high that the old concepts for software reprogramming are no longer able to fulfil the required process time limits (e.g. given by the assembly line clock).

1.3.2 Automotive development focus and priority

Compared to other industries the automotive industry has a very special role. In 2010 up to 77.6 million vehicles were manufactured worldwide². In contrast to the manufacturing of planes or industrial machines with less than 1000 parts/year³, manufacturing cost aspects are in focus. M. Broy et al. [Bro11a] describe several reasons, where typical issues to reduce system complexity by decoupling system layers are not or only partly implemented because of cost aspects. With focus on software reprogramming only the following three examples of his list are mentioned and evaluated:

- The number of running processes on a single microcontroller is increasing such that runtime behaviour and the schedule of those implemented processes have to be extensively organised.

The consequence is that new software (e.g. tasks or processes) is allocated on that microprocessor as long as resources are free. Clearly structured and layered software architecture is not implemented because memory intensive interfaces have been optimised for code size and runtime aspects. Also functionalities of different layers are combined to save memory resources. This results in an increasing software complexity and the necessity to reprogram the complete application software instead of single software parts.

- Microcontroller's memory is so scarcely dimensioned that additional functionality is only possible by an expensive step to the next microcontroller's memory size.

Another aspect from cost discussion's point of view is the fact that microcontroller manufacturers supply microcontrollers with tiered memory sizes. The step to the next memory size results in higher costs for the microcontroller (significantly higher costs during model line life cycle because of the high number of parts/year). Hence, the above described optimisation of code size is the consequence in order to use the given memory size.

- Compared to Ethernet commonly used within the PC industry, the simple and less resource consuming bus systems have less bandwidth and a strong dependency between physical layer, transport layer and application layer. Hence, they can not be parameterised independently.

Figure 1.3-2 provides an overview about the currently most important and most implemented bus systems.

² OICA - Provisional Production Statistics 2010 [OICA2010]

³ Refer to M.Broy et.al. in [Bro11a]

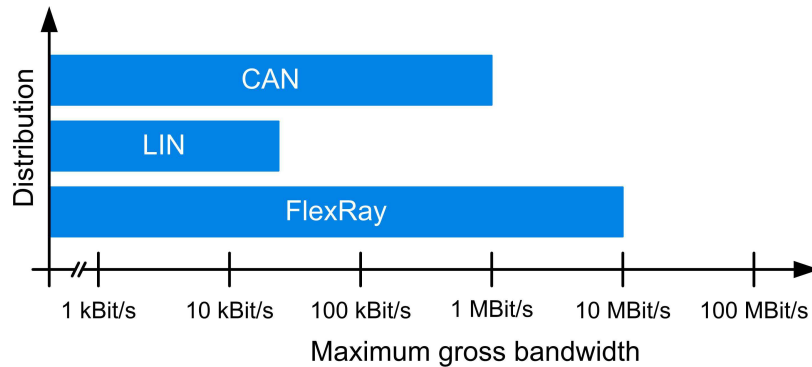


Figure 1.3-2: Most implemented automotive field bus systems

Until today these bus systems are sufficient for normal ECU communication based on signal exchange as well as for data transmission in case of software reprogramming. Cost aspects as well as construction⁴ and weight⁵ aspects are also reasons why an exclusively high performance network for software reprogramming purposes is not possible. Hence, other less cost intensive approaches are necessary to solve the current situation of increasing software despite the nearly constant vehicle bus system's bandwidth.

1.3.3 Automotive system complexity and compatibility

Innovation

During recent years a noticeable move from single innovations to system innovations is visible. In the past a typical ECU provided a single functionality and was used for a single assignment. Today more and more different functionalities are combined to a more complex function.

Figure 1.3-3 depicts this shift on a time line. According to that picture, single innovations will move into saturation but systems innovation will expand. M. Broy et al. mentioned that prospective innovation for infotainment systems, advanced driver assistance systems or safety systems will only be possible by distributed and connected functions [Bro11a].

⁴ The position of cable bundles as well as their cross section has to be considered for car body and chassis construction. The cable bundle cross selection could not be increased endlessly because of car body's sturdiness.

⁵ Additional cable results in a higher weight and this has an impact on the fuel consumption and vehicle emissions. This is opposed to the current vehicle manufacturer strategy with green technology, less fuel consumption and less/zero emissions.

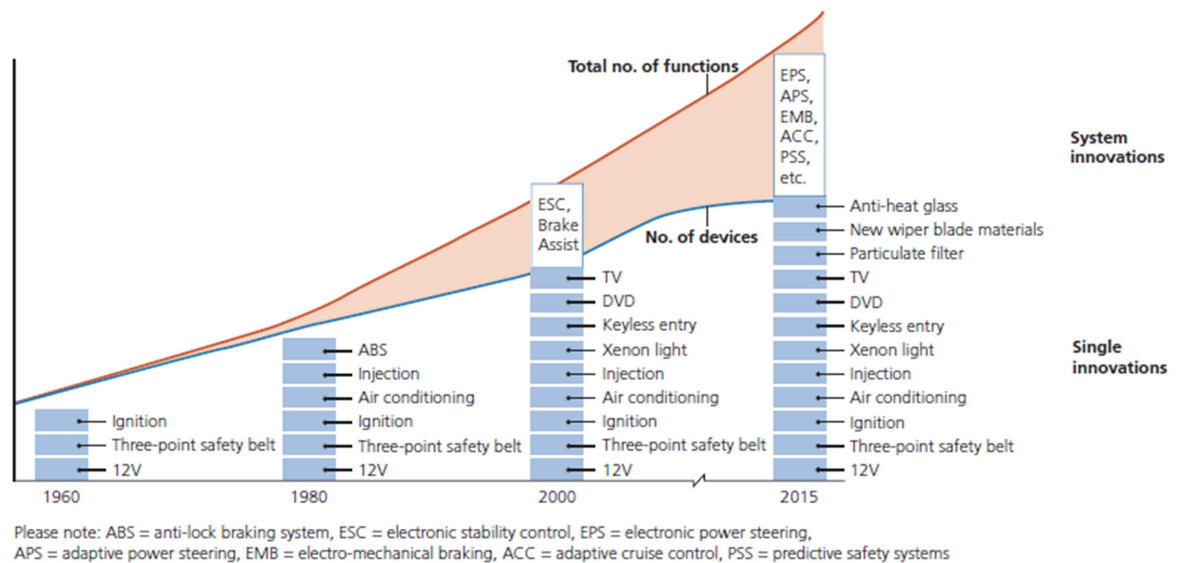


Figure 1.3-3: Shift from single to system innovation [Dan07]

J. Dannenberg et al. [Dan07] support the observation of this evolution by an example of the Mercedes-Benz PRE-SAFE⁶ system: “it links existing systems like crash sensors and ESP with seat controls, seatbelts and the sunroof, adding safety functions to existing components”.

The move to system innovations is also visible in new automotive terminologies like function-orientated development. This means that a system’s function is not tightly bound to a single ECU but multiple ECUs will be used to support a function based on factors such as memory availability and microcontroller load. This will have a significant impact on reprogramming times and strategies.

Compatibility

Due to the move to system innovations, a new challenge occurs: software compatibility. If an error occurs within such a distributed system it has to be guaranteed that a change of a single ECU’s software which is part of that system results in a compatible common system. Especially within the automotive industry, where vehicles have no fixed and stringent service intervals and a customer is free to visit a garage or not, new software could be programmed into an old vehicle. The functional distribution of systems and the complexity results in an increasing test demand to guarantee compatibility. Depending on the

⁶ Mercedes-Benz PRE-SAFE system was introduced in 2002 as the first anticipatory protection system. In case the system recognises that in a situation a crash is unavoidable, the system initiates some activities to optimise passenger’s situation if the crash occurs (e.g. close windows, tighten seat belts, move seats in an optimised position etc.).

involved ECUs the test permutation is so high, that it is not possible to test each possible combination. In that case testing of a well known release is a very efficient method but results in a release update where several ECUs have to be reprogrammed.

Table 1.3-1: Software release types

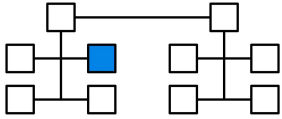
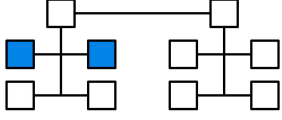
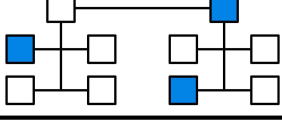
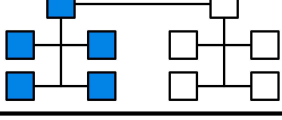
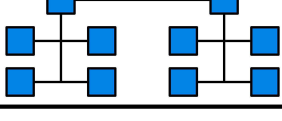
Release Type	Description	Usage
Single ECU 	Software release for a single ECU.	<ul style="list-style-type: none"> • Each ECU
Equal ECUs 	Software release for multiple but functional equal ECU.	<ul style="list-style-type: none"> • Door Control Unit • Seat Control Unit • etc.
System Release 	Software release for multiple/all ECUs which are part of a system.	<ul style="list-style-type: none"> • Illumination system • Driver assistance systems • etc.
Domain Release 	Software release for all ECUs which are part of the same domain.	<ul style="list-style-type: none"> • Power train • Body domain • Chassis domain • etc.
Vehicle Release 	Software release for all ECUs which are part of the vehicle.	<ul style="list-style-type: none"> • Car • Truck • etc.

Table 1.3-1 provides an overview about possible release types. As a consequence of less test demand within development the reprogramming amount to service a vehicle in a garage is increasing.

1.3.4 Automotive network aspects

Complexity

Since the first automotive systems were interconnected in the early 1980s vehicle communication networks have become more and more complex. Figure 1.3-4 depicts the network evolution. At the beginning only vehicle functionality was interconnected by bus systems. The main focus was to reduce the number of sensors and the reduction of discrete wires and cables between the different ECUs. The continuous introduction of electronic systems results in more complex networks. HMI, telematics and infotainment systems become more important and their communication demands have increased. The current development trend is on driver assistance systems. Surround camera systems, radar, infrared or ultrasonic systems continuously scanning the vehicle's environment. The

corresponding advanced driver assistance systems (e.g. park assistance⁷, lane departure warning⁸ [merced] etc.) process the generated data and support the driver. As a result more and more domains are interconnected.

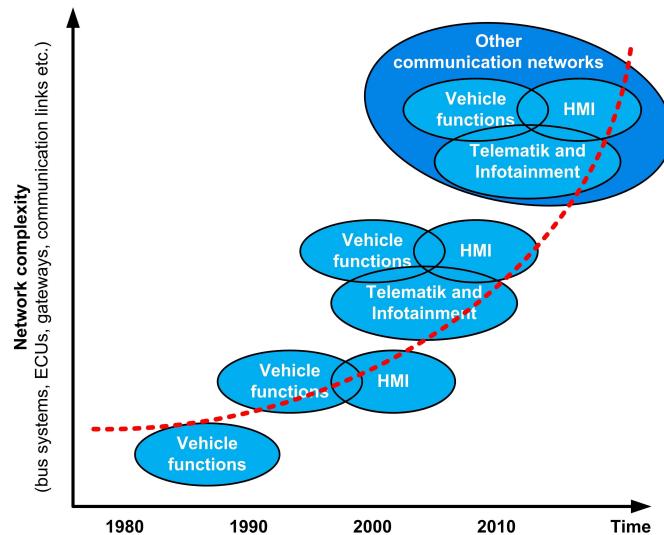


Figure 1.3-4: Increasing network complexity

While as of today only vehicle-internal domains are interconnected, the next technology steps will interconnect vehicles (i.e. car-to-car communication) as well as connect a vehicle into normal e-business networks (e.g. car hiring systems or smart charging communication for e-vehicles⁹). With focus on software reprogramming this development has some impacts:

- New bus systems are introduced to support functional communication requirements e.g. for new regulation systems. Those bus systems shall be used for software reprogramming communication too because cost aspects do not allow a second communication link exclusively for reprogramming.
- Depending on the domain architecture several gateways have to be passed for a software reprogramming communication link to establish communication to the most outlying ECU. The data routing time of these gateways results in time delays and increases the total programming time.
- Depending of the network connection type (heterogeneous network or homogeneous network) data routing is required on different layers according to the ISO/OSI

⁷ Park Assistance – autonomous car maneuvering from a traffic lane into a parking space.

⁸ Lane Departure Warning System – warns a driver when the vehicle begins to leave out of its lane.

⁹ E-vehicles: vehicles with electrical engine

reference model layered architecture [ISO7498-1]. Due to that fact the data routing results in different routing strategies which has impacts to required resources (e.g. Buffer) as well as the possible routing performance and routing execution times.

Costs

Cost aspects, as described above, are the reason why normal vehicle functional communication and reprogramming communication share the same vehicle network. As there was no functional requirement for a high speed and being more expensive network up to now high speed networks have not be placed inside automobiles. Since the amount of software has been increasing the resulting programming time is meeting the timing limits of network, thus testing and reprogramming requirements are starting to drive the network speed requirements and functional becoming less important. To evaluate a business case to incorporate design considerations to reduce reprogramming time requires accurate predictions of reprogramming times. The physical network infrastructure (e.g. the cable trees) within the vehicle is a foundation that forms the heart of the system. New bus systems and network architectures can not be introduced easily or at low cost within an older vehicle. In contrast more powerful ECUs' (microcontrollers with better performance, more interfaces, increased memory size etc.) will be developed during a vehicle's life cycle and could be also introduced into an older car. Hence, network architecture as well as the bus systems' performance and the gateways' routing performance paired with a long-term persistence have a deep impact for software reprogramming strategies.

1.3.5 Summary

Software reprogramming of automotive ECUs is a very important issue within a vehicle's life cycle. For bug fixing, software reprogramming is a method (sometimes the only low cost method) for the ECU supplier as well as for vehicle manufacturers within development, manufacturing and service/after-sales. The required time for software reprogramming is an important economical factor and depends on the software size to be reprogrammed as well as the communication link performance.

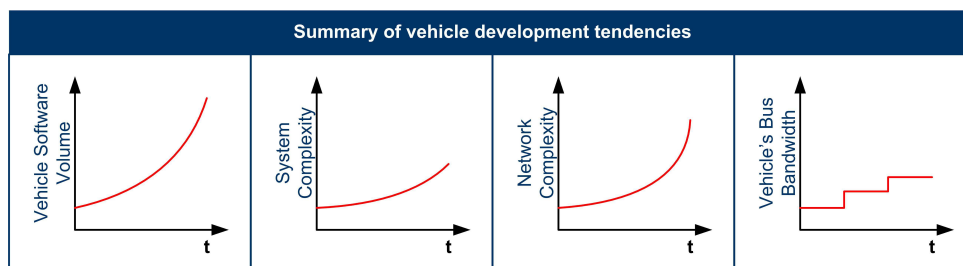


Figure 1.3-5: Vehicle development trends

Figure 1.3-5 depicts the current development tendencies. The software volume, system complexity (e.g. ECU's functionality dependencies) and network complexity (total number of bus systems, gateways and different bus system types) are significantly increased but the vehicle's network performance (bus bandwidth) is nearly constant. If the future trends discussed above become true, some challenges with focus on automotive software reprogramming occur:

- The demand for software reprogramming will still increase due to the increasing total amount of software within vehicles.
- Comprehensive reprogramming procedures for several ECUs will be necessary if complex distributed systems have to be reprogrammed to retain a compatible system.
- Software reprogramming time will still be increasing continuously due to increasing software size and the need to reprogram several ECUs.

Independent of future approaches to reduce the total complexity (e.g. software size, system dependencies and complexity) it might be necessary to accelerate the total automotive software reprogramming process.

1.4 Scope of the thesis

Within the automotive industry the ECU application software reprogramming process is quite complex. Figure 1.4-1 depicts the reprogramming process cycle divided into an off-board (non-vehicle) part and an onboard (vehicle) part.

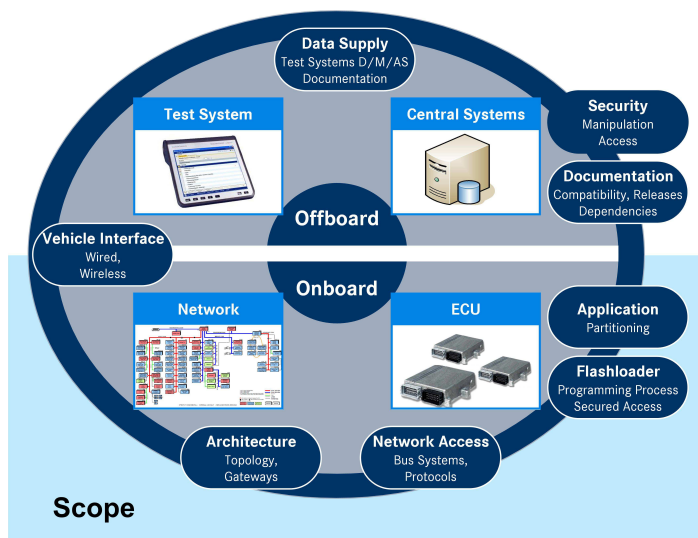


Figure 1.4-1: Software reprogramming process circle

The scope of the work reported in this thesis is on the on-board part of the global reprogramming process. ECU aspects (flashloader, application software, network access,

communication protocols etc.) and network aspects (architecture, topology etc.) are the focus of investigation.

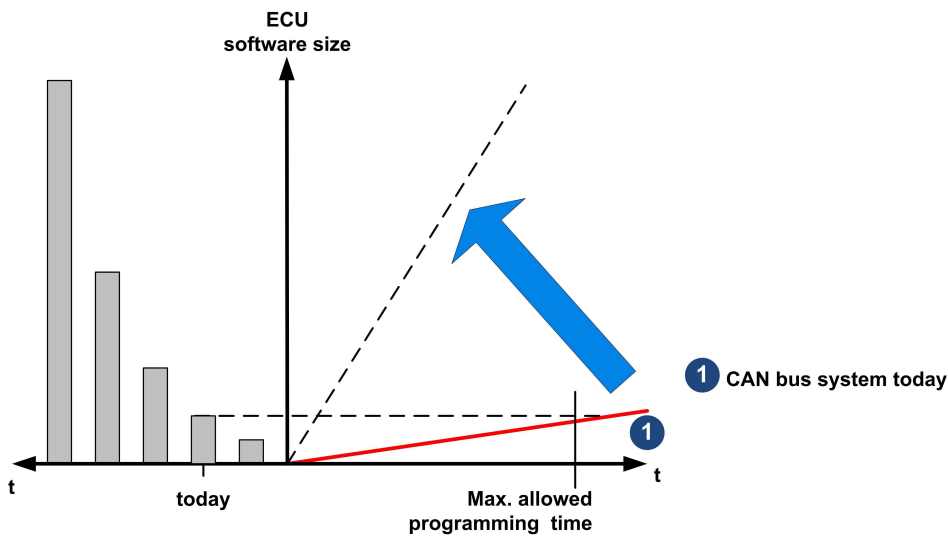


Figure 1.4-2: Software reprogramming time limitation

As depicted in figure 1.4-2 for future ECUs the given time limitation requirement to the maximum reprogramming time will not be fulfilled any longer without any optimisation. The research aims are:

- Quantify the reprogramming problem in today's vehicle architectures and communication standards and provide possible short term solution to the existing reprogramming issues.
- Quantify the impact that future emerging standards and technologies will have on reprogramming embedded ECUs and identify solutions to minimise their impact on reprogramming cost.

1.5 Organisation of the thesis

The thesis covers the onboard aspects for software reprogramming within automotive networks as defined in figure 1.4-1.

This chapter has identified the potential problem that the uptake of technology revolution, driven by the need for increased functionality has on reprogramming of complex embedded systems in the automobile industries. Not only does the industry have high production volumes, for an electronic product, it has long design times, production cycles and life time warrantee and thus long legacy costs. Problems caused today can have a very long term cost implication. Care must be taken to ensure future developments in car design consider all key life-time cost drivers.

Chapter 2 provides the key architectural and implementation background information to the software reprogramming process and the involved components for the current state of the art ECUs. It also identifies the key areas where acceleration is possible in reprogramming time for existing technologies i.e. improved data transfer rates and data compression strategies. Additionally all necessary terms and definitions are introduced.

In Chapter 3 a new concept of using double buffering in the reprogramming procedure of ECU is introduced. A model is generated to evaluate and quantify the improvement of this approach.

In chapter 4 the research results are discussed to accelerate data transfer by communication protocol optimisations which are currently in use for automotive systems' software reprogramming process (i.e. vehicles currently in production and legacy systems) and new technologies currently being designed into the new generation of car systems. For each of these communication protocols this chapter generates quantitative models that can be used to evaluate reprogramming data transfer performance.

Chapter 5 discusses and introduces new approaches to reduce the amount of data needed to be transmitted during software reprogramming. Quantitative models are again produced to complete the set of techniques needed calculate optimum reprogramming time for current technology solution available.

Chapter 6 introduces theoretical hardware modification that could be made to ECU designs to optimise and to speed up the reprogramming process.

In chapter 7 the impact of the network architecture on the reprogramming process is discussed and evaluated. Coupling of different networks and routing aspects within gateways are discussed.

Reprogramming several ECUs in parallel is a powerful approach to optimise reprogramming time. The required pre-conditions and a method to schedule the ECUs to be reprogrammed are discussed in chapter 8.

Chapter 9 provides an introduction to the newly awaited MRAM technology and identifies how it could be utilised to implement some of the reprogramming approaches presented in chapter 5.

Chapter 10 provides case studies where some of the discussed approaches are implemented within an ECU and quantitative models verified.

Chapter 11 summarises the work and provide an outlook for future methods and technologies.

2 Background

Content

2.1 Embedded Systems	19
2.2 Electronic Control Unit	21
2.2.1 Microcontroller	21
2.2.2 Memory	22
2.2.3 ECU Software Components Overview	24
2.2.3.1 Application Software	25
2.2.3.2 Flashloader	25
2.2.3.3 Boot Manager	27
2.3 Programming Control Unit (Test system)	28
2.4 Programming Sequence	29
2.5 Communication Stack	33
2.5.1 Field bus systems	35
2.5.2 Media Access Control Overview	36
2.5.3 Transport Layer Protocol	36
2.5.4 Application Protocols	38
2.6 Network	39
2.7 Summary	40

This chapter provides a short introduction into the software reprogramming process of embedded systems. The sub-components of an electronic control unit (ECU) which are relevant for the reprogramming process are explained and an introduction to the specific terms is given. The reprogramming sequence sub-clause shows the different steps of a

reprogramming process. The communication stack sub-clause introduces into automotive embedded system's communication and the relevant protocols.

2.1 Embedded Systems

Today monitoring, control and regulation problems in technical systems (e.g. medical instruments, machines, vehicles, aeroplanes) are mainly realised by microcontroller-supported embedded systems (refer to [Ren11]). "Embedded system" is more a general term than a well defined system definition. M. Barr characterised embedded systems as "a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function" [Bar07]. Today the term "electronic control unit" (ECU) is established as a synonym for an embedded system. It could become more complex if at least two or more ECUs are part of a distributed embedded system. In that case normally they are interconnected via field bus systems to exchange data.

Figure 2.1-1 depicts a simple embedded system with the relevant components.

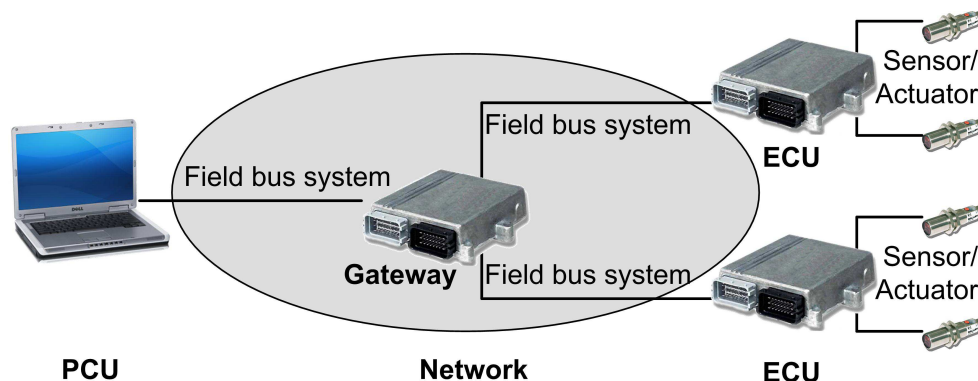


Figure 2.1-1: Embedded system components

A good example for a complex embedded system is a car where several systems allocated on several ECUs interact by exchanging data via a network. Such systems are also available on aircraft, trains, weapons and machines, and everywhere sensors and actors have to be controlled or regulation assignments are given.

Figure 2.1-2 and figure 2.1-3 depict the more complex embedded system of a Mercedes Benz model line 221 (S-Class) vehicle. Each coloured square represents an ECU. The coloured lines representing communication bus systems (field bus systems - refer to section 2.5.1). The number of ECUs within a vehicle is growing continuously from only 2 ECUs (Engine control system ("Motronic") and breaking system - ABS) in the 1980s up to nearly 80 ECUs within modern premium class vehicle.

The positions of all ECUs are depicted in figure 2.1-2.

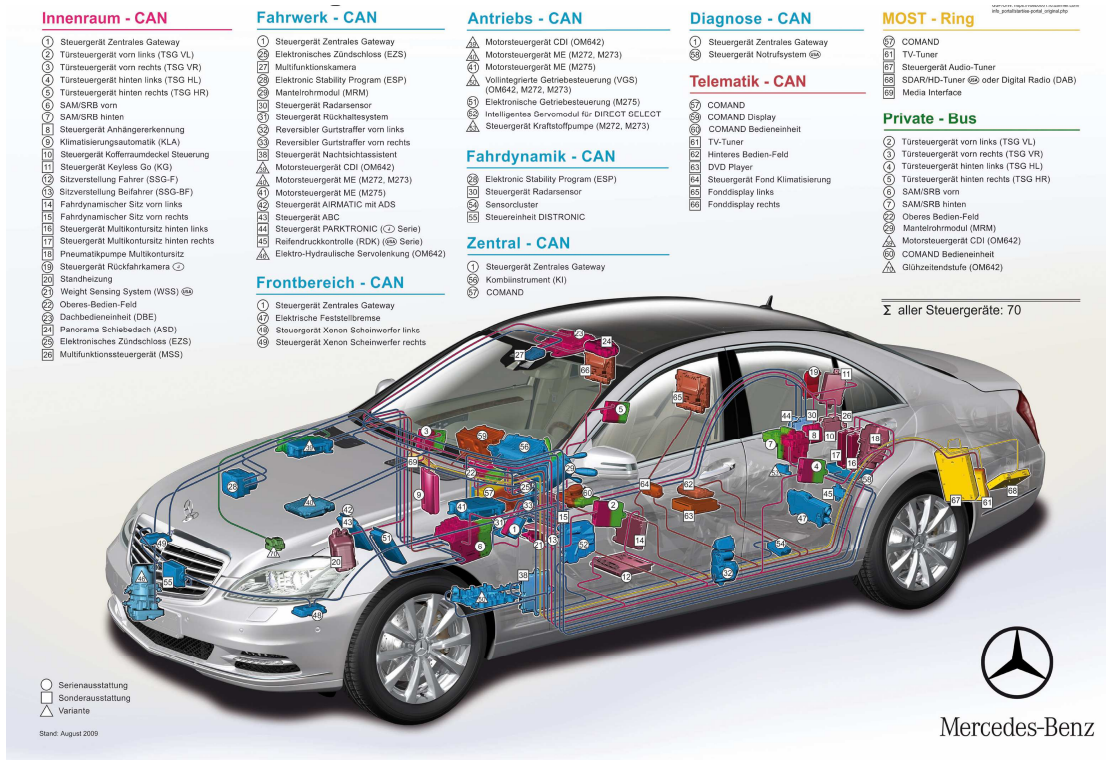


Figure 2.1-2: ECU network of a Mercedes-Benz Model line 221 (S-Class) [Mer09]

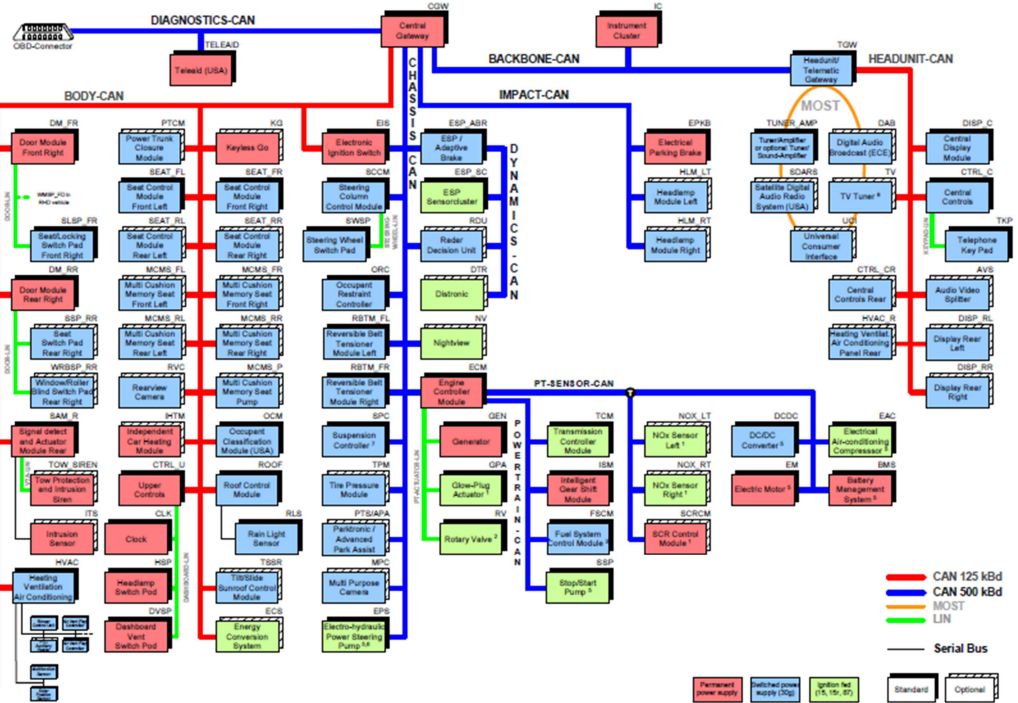


Figure 2.1-3: Mercedes-Benz Model Line 221 (S-Class) network architecture [Mer09-1]

For modern trains the architecture is similarly complex. In [Sie] the complex architecture of the SIEMENS Intercity Express 3 train (ICE-3) is given for different wagons. The list of

complex embedded systems could be continued at will. The nomenclature and special terms of embedded systems are described below.

2.2 Electronic Control Unit

As described above, the term “Electronic Control Unit” is established as a synonym for an embedded system too. An ECU is based on at least one microcontroller and is encapsulated in a closed package. The functionality is given by software. Hence, the same microcontroller with identical periphery could be used for different control assignments.

2.2.1 Microcontroller

In the 1960's Intel and later on Motorola developed the first microprocessor. Based on that technology the first single chip micro computer was developed in the 1970s. The structure of these initial microcontrollers¹⁰ is the base for many other microcontrollers today¹¹. In the past microcontrollers were derivatives of microprocessors developed for PCs. Today a lot of microcontroller families exist (e.g. Infineon TriCore, C166-family etc.) which were developed especially for embedded systems. The reasons for that development are very extensive because for embedded systems not only the pure computing performance has to be considered. Microcontrollers have to fulfil competing and some times opposing requirements: In addition to the computing performance, limiting current or energy consumption is very important as well as a lot of integrated interfaces exchange data.

Today, microcontrollers are no longer used for simple control and regulation purposes. Owing to the enormous technological progress high performance microcontrollers are available today to solve highly complex control and regulation assignments. Within the automotive area, for example, ECUs are used to control typical vehicle regulation systems e.g. engine, gearbox etc. During recent years, also additional driver assistance systems have been developed. High performance microcontrollers with complex periphery systems and sensors are the base for those systems. The computational power and memory sizes of embedded systems have been following continually expanding complexity of computer systems.

A detailed description of microprocessor architecture, internal processing, instruction sets etc. is given by J. L. Hennessy and D. A. Patterson [Hen03]. This basic knowledge is also

¹⁰ e.g. Intel 8048 or Motorola 6800

¹¹ The structure of the compiled and linked software for microcontrollers has been divided into two different types: Intel-Hex format [Int88] and Motorola S-Record format [Mot92].

true for microcontrollers. J. Schäuffele and T Zurawka published a technical introduction to microcontrollers [Scha10] as an introduction to the approaches of embedded software engineering.

With focus on the reprogramming of a microcontroller only the different communication interfaces, the central processing unit (CPU) and the memory have to be taken into account. All other internal or peripheral components like Input/Output ports, analogue/digital converters, watchdogs or other interfaces are not necessary in a reprogramming context.

2.2.2 Memory

Different memory types for different required functionality (store source code, data, volatile information etc.) have been established. Figure 2.2-1 depicts an overview of the different memory technologies used for embedded systems.

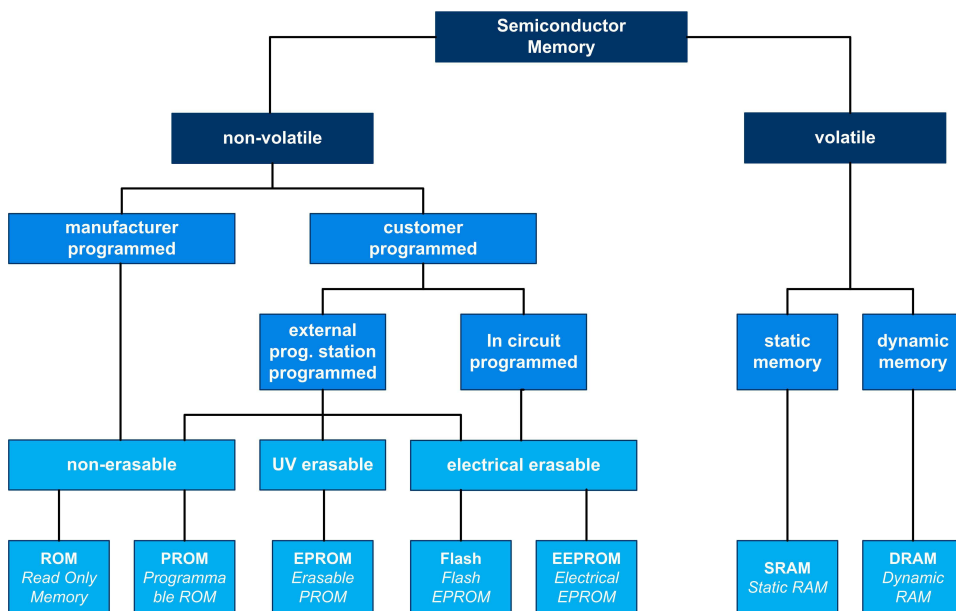


Figure 2.2-1: Memory Technologies Overview [Rei11]

Basically it is distinguished between volatile and non-volatile memory. RAM (Random Access Memory) is used for temporary stored information (e.g. source code variables etc.). Only in some special cases executable machine code is stored in RAM. Executable software is typically stored in non-volatile memories. ROM (Read Only Memory), PROM (Programmable Read Only Memory) and EPROM (Erasable Programmable Read Only Memory) have all together the disadvantage that these memory types are not electrically erasable. If application software must be changed the memory device has to be changed.

Flash Memory

“Flash Memory” is the current established memory technology for microcontrollers to store executable operation code. It is based on the metal oxide semiconductor field-effect transistor technology (MOS-FET) with floating gates and supports non-volatile storage of data. Flash memory could be electrically erased and reprogrammed. Within [Zim10-2] W. Zimmermann and R. Schmidgall described the abstract functionality of a Flash-Memory cell. The different conditions and the usage scenarios for the different memory types have been listed there, too (refer to table 2.2-1).

Table 2.2-1: ECU's semiconductor memory overview [Zim10-2]

Memory type	Programming	Erasing	Usage
ROM Read Only Memory	IC production time	no	Fix code
EPROM Erasable ROM	Only in dismantled state by the ECU supplier		Fix code Data (e.g. characteristic line)
Flash-ROM	In the ECU at any time > 100,000 times		Fix code and data
EEPROM Electrical Erasable ROM	In the ECU at any time > 100,000 times		Variable data with less update ratio (e.g. operation hour counter, status information etc.)
RAM Random Access Memory	Not necessary		Variable data Volatile after power-off

Compared to EEPROM (Electrical Erasable Programmable Read Only Memory), Flash memory is faster for both access types: read and write. Flash memory is organised in pages or blocks of several kilobyte. Unfortunately Flash memory is only erasable page by page (block by block) and until a memory page is erased or reprogrammed no instruction code read access is possible. Hence, the normal operation of the ECU has to be interrupted for the erase and program procedure.

Table 2.2-2: Physical programming performance

Microcontroller	Programming performance		Source
INFINEON TC1796	51.2 kByte/s	(256 Byte / 5 ms)	[TC1796]
NEC V850 Ex3	91 kByte/s	(4,096 Byte / 45 ms)	[V850-Ex3]
TMS470	128 kByte/s	(256 kByte / 2 s)	[TMS470]

Table 2.2-2 depicts an overview about typical physical reprogramming performance of current automotive microcontrollers.

For the integration of the erasing and programming process into the remaining ECU software, an independent software component is normally used inside the ECU. Compared to the remaining functions, this component encapsulates the programming process and provides defined interfaces. This component is referred to as the flashloader and will be described in section 2.2.3.2 in detail.

2.2.3 ECU Software Components Overview

With focus on embedded software reprogramming processes the software on an ECU has to be divided into three basic software components:

- a) boot manager
- b) flashloader
- c) application

All three parts are independent software components allocated in the Flash memory with different assignments. Figure 2.2-2 depicts an overview of those different software components.

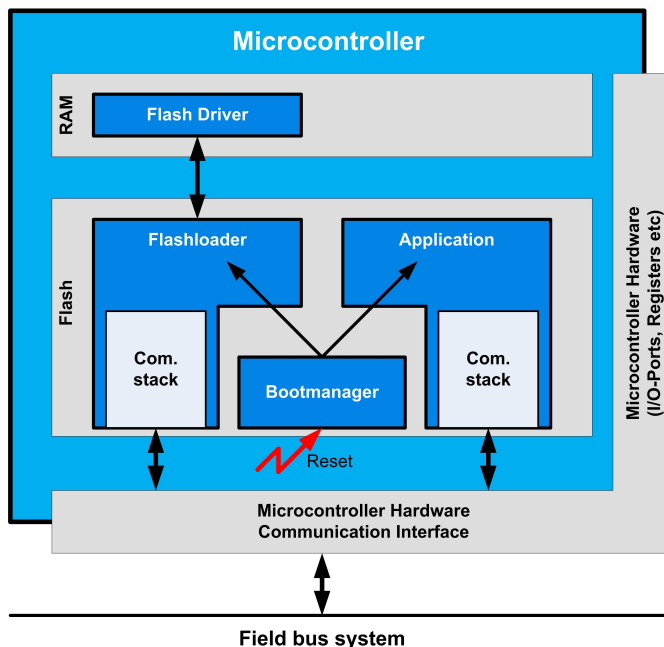


Figure 2.2-2: ECU Software Components Overview

The software component's individual functionalities with respect to a reprogramming process are described below.

2.2.3.1 Application Software

The application software (later referred as the application) implements the real control and measurement software to fulfil the basic assignment of that specific ECU, e.g. vehicle engine control or wash machines heating system etc. Hence, it implements the necessary drivers for all communication interfaces as well as the required communication protocol stacks, self diagnostic analysing systems, error storage system etc. This application software is responsible for the complete normal operation processing of an ECU. In many cases application software is divided into several partitions. A typical segmentation is the splitting into functional code and parameter sets. Additional segmentation could be possible too. It depends on the final assignment of the corresponding ECU and on the possibility to split the software into such logical groups. For example, if the ECU has any interaction with the user (HMI¹²) the fonts could be allocated in a separate partition.

If an application software update is necessary, e.g. in case of bug fixing or functionality upgrade the application software is partly (according to the configured partitions) or fully erased and reprogrammed.

2.2.3.2 Flashloader

The flashloader software (later referred as flashloader) is an independent software component that controls the reprogramming process. Typically it shares software modules with the other software components application or Boot Manager. Figure 2.2-3 depicts an abstract overview to the flashloader component. The flashloader has access to the complete memory area where the application (all partitions) is allocated and is able to erase and reprogram that memory area. A flashloader implements a complete communication protocol stack. In case a reprogramming process is initiated, the flashloader communicates with an external programming control unit (PCU). Especially for the automotive usage the German OEMs¹³ have standardized a flashloader within the HIS¹⁴ standardisation group. The document [HIS06-1] specifies the basic requirements for a flashloader based on the diagnostic protocol UDS¹⁵ (refer to section 2.5.4) and communication via CAN. W. Zimmermann and R. Schmidgall also described the requirements and

¹² HMI .. Human – Machine – Interface

¹³ OEM .. Original Equipment Manufacturer

¹⁴ HIS .. German: Hersteller Initiative Software (English: manufacturer's initiative for software)

¹⁵ UDS – Unified Diagnostic Services (refer to section 2.5.4). UDS is the current standard diagnostic protocol for the automotive industry.

implementation approaches of flashloader in context of the ECU's reprogramming process [Zim10-3] in a more abstract view independent of the used field bus system.

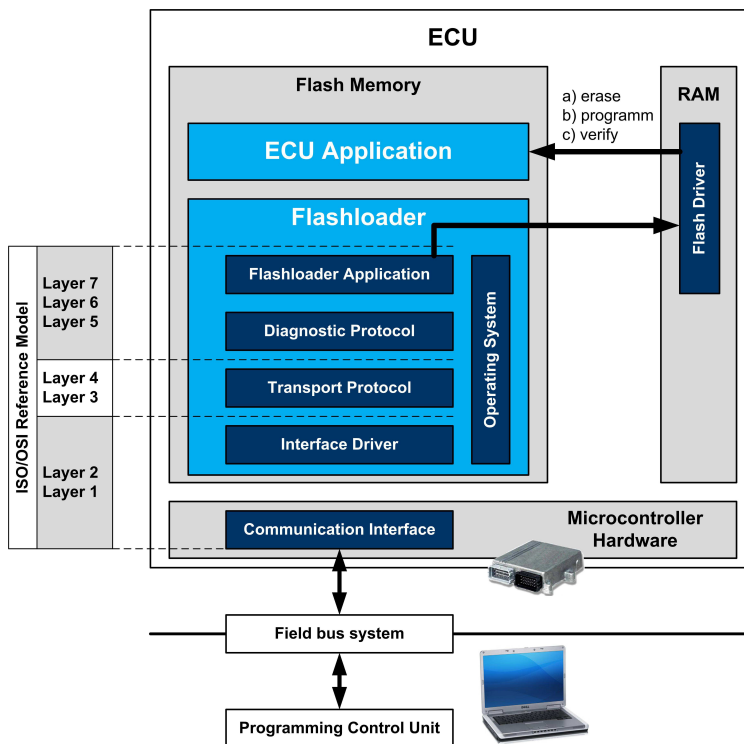


Figure 2.2-3: Overview flashloader component

The main assignments of a flashloader are:

- a) Establishing and managing a communication connection to an external Programming Control Unit.
- b) Control communication access to the ECU.
- c) Erase the addressed memory segments
- d) Program the new software parts physically.
- e) Verification of the programmed software parts (programming, integrity).
- f) Check compatibility from hardware and software or, if more independent software partitions exist, check the compatibility of the different software modules.
- g) Documentation of a reprogramming process
- h) Error handling if reprogramming failed

The different requirements in context of the consecutive steps during a reprogramming process are described in section 2.4 where the typical reprogramming sequence is discussed.

The flash driver is a part of the flashloader's software. It is required because it is not possible to execute code from part of the flash memory section and erase another section

concurrently. Hence, the driver has to be copied into RAM and executed there. Because of the flash memory development and the fact that on-chip flash memory enlarges very fast, many microcontroller manufacturers are able to provide on-chip flash memory with at least two memory banks (physically divided memory blocks). Here it is possible to execute code from one bank and erase the memory partitions of another bank. In consequence the copy process of executable memory driver code to RAM is no longer necessary. Today the flashloader's performance has a significant impact on the total reprogramming time. Flashloader implementation aspects are part of the research and the results will be discussed within this document.

2.2.3.3 Boot Manager

The term boot manager was introduced by the HIS sub-working group for software reprogramming process [HIS06-2].

After Power-On the microcontroller processes the start up sequence (initializing of system registers, PLL¹⁶ and VCO¹⁷ settings etc.). After that initialisation the system has to distinguish whether a valid application software is available to execute or the flashloader software has to be executed. This distinction is processed by the boot manager software. The boot manager is the first active software component after a system reset.

The boot manager has to distinguish whether application software or flash loader software shall be started and executed next. Typically the boot manager starts the application software. If no application software is available, the boot manager starts the flash loader software and the system waits in flashloader's idle mode until an external Programming Control Unit (refer to section 2.3) initiates a reprogramming process. The decision whether application is executable depends on the result of some start-up checks:

- a) Is application software available?
- b) Is the application software correct (not corrupted)?
- c) Is the application software compatible to the hardware?
- d) If the software contains more than one separate reprogrammable module (e.g. regulation algorithm and parameter set etc.) are these different modules compatible to each other?

¹⁶ PLL .. Phase-locked Loop

¹⁷ VCO .. Voltage-controlled Oscillator

Depending on the basic reprogramming strategy and the implemented protocol the boot manager has also to distinguish if a reprogramming request is available. In that case the flashloader has to be executed although valid application software is available.

Of course, the boot manager is an important module for an ECU but with focus on reprogramming application software this component is not of essential interest.

2.3 Programming Control Unit (Test system)

The programming control unit (PCU) manages the reprogramming process. The PCU has access to the data that shall be programmed and knows the reprogramming sequence for the ECU.

PCU is the generic, abstract term for all the different applications, tools and systems that are available in context of microcontroller's software reprogramming. The spread is from simple "download applications" integrated in embedded software development suites up to more extensive test systems¹⁸ for industrial ECU manufacturing and after sales support. Common to all PCUs is the necessity to implement communication interfaces on either the microcontroller or the ECU. The available interfaces depend on the current position within microcontroller's or ECU's life cycle. In the early development phase for example communication via JTAG¹⁹ interface might be possible whereas in a post-development phase (e.g. production or service) no access to that interface is possible. Here in many cases access to the ECU is only possible via the (normal) application communication interfaces.

Within the automotive area, PCUs are typically integrated components of more complex test systems. The Association for Standardisation of Automation and Measurement Systems (ASAM) has standardised those test systems and provides many documents for the different layers [ASAM]. In [Zim10-8] W. Zimmermann and R. Schmidgall give a short overview about the standardised test system, the different interfaces and the standardised exchange data formats. In the reprogramming context of this document, the PCU is defined as an abstract data source that communicates via field bus systems and implements the corresponding communication protocols (refer to chapter 3). The PCU

¹⁸ Other terms for test system in publications or standards are: tester, diagnostic tester, external test tool, diagnostic test tool, test equipment, diagnostic test system

¹⁹ JTAG .. Joint Test Action Group describes the IEEE-1149.1 standard that collects several methods for testing and debugging of electrical hardware directly within the circuit. A sub-method is to program embedded memory by direct access to the memory cells.

implementation, as well as the data exchange formats and container, are not in the scope of this work.

2.4 Programming Sequence

The reprogramming sequence specifies the consecutive steps that are necessary to program the ECU's memory. From an abstract point of view, the reprogramming process for ECUs is always the same. A typical reprogramming sequence for embedded systems based on microcontrollers could be divided into three abstract sub-sequences:

1. Pre-programming sequence
2. Major programming sequence
3. Post-programming sequence

These sequences will differ depending on several environment conditions like the reprogramming scenarios (e.g. initial programming, reprogramming), reprogramming places (e.g. direct access to the microcontroller, direct link between PCU and ECU, reprogramming via network etc.). Pre- and post-programming sequences are necessary to prepare a communication network for the reprogramming process. The major programming sequence implements the physical programming process as well as verification and administration data processing.

Major programming sequence

Figure 2.4-1 depicts the major programming sequence in an abstract view.

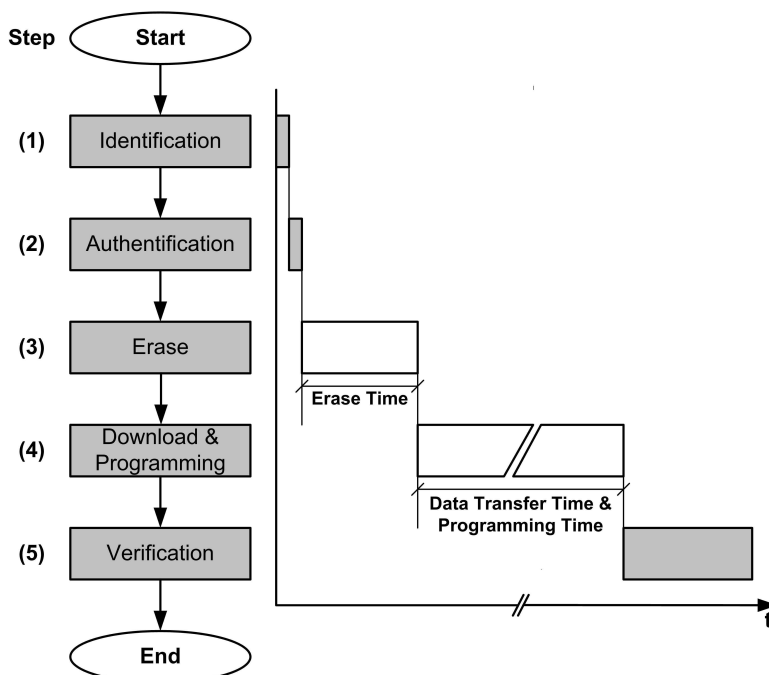


Figure 2.4-1: Abstract major programming sequence

Initially the PCU has to identify the ECU (step 1) and gets access for reprogramming process execution (step 2). ECUs Flash memory must be erased previously before it can be reprogramming (step 3). The PCU transfers data to the ECU where they are physically programmed into the Flash memory (step4). Finally the physical programming process is verified (step 5) before the reprogramming sequence has finished.

Due to the reprogramming sequence according to figure 2.4-1 two approaches are possible to reduce the total software reprogramming process time significantly:

- 1) Accelerate data transfer from PCU to ECU
- 2) Reduce data size to be transferred from PCU to ECU.

The other stages within that sequence are hardware dependant (erase process, verification e.g. CRC calculation) and based on technology used or require only a small part of the overall reprogramming process time (Identification, authentication).

Major programming sequence mapped to UDS protocol

There have been efforts for many years by the ISO and the HIS to standardise the reprogramming sequence for the automotive industry. The sequence is based on the diagnostic protocol²⁰ and specifies the ordering of the required diagnostic services. In fact, the vehicle manufacturers and suppliers vary significantly in terms of the used diagnostic services. An attempt is made in [HIS06-2] to standardise the procedures although a couple of steps are optional so that different options are still possible. Within [ISO14229] (*Unified diagnostic services – UDS* – refer to section 2.5.4) the reprogramming sequence shall now be standardised in a common international standard.

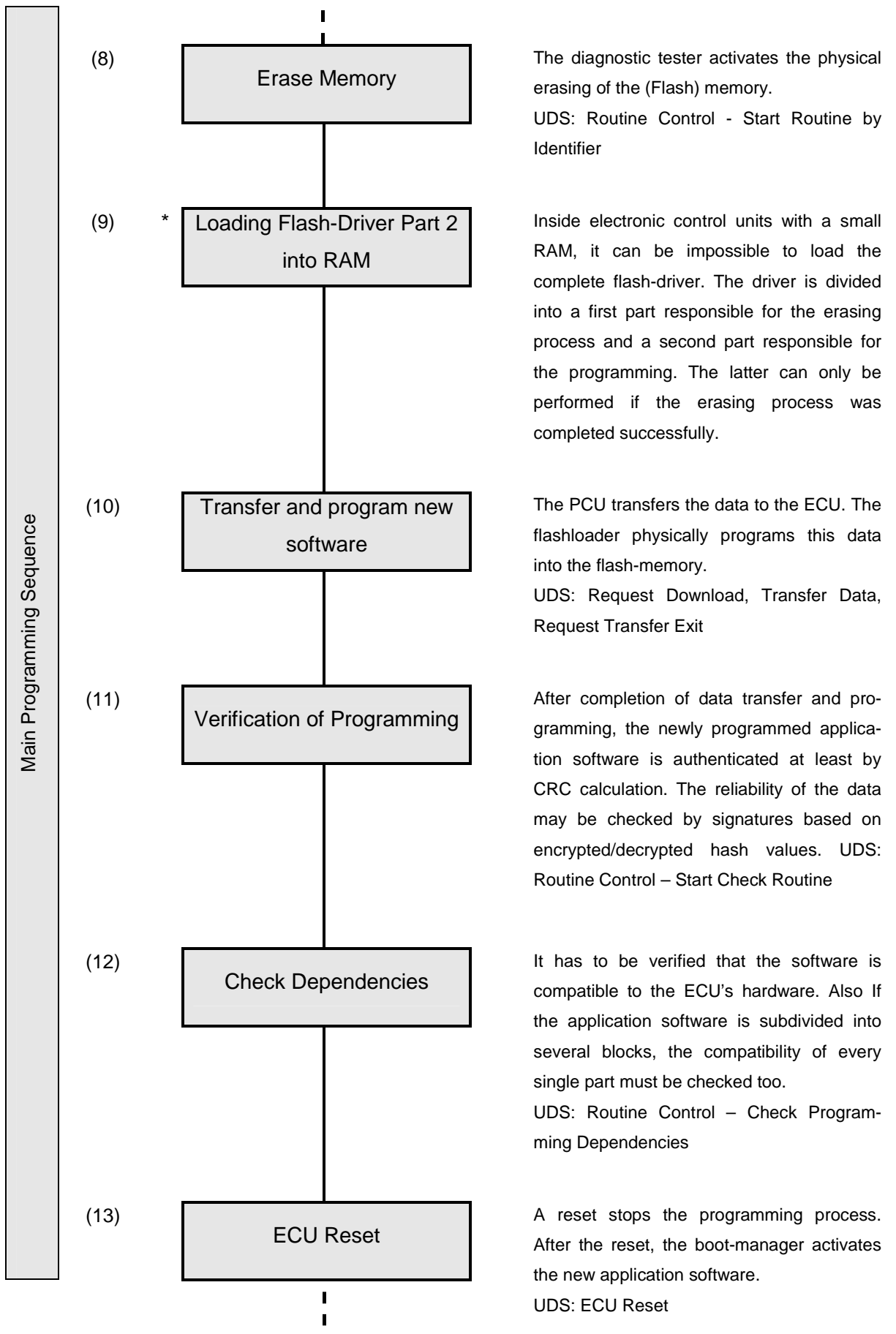
[Zim10-3] explains the reprogramming sequence based on [HIS06-2]. Table 2.4-1 depicts the reprogramming sequence in a more generic manner. The text column explains the abstract steps and maps the abstract requirement to the corresponding diagnostic services as defined in UDS [ISO14229] and [HIS06-2].

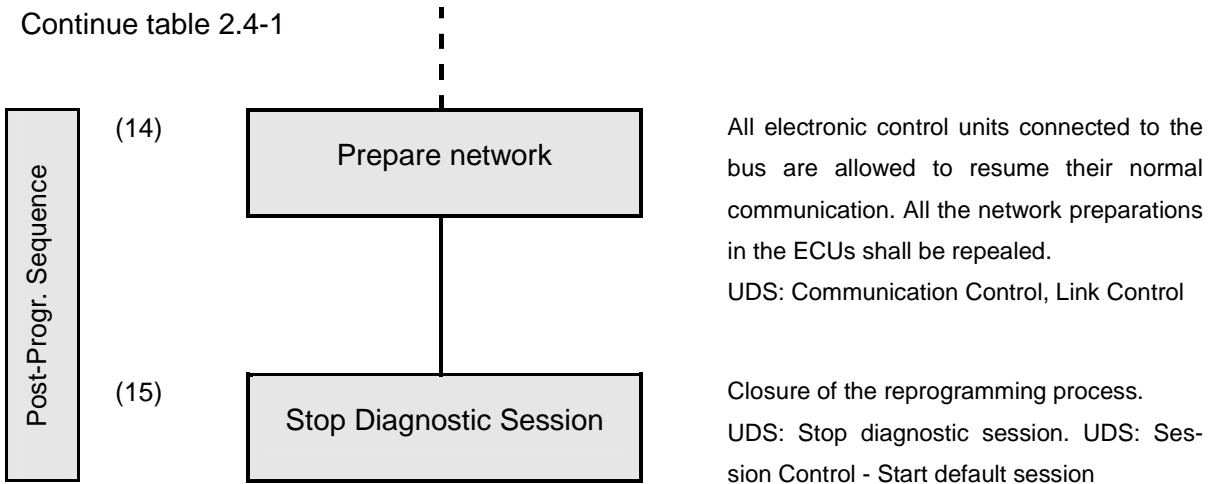
²⁰ Several diagnostic protocols have been standardised in the past. The currently most significant protocol is standardised in the document of ISO 14229 – Unified Diagnostic Services (UDS) [ISO14229].

Table 2.4-1: Software Re-Programming Process according to [HIS06-2] and [Zim10-3]

Step	Sequence	Description	
Pre-Programming Sequence	(*).. conditional steps		
	(1)	Start Pre-Progr. Sequence	Starting the Pre-Programming sequence for the preparation of the programming process. UDS: Diagnostic Session Control
	(2)	* ECU Identification	Readout of ECU identification to identify the hardware and the current software version. UDS: Read Data by Identifier
	(3)	* Check Pre-Conditions	Checking if all device-specific preconditions necessary for the programming are fulfilled (e.g. automotive area: engine off). UDS: Routine Control - Check Programming Pre Condition
(4)	* Network Preparation	Prepare all ECUs within the network for a reprogramming process (e.g. disable the normal communication to gain full bandwidth for reprogramming communication, deactivation of communication timeout monitoring etc.). UDS: Communication Control, Link Control	
Major Programming Sequence	(5)	Start Main Programming Sequence	Switch to ECU's flashloader. UDS: Diagnostic Session Control
	(6)	Authentication	Authentication of the tester to the ECU. The access is denied if the authentication fails. UDS: Security Access – Get Seed, Send Key
	(7)	Loading Flash-Driver Part 1 into RAM	If the flash-driver is a fixed component of the flash-loader, the latter copies the flash-driver into the RAM in due time. Otherwise, the diagnostic tester carries out this task. UDS: Request Download, Transfer Data, Request Transfer Exit
		--- --- ---	

Continue table 2.4-1





Of course, the sequence as mentioned above is a very generic approach. But the conditional steps (marked by a ‘*’) provide the possibility to process these sequences for each microcontroller on many different scenarios during the life cycle. Independent of the final sequence and common to all reprogramming scenarios is the requirement to reprogram an ECU as fast as possible.

2.5 Communication Stack

In distributed embedded systems the different ECU applications have the need to exchange data. According to the Open System Interconnection model (OSI model) standardised in [ISO7498-1] embedded system’s communication is mapped to that model, too. As depicted in table 2.5-1, a communication system is sub-divided into different layers.

Table 2.5-1: OSI reference model

Layer		Description
7	Application	Network process to application
6	Presentation	Data representation, encryption and decryption, convert machine dependent data to machine independent data
5	Session	Inter-host communication
4	Transport	End-to-end connections, segmentation
3	Network	logical addressing, routing, flow control
2	Data Link	Bus access, physical addressing, bit error detection etc.
1	Physical	Signal and binary transmission, bit coding

Equal layers within different ECUs are communicating via protocols. Besides the payload additional layer specific header and trailer information are also transmitted. In many cases

the header is also called “*Protocol Control Information*” (*PCI*). The trailer implements control information, e.g. check sums etc. In some cases, no protocol trailer is defined.

Each layer instance (N) provides services to the layer instance above (N+1) and below (N-1) or uses services from the layer instance above or below. Figure 2.5-1 depicts an abstract view of the internal communication structure of a protocol stack.

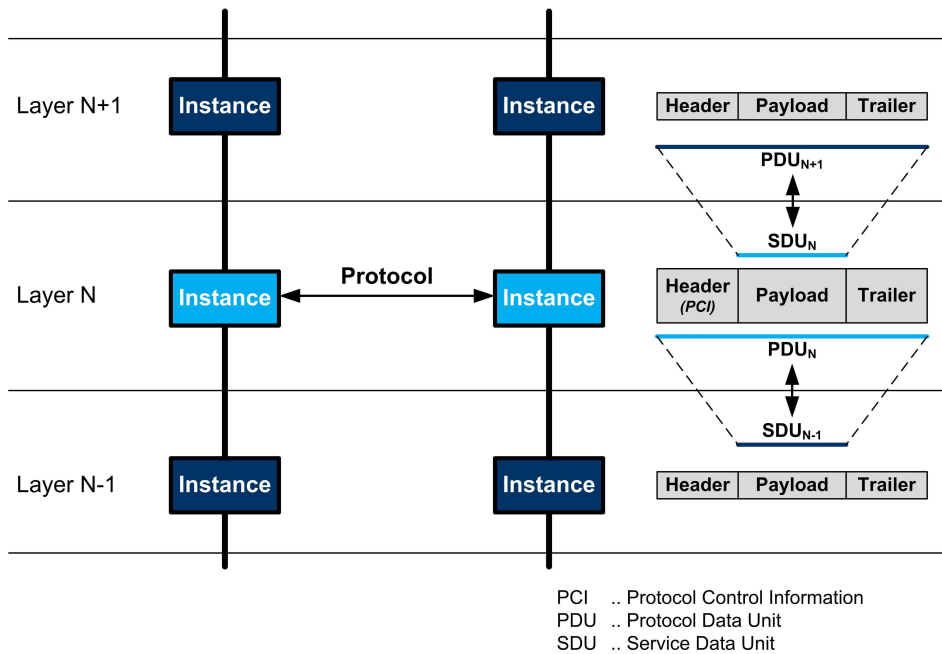


Figure 2.5-1: Communication structure within a protocol stack

Usually embedded software has strict resource restrictions because memory resources and microcontroller performance are limited in contradiction to the PC world, whereas from an embedded system’s point of view memory resources and processor speed are unlimited. Some reasons for those restrictions especially within the automotive area were presented in chapter 1. As a result of these resource restrictions within the embedded world and their field bus systems (refer to section 2.5.1) it might be possible that some layers are either combined (layer 3 – *network layer* and layer 4 – *transport layer*) or not available (layer 6 – *presentation layer*).

Many protocols on the different layers within the automotive area are standardised within ISO²¹ or SAE²².

²¹ ISO – International Standardisation Organisation

²² SAE - Society of Automotive Engineers

2.5.1 Field bus systems

Field bus is the name of a family of industrial computer network protocols. According to the ISO/OSI reference model of table 2.5-1, the protocols typically specify layer 1 and 2. Field buses connect field components like sensors and actors and ECUs with the purpose of exchanging data. The first generation of field bus systems was developed in the 1980s²³. Since 1999 field bus systems are standardised within the specification IEC 61158 - Digital data communication for measurement and control – Field bus for use in industrial control systems [IEC61158]. The different usage scenarios of field busses within the different business areas provide the opportunity for competing field bus technologies. G. Schnell and B. Wiedemann [Schn08] provide an overview about field bus systems within automation systems, while W. Zimmermann and R. Schmidgall do it for automotive systems [Zim10-1].

Table 2.5-2: Field bus systems in automotive area

Name	Bus access method	Bandwidth	Payload
K-Line	-	10,4 kBit/s	1..255 Byte
LIN	Master-Slave	1 .. 20 kBit/s	1..8 Byte
CAN	CSMA/CR ²⁴	1 MBit/s	0..8 Byte
TTCAN	TDMA ²⁵	1 MBit/s	0..8 Byte
FlexRay	TDMA	10 MBit/s	0..254 Byte
Byteflight ²⁶	TDMA	10 MBit/s	0..12 Byte

Today different field bus systems are established within the different business areas. Table 2.5-2 provides an overview of the currently most used field bus systems within the automotive area. With focus on software reprogramming acceleration, different approaches for CAN and FlexRay will be discussed in detail in chapter 4. It is possible to adapt the methods to other bus systems based on equal bus access strategies.

²³ ISA S50.02 standard

²⁴ Carrier Sense Multiple Access with Collision Resolution (refer to chapter 4)

²⁵ Time Division Multiple Access (refer to chapter 4)

²⁶ Developed by the BMW AG

2.5.2 Media Access Control Overview

Master-Slave bus access is a general term for systems where a node has unidirectional control over one or more other devices. Only the master initiates a communication link. The slave nodes are not allowed to communicate without a master request. *Local Interconnection Network (LIN)* is a typical *Master/Slave* system for embedded system's communication.

Carrier Sense Multiple Access (CSMA) is a general term for asynchronous (event based) bus access. A node verifies the idle state (absence of other traffic) of a shared transmission medium before transmission is initiated. Data transmissions of a node are generally received by all other nodes connected to the medium. A. Tanenbaum and D. Wetherall provide a detailed introduction into the different media access control (MAC) methods [Tan10].

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is a modification of the above described CSMA method. If a currently transmitting node detects another transmission, it stops transmitting the frame and then waits for a random time interval before trying to send again.

Carrier Sense Multiple Access with Collision Resolution (CSMA/CR) is a second modification of the above described CSMA method. The method is used to provide a deterministic communication system based on CSMA. If a collision is detected, a priority definition forces the transmission of the higher priority note or PDU (*frame*). *Controller Area Network (CAN)* is one of the most popular bus systems based on that media access control.

The *Time Division Multiple Access (TDMA)* method allows several nodes to share a bus system by dividing the channel bandwidth into different time slots. Each has an exclusive transmission access to a time slot defined within a global schedule. *Time Triggered Controller Area Network (TTCAN)* and FlexRay are two bus systems based on that media access control.

2.5.3 Transport Layer Protocol

As described in table 2.5-2 the payload of field bus systems is limited. But for the purpose of reprogramming, it is necessary to transfer data in larger segments than the maximum payload of the physical protocol data unit (PDU). Hence, a protocol mechanism is necessary to adapt large data strings to physical layer protocol's PDU. According to the ISO standardised OSI reference model this mechanism is implemented on layer 4 [ISO7498-1a]. Figure 2.5-2 depicts an overview.

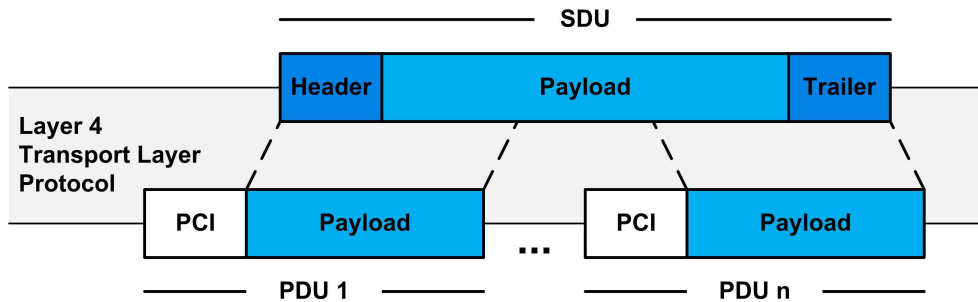


Figure 2.5-2: Protocol Stack Overview – Transport Layer

As mentioned above, not all layers of the basic OSI reference model are defined and/or implemented within embedded system's communication protocol stacks. For many field bus systems, layer 3 (Network) and 4 (Transport) are combined. The title of [ISO15765-2] "*Road vehicles – Diagnostics on Controller Area Networks (CAN) – Part 2: Network layer services*" misleadingly suggests the specification of network layer functionality. Nevertheless, a method for segmented data transmission via CAN (according to ISO/OSI reference model done on layer 4) is specified too. [ISO10681-2] specifies "*Communication Layer services*" for FlexRay. Address handling (ISO/OSI reference model - layer 3) and transport protocol handling (ISO/OSI reference model - layer 4) is defined within the same specification. Due to the fact that the terms used within standards are ambiguous where layer 4 and layer 3 are combined for embedded systems, the term "Transport Protocol" will be used within the further document.

Table 2.5-3: Automotive related transport protocol specifications

Name	Transport Protocol Specification
LIN	Parts of ISO 15765-2
CAN	ISO 15765-2 SAE J1939/21
FlexRay	ISO 10681-2 AUTOSAR 2.1 – FrTp
Ethernet	ISO 13400

It has to be distinguish between ISO standardised protocols and proprietary protocols. Proprietary protocols are defined if no ISO standard is available, or also if the physical medium is not standardised in ISO or SAE. Especially the automotive industry has a big interest to standardise such protocols. Hence, most transport layer protocols for automotive usage are standardised or standardizing activities have started. Table 2.5-3 depicts an overview of the different automotive related standardised protocols.

W Zimmermann and R. Schmidgall explain the different protocol mechanisms of the transport protocols actually used within the automotive area in detail in [Zim10-5]. The transport protocol topic will also be discussed in chapter 4.

The basic functionalities of such a transport protocol are:

- a) Segmentation of large service data units (SDU) into several protocol data units (PDU)
- b) Reassembling of received PDUs to an SDU
- c) Data flow control management
- d) Timing control of the established data transfer link (timeout management)

Within software reprogramming processing large data blocks are transferred from the PCU to the different ECUs. Therefore, transport protocols are necessary to segment these large data blocks to fragments with size of field bus system's payload. For some of the automotive relevant field bus systems the transport protocols are typically specified within the ISO (refer to table 2.5-3). Transport layer configuration has a deep impact on the overall communication speed and the data transfer bandwidth. The impact of those different configuration possibilities will be discussed in chapter 4.

2.5.4 Application Protocols

Today several protocols exist to reprogram ECU's software. Within the automotive industry software reprogramming in production and service is a part of diagnostics. In recent years diagnostic communication was strictly standardised within the ISO. Based on OEM specific, manufacturer specific and proprietary protocol implementations *Key Word Protocol 2000 (KWP2000)* was standardised in [ISO14230-3] in 1999. The next generation of automotive diagnostic protocol is *Unified Diagnostic Services (UDS)* and standardised in [ISO14229].

Some ECUs have to be adapted to the environment, e.g. engine control units to the engine or transmission control module to the engine or the gearbox. Within the automotive industry the protocols for measurement and calibration are standardised by the ASAM²⁷ standardisation group. In [Zim10-8] a detailed introduction to the *Universal Measurement and Calibration Protocol (XCP)* and the older *CAN Calibration Protocol (CCP)* is given by W. Zimmermann and R. Schmidgall. During the adaptation and calibration process, it could be necessary to reprogram parts of the memory, e.g. with new values for characteristic curves etc. If the system is produced or serviced within the after sales

²⁷ ASAM .. Association for Standardisation of Automation and Measuring Systems [ASAM].

market, the calibration or measurement protocol is no longer needed. In contrast, diagnostic is required during the ECU's complete life cycle. Hence, the focus for reprogramming process acceleration is on diagnostic protocols. It is also possible to reprogram software with calibration protocols, but this is not their initial intention.

2.6 Network

“The networks used in distributed systems are built from a variety of transmission media, including wire, cable, fibre and wireless channels; hardware devices, including routers, switches, bridges, hubs, repeaters and network interfaces; and software components, including protocol stacks, communication handlers and drivers” [Cou01-1].

The definition of James Coulouris et al. is also correct for automotive embedded systems. However, wireless transmission media is currently only used to interconnect customer's consumer devices (e.g. mobile connection via blue tooth) and the number of different network nodes is reduced to gateways. But, in the case of software reprogramming it is also true that “the resulting (...) performance available to distributed system (...) is affected by all of these” [Cou01-1].

In section 2.5 the different field bus systems were described which are currently relevant for the automotive industry. A network is the combination of at least two field bus systems. The network and its data transfer rate have a significant impact on the overall reprogramming performance. Nevertheless, optimisation of networks for the reprogramming process has not been the focus point during recent years. As described in chapter 1, cost aspects have the main priority. On the other hand, there was no pressure to optimise the automotive networks because software reprogramming was not a problem. Hence, the focus, in terms network architecture aspects, was on the ECU's application software's communication.

For the network design today, the reprogramming issue has to be taken into account. The challenges of diagnostic communication within modern vehicle networks have increased and network configuration has become more complex (refer to appendix C) [Sch11-1]. Tool supported network analysis is necessary, but tools for diagnostic specific protocol analysis are currently not available (refer to appendix D) [Sch11]. Therefore, network architecture and design aspects also need to be analysed within this document.

2.7 Summary

This chapter has provided background information to embedded systems and the involved components for the software reprogramming process where a PCU communicates with an ECU's flashloader via a communication network. All above discussed aspects contribute to a system model for embedded systems' reprogramming process as depicted in figure 2.7-1.

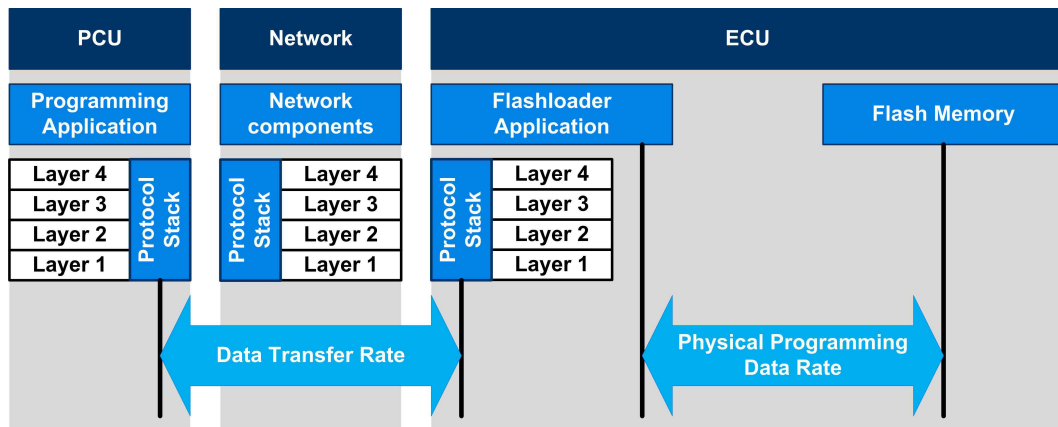


Figure 2.7-1: System model for embedded system's software reprogramming

Current ECUs based on Flash memory technology require a special reprogramming sequence because the Flash memory allows not reprogramming a memory cell without previously erasing. This is a strong restriction to possible reprogramming strategies.

Due to the reprogramming sequence of section 2.4, process acceleration is possible if

- a) data transfer is accelerated or
- b) data size to be transferred is reduced.

The data transfer rate as an indication of communication performance depends on:

- a) The used bus systems,
- b) The upper layers communication protocols,
- c) The hardware performance and
- d) The performance of network coupling elements like gateways.

Data size reduction will accelerate the data transfer process because less data has to be transferred. Different approaches are possible to reduce the total amount of transferred data.

3 Double buffered data transfer

Content

3.1 Reprogramming Protocol	42
3.2 Double buffered data transfer	42
3.3 Method's utilisation	49
3.3.1 Mapping to Diagnostic Protocol ISO-14229 – UDS	49
3.3.2 Mapping to other application protocols.....	51
3.3.3 Mapping to multi controller systems	51
3.4 Conclusion.....	52

This chapter is intended to discuss an approach to accelerate the data transfer between an external programming control unit (PCU) and an electronic control unit (ECU).

According to ISO/OSI reference model (refer to section 2.5), communication protocols on layer 5 to layer 7 are independent of the underlying bus system (represented by layer 1 to layer 4 protocols). Hence, optimisations on layer 5 to 7 are generic approaches, usable for data transfer via all field bus systems (refer to section 2.5). Nevertheless, the indisputable thesis for a communication system is given as:

The upper limit for the communication performance on a physical layer is 100% bus load.

Hence, upper layer protocols shall reduce protocol delays that finally results in delays on the physical bus system and therefore reduce data transfer rate.

Based on the initial thesis and with focus on automotive communication protocols the following issues are discussed:

- a) The theoretically maximum of the protocol's data transfer rate
- b) The influencing parameters and restrictions to reach the maximum value.

3.1 Reprogramming Protocol

Reprogramming protocols control the reprogramming process. Typically they are mapped to layer 7 within the ISO/OSI reference model nomenclature. Within the automotive area typical protocols to control the reprogramming process are diagnostic protocols like “Key Word Protocol 2000” (KWP2000) as defined in [ISO14230-3] or “Unified Diagnostic Services” (UDS) as defined in [ISO14229-1]. Figure 3.1-1 depicts a system overview.

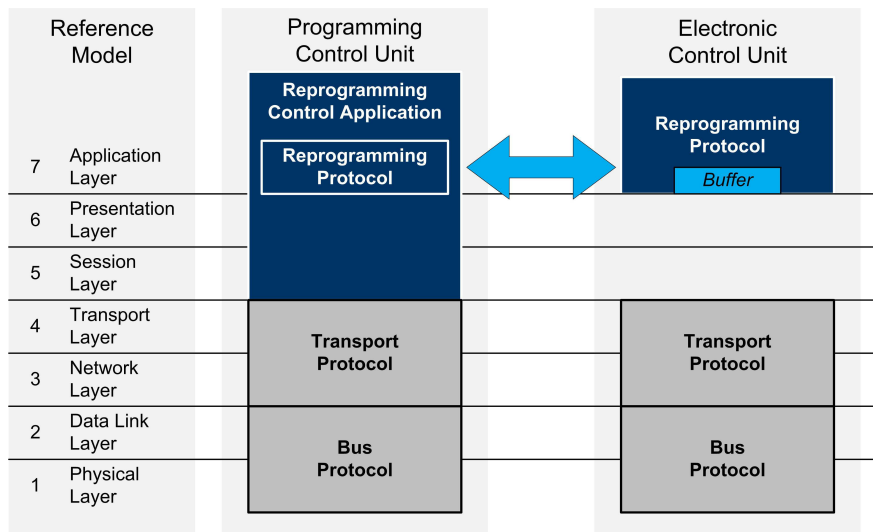


Figure 3.1-1: Reprogramming protocol overview

With the aim to use the full bandwidth and get 100% bus load the reprogramming protocol should be analysed to identify protocol dependent delay times which reduce bus load. Of course, the final overall performance depends also on the underlying communication system performance. Nevertheless, a delay on higher protocol layers is propagated through the communication stack and results usually in a delay on the physical layer. Therefore, it might be sufficient to find a generic approach to accelerate data transfer on the reprogramming protocol layer. In a second step the power of the generic approach will be discussed if it is mapped to the real existing automotive diagnostic communication protocol UDS.

3.2 Double buffered data transfer

Within this chapter a generic approach to accelerate data transfer on the programming protocol layer is discussed. Finally a solution to map this approach to a standardised protocol will be provided.

An electronic control unit (ECU) typically provides buffer to receive data within an established communication link (refer to figure 3.1-1). The maximum buffer size can vary

depending on the underlying bus system and the corresponding communication protocol stack (refer to chapter 4). Today's state of the art reprogramming process e.g. within the automotive industry²⁸ is an alternating sequence of data transfer and physical reprogramming. The programming control unit (PCU) segments the complete data to be programmed into smaller packages according to the maximum data size that can be transferred via the bus system. The ECU receives that data in the buffer and programs them into the physical non-volatile-memory (NVM), e.g. flash memory or EEPROM. This sequence will repeat until all data are transmitted from the PCU to the ECU and successfully programmed. Figure 3.2-1 depicts that basic scenario with a view to the bus system traffic and the corresponding single buffer.

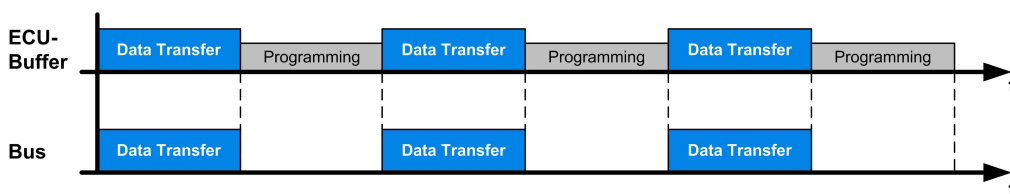


Figure 3.2-1: Single buffer data transfer

This buffer is the target for the data communication link and also the data source for physical reprogramming process. During the data transfer the received data are stored in the buffer (write access). During the physical reprogramming process (read access) the buffer is locked for data reception. Hence, no data are transmitted and a gap is visible on the bus. After successful reprogramming process execution the buffer is unlocked and the PCU can start to transmit the next data segment.

According to the initial aim to use the full bandwidth and to get 100% bus load the gap shall be filled by a concurrent data transfer to a second buffer while the first buffer content is physically programmed.

This approach will be discussed below by the calculation of data transfer ratio. Starting from a single buffer system the time that is required to transmit and program the total amount of data is calculated by formula 3.2-1. The total programming time t_{Prog} is the sum of the total data transfer time $t_{\text{DataTransfer}}$ and the total physical programming time t_{PhysProg} :

$$t_{\text{Prog}} = \sum t_{\text{DataTransfer}} + \sum t_{\text{PhysProg}}$$

$$t_{\text{Prog}} = n \cdot t_{\text{DataTransfer}} + n \cdot t_{\text{PhysProg}}$$

²⁸ Refer to the standardised reprogramming sequence based on diagnostic protocol UDS in chapter 2.3 and chapter 4.1.3.

$$t_{\text{Prog}} = n \cdot (t_{\text{DataTransfer}} + t_{\text{PhysProg}}) \quad (3.2-1)$$

The term n represents the number of repetitions to transfer the total amount of data (“DataSize”) by bus system specific segments (“SegmentSize”) which can be transmitted by a single data transfer.

$$n = \frac{\text{DataSize}}{\text{SegmentSize}} \quad (3.2-2)$$

Typically a data transfer results in a last segment with less data bytes than the segment size. In that case the individual time required for that data transfer and physical reprogramming has to be calculated. On the other hand the influence of the transmitted last data segment and the resulting calculation error is smaller as more data segments are transmitted²⁹. Hence, formula 3.2-2 is simplified and the term n will be rounded up to the next integer value.

$$n = \left\lceil \frac{\text{DataSize}}{\text{SegmentSize}} \right\rceil \quad (3.2-3)$$

If a second application buffer is available, a second data transfer could be initiated in parallel to the second buffer until the first buffer’s data are programmed. By this buffer architecture two different scenarios are possible for the total programming time calculation:

Scenario 1: $t_{\text{Data Transfer}} \geq t_{\text{PhysicalProgramming}}$

Scenario 2: $t_{\text{Data Transfer}} < t_{\text{PhysicalProgramming}}$

The maximum value for the total programming performance increase depends on two different parameters:

- The number of transmitted segments n (refer to formula 3.2-2) where the gap on the bus could be visible.
- The ratio between data transfer time $t_{\text{DataTransfer}}$ and the physical programming time t_{PhysProg} .

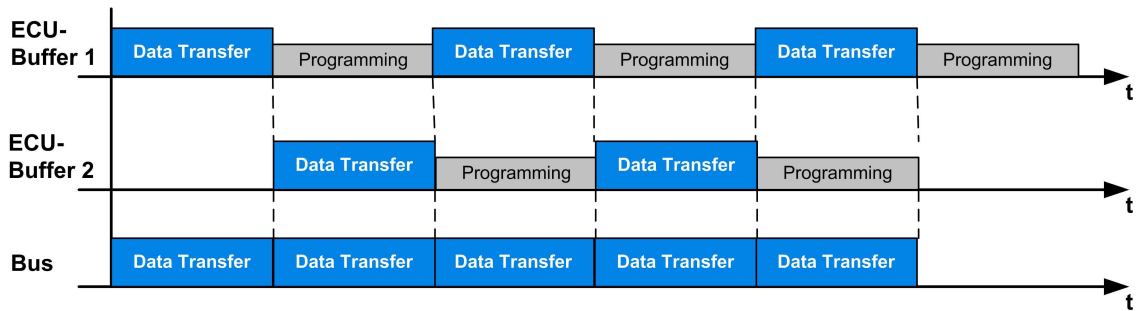
$$x = \frac{t_{\text{DataTransfer}}}{t_{\text{PhysProg}}} \quad (3.2-4)$$

²⁹ For $n \geq 20$ the last segment influences the total programming time by $< 5\%$. For $n \geq 50$ the last segment influences the total programming time by $< 2\%$.

Scenario 1 ($t_{\text{Data Transfer}} \geq t_{\text{Physical Programming}}$)

Figure 3.2-2 depicts the scenario 1 with $t_{\text{Data Transfer}} \geq t_{\text{Physical Programming}}$. In that case there is no gap visible on the bus. The physical programming time is only visible on the very last segment.

$$t_{\text{Data Transfer}} = t_{\text{Physical Programming}}$$



$$t_{\text{Data Transfer}} > t_{\text{Physical Programming}}$$

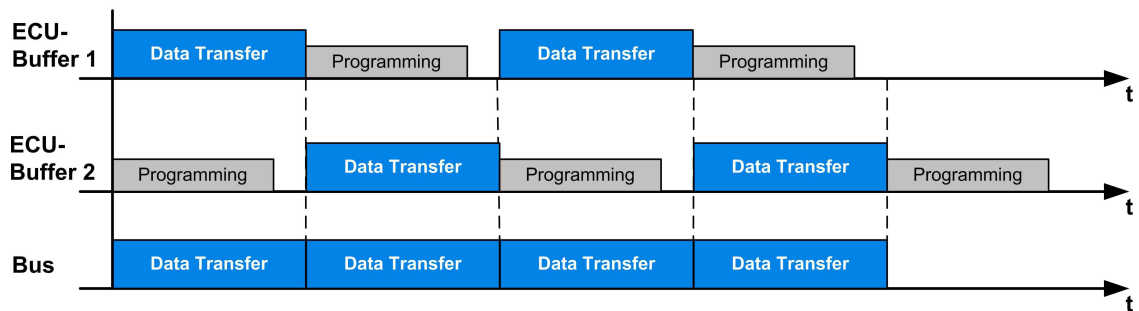


Figure 3.2-2: Double buffered data transfer – scenario 1

The relation between data transfer time $t_{\text{Data Transfer}}$ and the physical programming time t_{PhysProg} is:

$$x = \frac{t_{\text{Transfer}}}{t_{\text{PhysProg}}} \Rightarrow t_{\text{PhysProg}} = \frac{t_{\text{Transfer}}}{x} \quad | \quad x \geq 1 \quad (3.2-5)$$

The optimised total programming time $t_{\text{Prog_opt1}}$ is calculated by:

$$t_{\text{Prog_opt1}} = n \cdot t_{\text{Transfer}} + t_{\text{PhysProg}} \quad (3.2-6)$$

Compared with the initial transfer concept to a single bus with visible gaps the relative programming time reduction relation is:

$$R_n = 1 - \frac{t_{\text{Prog_opt1}}}{t_{\text{Prog}}} = 1 - \frac{n \cdot t_{\text{Transfer}} + t_{\text{PhysProg}}}{n \cdot t_{\text{Transfer}} + n \cdot t_{\text{PhysProg}}} \quad (3.2-7)$$

With (3.2-5) the generic formula for the overall time reduction ratio is:

$$R_n = 1 - \frac{n \cdot t_{\text{Transfer}} + \frac{t_{\text{Transfer}}}{x}}{n \cdot t_{\text{Transfer}} + n \cdot \frac{t_{\text{Transfer}}}{x}} = 1 - \frac{t_{\text{Transfer}} \left(n + \frac{1}{x} \right)}{t_{\text{Transfer}} \left(n + \frac{n}{x} \right)}$$

$$R_n = 1 - \frac{\left(n + \frac{1}{x} \right)}{\left(n + \frac{n}{x} \right)} \quad (3.2-8)$$

If $t_{\text{DataTransfer}} = t_{\text{PhysicalProgramming}}$ the value for x is equal to 1. Depending on the number of transmitted segments n the maximum time reduction varies.

$$R_n = 1 - \frac{n \cdot t_{\text{Transfer}} + t_{\text{Transfer}}}{n \cdot t_{\text{Transfer}} + n \cdot t_{\text{Transfer}}} = 1 - \frac{n+1}{2n} \quad (3.2-9)$$

$$\lim_{n \rightarrow 1} R_n = 1 - \frac{n+1}{2n} = 1 - \frac{2}{2} = 0$$

$$\lim_{n \rightarrow \infty} R_n = 1 - \frac{n+1}{2n} = 1 - \frac{1}{2} = 0.5$$

If $t_{\text{Data Transfer}} = t_{\text{PhysicalProgramming}}$ the maximum reduction is in range of

$$0 \leq R_n \leq 0.5 \quad | \quad n > 0$$

and is in maximum 50%. Figure 3.2-3 depicts the graphical results of formula 3.2-8 and 3.2-9 and visualise the effect of double buffered data communication. The approach with two buffers provides already a benefit if only 2 different data segment transfers are necessary ($n=2$). A saturation is visible if many data segments transfers ($n>20$) are necessary.

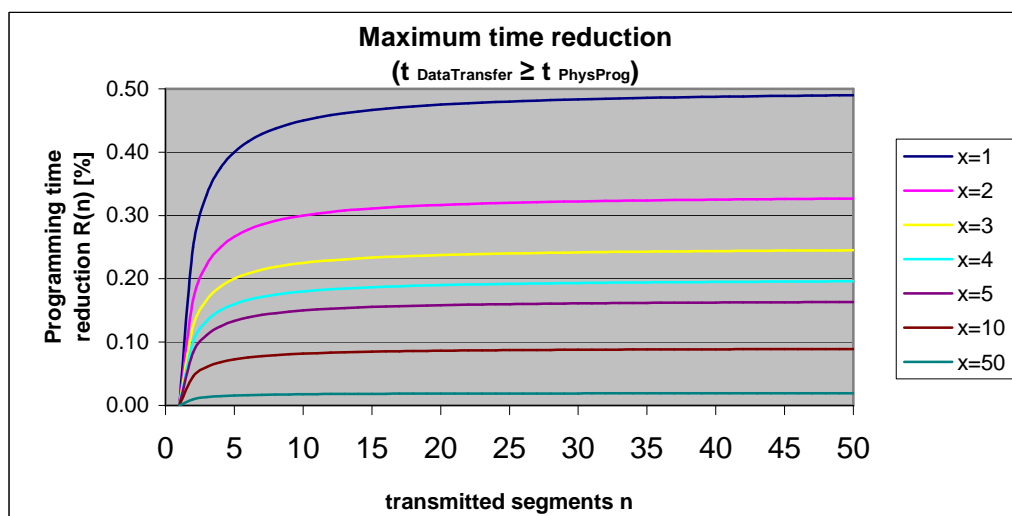


Figure 3.2-3: Maximum time reduction for $t_{\text{Data Transfer}} \geq t_{\text{PhysicalProgramming}}$

Figure 3.2-3 depicts also the impact of the relation between the data transfer time $t_{\text{DataTransfer}}$ and the physically programming time $t_{\text{PhysicalProgramming}}$.

The maximum value is only possible if data transfer time is equal to physical programming time ($x=1$). In that case the maximum time reduction $R(n)$ for a double buffered data transfer is up to 40% if already 5 segments are transmitted.

If the microcontroller is able to program data several times faster than it requires to transferring that amount of data the time reduction benefit will decrease. The longer $t_{\text{Data Transfer}}$ is ($x>1$), the less programming time reduction $R(n)$ is possible. If $t_{\text{Data Transfer}} \gg t_{\text{PhysicalProgramming}}$ ($x \rightarrow \infty$) no or only a small reduction of the total reprogramming time is possible independent of the number of transmitted segments.

$$\lim_{x \rightarrow \infty} R_n = 1 - \frac{\left(n + \frac{1}{x}\right)}{n \left(1 + \frac{1}{x}\right)} = 1 - \frac{n}{n} = 0 \quad | \quad n > 0 \quad (3.2-10)$$

Figure 3.2-4 depicts the maximum values of total reprogramming time reduction $R(x)$ depending on the relation x between the data transfer time $t_{\text{DataTransfer}}$ and the physically programming time $t_{\text{PhysicalProgramming}}$ for different number of transmitted data segments n .

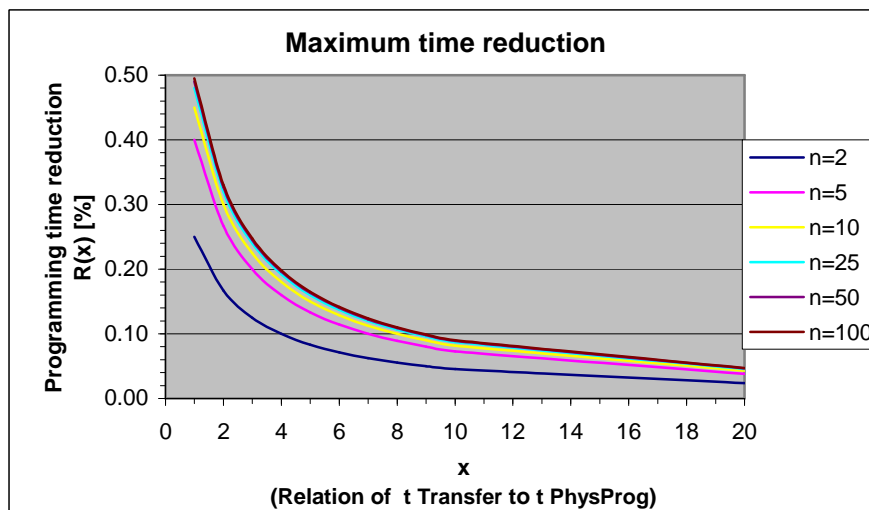


Figure 3.2-4: Total reprogramming time reduction - details

Figure 3.2-4 depicts that the effect of double buffered data transfer is decreasing if data transfer time is longer than the physical programming time. The total programming time reduction $R(x)$ is less than 10% if the data transfer time $t_{\text{Data Transfer}}$ is 9 times longer than the physical programming time $t_{\text{PhysicalProgramming}}$ ($x \geq 9$).

Scenario 2 ($t_{\text{Data Transfer}} < t_{\text{Physical Programming}}$)

For $t_{\text{Data Transfer}} < t_{\text{Physical Programming}}$ a gap on the bus will occur where no data transfer will be processed. Figure 3.2-5 depicts that scenario for a double buffered data transfer.

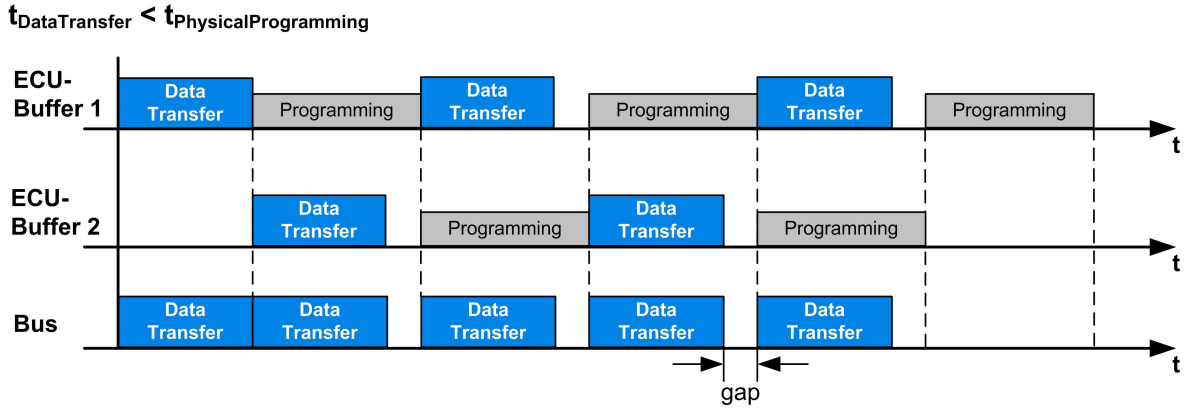


Figure 3.2-5: Double buffer data transfer – scenario 2

The relation between data transfer time $t_{\text{Data Transfer}}$ and the physical programming time t_{PhysProg} is according to formula 3.2-5:

$$x = \frac{t_{\text{Transfer}}}{t_{\text{PhysProg}}} \Rightarrow t_{\text{PhysProg}} = \frac{t_{\text{Transfer}}}{x} \quad | \quad x < 1$$

The optimised total programming time $t_{\text{Prog_opt2}}$ is calculated by:

$$t_{\text{Prog_opt2}} = t_{\text{Transfer}} + n \cdot t_{\text{PhysProg}} \quad (3.2-11)$$

Compared with the initial transfer concept to a single bus with visible gaps the programming time reduction relation is:

$$R_n = 1 - \frac{t_{\text{Prog_opt2}}}{t_{\text{Prog}}} = 1 - \frac{t_{\text{Transfer}} + n \cdot t_{\text{PhysProg}}}{n \cdot t_{\text{Transfer}} + n \cdot t_{\text{PhysProg}}}$$

$$R_n = 1 - \frac{t_{\text{Transfer}} + n \cdot \frac{t_{\text{Transfer}}}{x}}{n \cdot t_{\text{Transfer}} + n \cdot \frac{t_{\text{Transfer}}}{x}} = 1 - \frac{t_{\text{Transfer}} \left(1 + \frac{n}{x}\right)}{n \cdot t_{\text{Transfer}} \left(1 + \frac{1}{x}\right)}$$

$$R_n = 1 - \frac{1 + \frac{n}{x}}{n \cdot \left(1 + \frac{1}{x}\right)} \quad (3.2-12)$$

If the physical programming time is near to the data transfer time ($x \rightarrow 1$), the maximum programming time reduction for a large number of transmitted segments ($n \rightarrow \infty$) will be:

$$\lim_{\substack{n \rightarrow \infty \\ x \rightarrow 1}} R_n = 1 - \frac{1 + \frac{n}{x}}{n \cdot \left(1 + \frac{1}{x}\right)} = 1 - \frac{1}{2} = 0.5$$

On the other hand if the physical programming time is several times longer than the data transfer time ($x \rightarrow 0$), no reduction is possible:

$$\lim_{\substack{n \rightarrow \infty \\ x \rightarrow 0}} R_n = 1 - \frac{1 + \frac{n}{x}}{n \cdot \left(1 + \frac{1}{x}\right)} = 1 - 1 = 0$$

3.3 Method's utilisation

3.3.1 Mapping to Diagnostic Protocol ISO-14229 – UDS

Within the automotive area software reprogramming is typically part of the ECU diagnostic. Hence, the approach of doubled receive buffer should be discussed for a reprogramming process based on the diagnostic protocol “Unified Diagnostic Services” (UDS) as defined in [ISO14229].

UDS according to ISO 14229 provides the request-response behaviour for the communication between the PCU and the ECU. The PCU transmits a diagnostic request. The ECU receives that diagnostic request, processes the required functionality and sends a response (positive if successful, negative if not successful) back to the PCU. W. Zimmermann and R. Schmidgall described an overview about the protocol behaviour [Zim10-4]. A more detailed description to all defined diagnostic services is given by C. Marscholik and P. Subke [Mar07].

The programming control unit (PCU) segments the complete data to be programmed into smaller packages according to the maximum data size that could be transferred via the bus system. Figure 3.3-1 depicts that programming sequence.

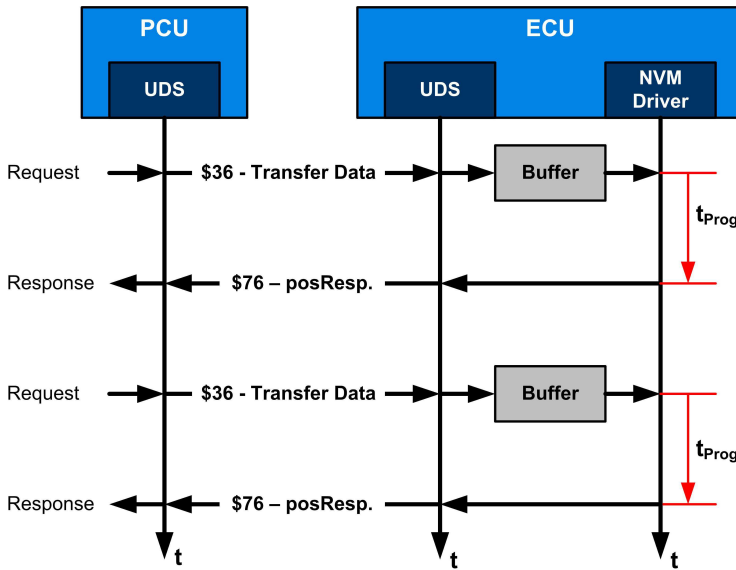


Figure 3.3-1: UDS communication via single buffered system

To transmit each data segment the PCU sends a diagnostic service request Transfer Data (Service Identifier \$36). The ECU receives the data in the buffer and programs them into the physical non-volatile-memory (NVM), e.g. flash memory or EEPROM. After successful reprogramming, the ECU sends a positive response and the sequence will repeat until all data are transmitted from the PCU to the ECU and successfully programmed.

ISO-14229 does not specify that the positive response of the diagnostic service request “\$36 - transfer data” shall be transmitted after physically reprogramming. It is allowed to separate data transfer and physical data programming and send the positive response immediately after the successfully data reception. Hence, the double buffer approach for an ECU is possible. Figure 3.3-2 depicts that approach.

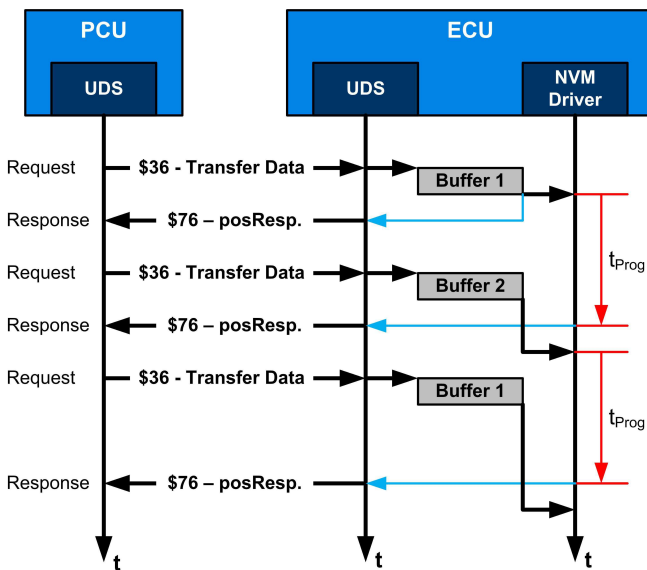


Figure 3.3-2: UDS communication via double buffer system

Sending a positive response requires two fulfilled conditions:

- 1) All data have been transferred to buffer_x
- 2) All data of the previous buffer_{x-1} have been programmed successfully

A double buffer system is a possible approach for all reprogramming protocols to accelerate the total reprogramming process.

3.3.2 Mapping to other application protocols

It is possible to map the double buffer communication approach to other application protocols. The only requirement is to prevent the ECU from concurrent access to the same buffer. It must be avoided that receive data processing and physical data reprogramming processing use the same buffer at the same time. If this is assured the approach could be ported to any other embedded application communication protocol (e.g. CCP, XCP etc.).

3.3.3 Mapping to multi controller systems

Double buffering is also a powerful approach to accelerate data transfer on multi processor systems. Figure 3.3-3 depicts an example for an ECU with two microcontrollers.

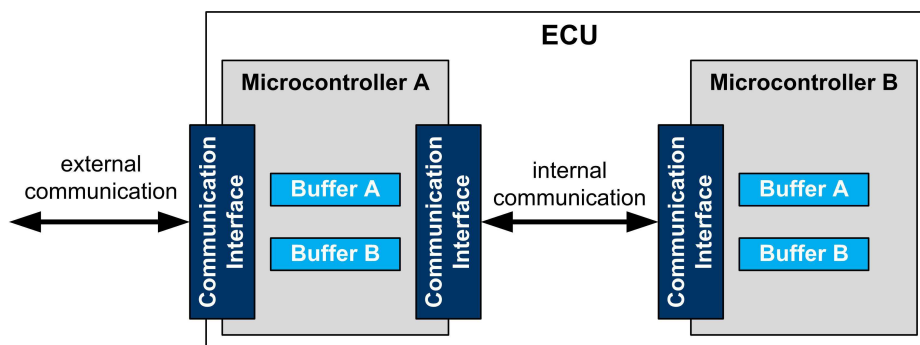


Figure 3.3-3: Multi controller system

The double buffered system provides the possibility to communicate externally and internally in parallel. Depending on the internal communication bandwidth (bus system, interface, protocol etc.) that approach reduces the total communication time.

3.4 Conclusion

With the approaches as discussed above it is possible to accelerate software reprogramming communication independent of the underlying field bus system.

Method utilisation

The benefit of double buffered systems for data transfer to a microcontroller depends on the relation between data transfer time and microcontroller's physical reprogramming time. Significant benefit is given if two environmental requirements are fulfilled:

- 1) The number of data segments n that shall be transferred to the microcontroller is at least more than 1 ($n > 1$) and
- 2) The data transfer time ($t_{\text{DataTransfer}}$) is in maximum 9 times longer than the microcontroller's physical programming time ($t_{\text{PhysicalProgramming}}$) ($x \leq 9$).

Best results are given if the data transfer time ($t_{\text{DataTransfer}}$) is equal to the microcontroller's physical programming time ($t_{\text{PhysicalProgramming}}$) ($x = 1$) and more than 100 segments shall be transmitted.

Impact to system design

The implementation of double buffered data transfer requires additional buffer resources (RAM). If the above described basic requirements are fulfilled and RAM is available it is recommended to implement that method.

An implementation of more than two buffers provides no benefit because either the additional buffers are not filled (scenario 1: $t_{\text{Data Transfer}} \geq t_{\text{PhysicalProgramming}}$) or the additional buffers are filled and can not be programmed (scenario 2: $t_{\text{Data Transfer}} < t_{\text{PhysicalProgramming}}$).

Base method for parallel processing utilisations

The approach of double buffered data reception can be utilised always when processes shall be executed in parallel to an ongoing data reception. Double buffered data transfer is a precondition to several other optimisation methods discussed later in this thesis (e.g. data compression in chapter 5, gateway routing optimisation in chapter 7 etc.). Hence, also if the relation between data transfer time and microcontroller's programming time is not given as discussed above, the implementation of double buffered data reception is recommended anyway.

4 Field bus system protocol stacks

Content

4.1 Controller Area Network.....	54
4.1.1 CAN bus protocol according to ISO 11989.....	54
4.1.2 CAN-TP according to ISO 15765-2	58
4.1.3 Complete reprogramming process based on UDS	70
4.1.4 Conclusion.....	71
4.2 FlexRay	73
4.2.1 FlexRay (FlexRay Specification 2.1)	73
4.2.2 FlexRay Transport Protocol (ISO 10681-2)	84
4.2.3 Complete reprogramming process based on UDS	91
4.2.4 Conclusion.....	93
4.3 Summary.....	94

This chapter is intended to discuss approaches to accelerate the data transfer via field bus system protocol stacks between an external programming control unit (PCU) and an electronic control unit (ECU).

According to ISO/OSI reference model (refer to section 2.5) all bus system's dependencies are encapsulated within layers 1 to 4. Hence, optimisations on layer 1 to 4 are bus system specific and therefore not necessarily common to other protocols on that layer. Nevertheless, the indisputable thesis for a communication system is given as:

The upper limit for the communication performance on a physical layer is 100% bus load.

Based on the initial thesis and with focus on automotive communication systems the following issues are discussed:

- a) The theoretically maximum of the data transfer rate on the corresponding bus system.
- b) The influencing parameters and restrictions to reach the maximum value.

As introduced in section 2.5, today many different field bus systems exist for the usage within (automotive) embedded systems. The bus access method (media access control – MAC) is a major criterion to differentiate between the systems. In [Zim10-11] W. Zimmermann and R. Schmidgall provide an overview to the different bus access methods in section 2.5.2.

With focus on automotive system's software reprogramming CAN as a representative system based on CSMA/CR media access and FlexRay as a representative of systems based on TDMA media access will be discussed.

In many cases the discussed protocol optimisation has an impact on sender and receiver side. Special optimisation steps on the PCU are not discussed explicitly.

4.1 Controller Area Network

Currently CAN is the mostly used bus system within the automotive area. CAN was developed in the late 1980s and is standardised in ISO11898 or SAE J2284 [Ets06]. The CAN protocol is required by law as the standardised communication protocol for the onboard diagnostic (OBD) communication to emission related systems (refer to ISO 15765-4 etc). Because of the high cost pressure as discussed in chapter 1 this standardised communication interface is used for the enhanced (not-emission related) diagnostics, too. The requirement by law guarantees that this interface is available for each vehicle and therefore it is typically used for software reprogramming based on a diagnostic protocol like UDS or KWP2000. Acceleration of the data transfer via CAN provides benefits for all vehicles independent of the OEM, the class or model line.

4.1.1 CAN bus protocol according to ISO 11989

4.1.1.1 Introduction

CAN is specified as a bit-oriented field bus system with a maximum bit rate of 1 Mbit/s. Mostly used bit rates are 500kbit/s, 250kbit/s and 125kbit/s. CAN provides a CSMA/CR³⁰ bus access. W. Zimmermann and R. Schmidgall give an overview to the physical layer in

³⁰ CSMA/CR: Carrier Sense Multiple Access / Collision Resolution.

detail [Zim10-7]. The data link layer [ISO11898-1] specifies the basic CAN PDU³¹ (“CAN frame”). Table 4.1-1 depicts an overview of the PDU layout in ascending bit order and the resulting PDU length.

Table 4.1-1: CAN PDU length without stuff bits

Definition	Length [bit]	Length [bit]	Unit
Start bit	1	1	bit
Arbitration field (Identifier + RTR + SRR + IDE)	11 + 1	29 + 3	bit
Control field	6	6	bit
Data field (Payload)	0 - 64	0 - 64	bit
CRC field	15	15	bit
Acknowledge field	3	3	bit
End of Frame	7	7	bit
Bus idle time	≥ 3	≥ 3	bit
Sum	47 - 111	67 - 131	bit

A CAN specific issue is the receiver’s clock generation method for bit sampling purpose. The CAN specification ISO11898 defines that only 5 bits are allowed to be equal. After 5 equal bits a signal level change must be integrated into the bit stream. These additional bits are called stuffing bits.

The number of stuffing bits depends on the payload values and could be calculated by the formula of T. Nolte et al [Nol01] and A. Burns et al [Bur07]:

$$n_{\text{Stuff}} = 0 \dots \left\lceil \frac{n_{\text{Header}} + n_{\text{Trailer}} + n_{\text{Idle}} + n_{\text{Data}} - 14 \text{ bit}}{4} \right\rceil \quad (4.1-1)$$

The net data rate can be calculated as follows:

$$f_{\text{Data_net}} = \frac{n_{\text{Data}}}{(n_{\text{Header}} + n_{\text{Trailer}} + n_{\text{Idle}} + n_{\text{Data}} + n_{\text{Stuff}})} f_{\text{Data_gross}} \quad (4.1-2)$$

A CAN PDU with 11 bit CAN identifier and 64 bit payload can have 24 stuffing bits in maximum.

³¹ PDU .. Protocol Data Unit

A CAN PDU with 29 bit CAN identifier and 64 bit payload can have 29 stuff bits.

Table 4.1-2 depicts the PDU length and the corresponding net data rate for different numbers of stuffing bits and gross data rates.

Table 4.1-2: Net data rate for CAN PDUs with 64 bit payload

11 bit CAN-ID PDU Length	Net Data Rate (f_{Data_net})				unit
	1000 kbit/s	500 kbit/s	250 kbit/s	125 kbit/s	
111 bit (0 stuff bits)	576.6 72.1	288.3 36.0	144.1 18.0	72.1 9.0	kbit/s kByte/s
123 bit (12 stuff bits)	520.3 65.0	260.2 32.5	130.1 16.3	65.0 8.1	kbit/s kByte/s
135 bit (24 stuff bits)	474.1 59.3	237.0 29.6	118.5 14.8	59.3 7.4	kbit/s kByte/s
29 bit CAN-ID PDU Length	Net Data Rate (f_{Data_net})				unit
	1000 kbit/s	500 kbit/s	250 kbit/s	125 kbit/s	
131 bit (0 stuff bits)	488.5 61.1	244.3 30.5	122.1 15.3	61.1 7.6	kbit/s kByte/s
145 bit (14 stuff bits)	441.4 55.2	220.7 27.6	110.3 13.8	55.2 6.9	kbit/s kByte/s
160 bit (29 stuff bits)	400.0 50.0	200.0 25.0	100.0 12.5	50.0 6.3	kbit/s kByte/s

4.1.1.2 Discussion

The data transfer rate depends a) on the basic bandwidth and b) on the ratio of PDU payload and protocol overhead. The PDU payload varies in the specified boundaries ($0 \text{ bit} \leq \text{payload} \leq 64 \text{ bit}$).

The protocol overhead varies on the CAN Identifier length (11 bit or 29 bit) and the resulting stuffing bits ($0 \text{ bit} \leq \text{stuffing bits} \leq 24 \text{ bit}_{11\text{bit ID}} \text{ or } 29 \text{ bit}_{29\text{bit ID}}$).

Bandwidth

Increasing bandwidth is an effective approach to accelerate the data transfer if the system is not running on the upper limit of 1 MBit/s given by the ISO 11898 protocol. Doubling the bandwidth will result in approximately a double of net data transfer rate. On the other hand increasing bandwidth reduces the possible cable length [Zim2010-12]. This has to be taken into account if a lower bit rate is in use for normal system's communication especially for wide area distributed systems (e.g. trucks, planes, trains etc.). Several recommendations are given by different standards (e.g. appendix of [ISO 11898-2],

[ISO 11898-3], [ISO 11898-5], [SAE J1939-11], [CiA³² 102] etc.). If cable length provides no restriction, then increasing the bandwidth provides a strong method to accelerate data transfer.

PDU payload

According to formula 4.1-1 the maximum ratio between payload and protocol overhead is given if the payload is configured to the maximum possible (specified) value. Hence, only CAN PDUs with 64 bit payload (maximum value) shall be configured. For further analysis and discussion only CAN PDUs with 64 bit payload are assumed.

CAN Identifier

The system performance is represented by the net data rate. For a CAN system running on 100% bus load the net data rate is limited by ISO 11898 protocol and can be in maximum only 58.2% for PDUs with 11 bit identifier (no stuffing bits) or 50% for PDUs with 29 bit identifier (no stuffing bits). But these values are not realistic in practice because a communication without any stuffing bits will usually not occur (e.g. a CAN identifier with 5 consecutive bits of an equal value (one or zero) will force a stuffing bit). No statistical evaluation of a best practice value will be discussed because this value is significantly influenced by the SDU value (payload of the CAN PDU) and this value is random from a statistical point of view.

The CAN identifier length has a significant influence on the net data rate. Compared to a PDU with 11 bit CAN identifier a PDU with 29 bit CAN identifier requires approximately 16,4%³³ more bits to transmit the same data payload of 64 bit. The net data rate is reduced equally. Hence, best performance is possible only on 11 bit CAN identifier. CAN uses the arbitration method to prevent PDU collisions. According to ISO 11898, if two or more sender nodes initiate a data transmission concurrently, the CAN identifiers are bit-wise analysed and compared by each sending node to identify the higher priority. According to ISO 11898 protocol specification the priority is the higher, the lower the CAN identifier is. This fact results in two basic requirements: 1) if reprogramming communication is in parallel to normal system's communication, the CAN identifier of the reprogramming communication shall be low (to get high priority) or 2) normal system's communication shall be disabled. Within the reprogramming sequence according to [HIS06-1], a diagnostic service is specified to disable normal communication of all

³² CiA .. CAN-in-Automation

³³ 16.36% (no stuffing bits); 16.41% (max number of stuffing bits)

currently not reprogrammed network nodes to guarantee full bandwidth for reprogramming also for less priority CAN identifiers.

4.1.2 CAN-TP according to ISO 15765-2

For communication via CAN the ISO has specified a transport protocol in ISO 15765-2. “This part of ISO 15765 specifies an unconfirmed network layer communication protocol for the exchange of data between network nodes, e.g. from ECU to ECU, or between external test equipment and an ECU. If the data to be transferred do not fit into a single CAN frame, a segmentation method is provided” [ISO15765-2_1].

A CAN PDU provides in maximum 8 byte (64 bit) payload data (refer to table 3.2-1). Hence, an ISO-15765-2-PDU is up to 8 bytes (64 bit) long. The protocol itself defines a *Protocol Control Information (PCI)*. The protocol distinguishes between four different PDU types. A *Single Frame (SF)* is used if the service data unit (SDU) has equal or less than 7 data bytes. If the SDU is larger than 7 bytes the SDU has to be segmented into several PDUs. A segmented data transfer starts with a *First Frame (FF)* and implements 2 byte PCI and 6 byte payload data. All other SDU data are transmitted by *Consecutive Frames (CF)*. A *CF* provides 7 byte payload data.

The data flow is controlled by *Flow Control frames (FC)*. A *FC* frame is always sent by the initial receiver of the data transfer. The idea is to have a handshake mechanism implemented to control the data flow on the established communication link by only two parameters (*Minimum Separation Time (ST_{min})* and *Block Size (BS)*). The receiver is able to control data transfer by these parameters.

Figure 4.1-1 depicts an overview of the different PDU types and the PDU format.

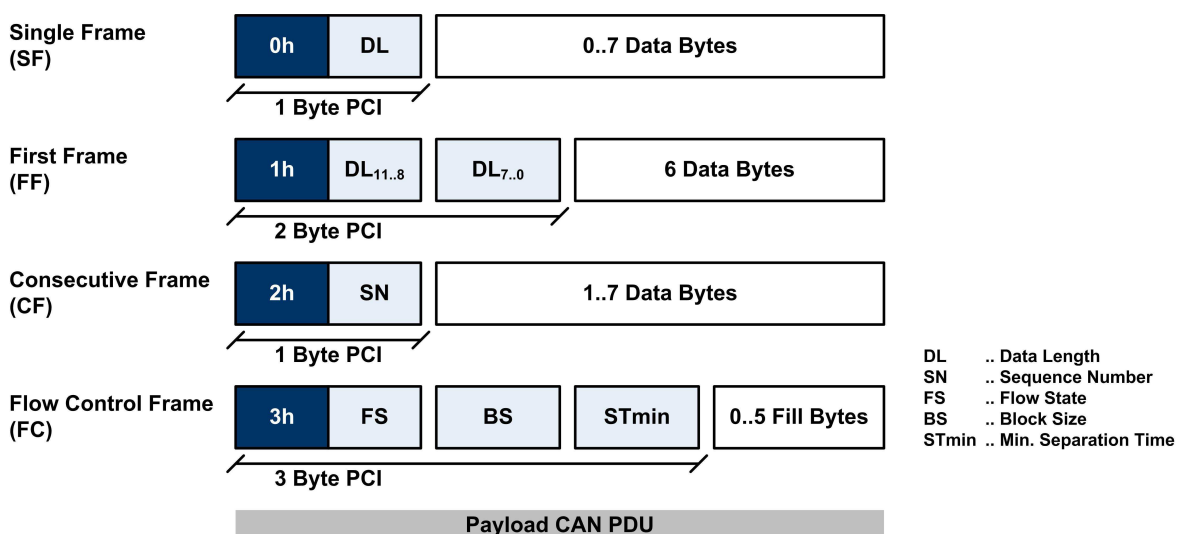


Figure 4.1-1: ISO 15765-2 Protocol Data Units format

Figure 4.1-2 depicts both possible data transfer scenarios: unsegmented data transfer and segmented data transfer:

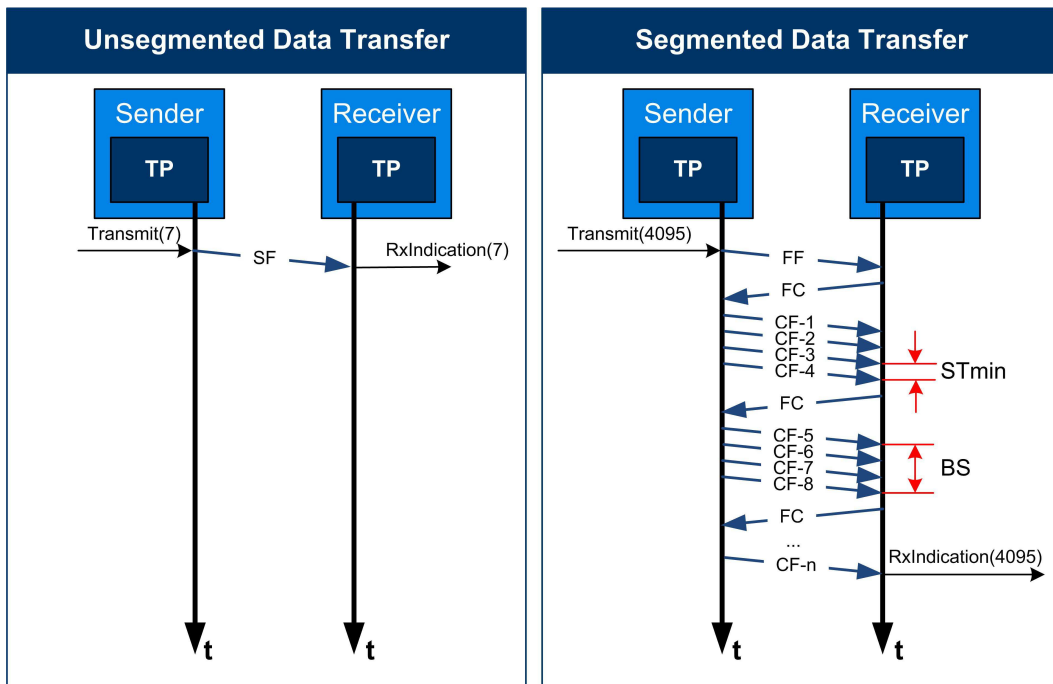


Figure 4.1-2: ISO 15765-2 communication scenarios

Within a *Single Frame (SF)* up to 7 data bytes can be transferred via CAN. The maximum size of an SDU is limited by ISO 15765-2 to 4095 byte because the data length (DL) field of a *First Frame (FF)* is 12 bit ($2^{12} - 1 = 4095$).

Unsegmented vs. segmented data transfer

It seems that an unsegmented transmission provides higher performance because no additional *Flow Control (FC)* PDUs have to be transmitted. This is correct if only one direction is analysed. For software reprogramming it has to be taken into account that the upper layer reprogramming protocol (e.g. UDS, KWP2000) requires a request – response behaviour. As depicted in figure 4.1-3 after each request a corresponding response PDU has to be sent.

The number of required PDUs is calculated by

$$\eta_{\text{PDU_unsegmentedTransfer}} = \left\lceil \frac{d_{\text{SDU}}}{d_{\text{PDU_Payload}}} \right\rceil \cdot 2 \quad (4.1-3)$$

To transfer 4095 bytes in unsegmented mode 1170 PDUs (*Single Frames*) are necessary.

$$\eta_{\text{PDUs_unsegTransfer}} = \left\lceil \frac{4095}{7} \right\rceil \cdot 2 = 1170 \text{ PDUs}$$

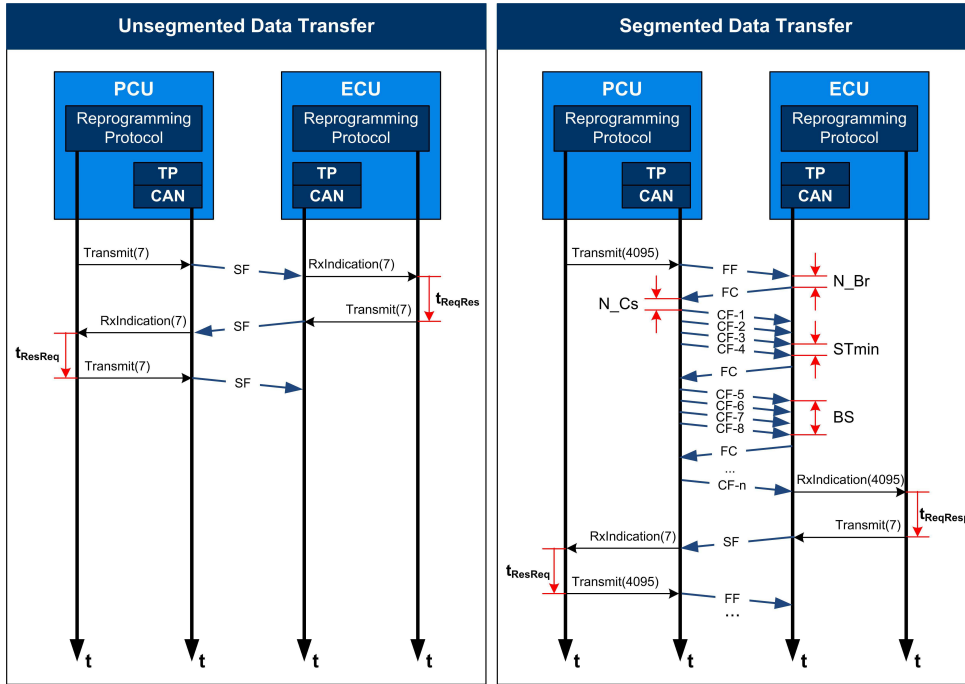


Figure 4.1-3: Request / response communication scenarios based on ISO 15765-2

To transmit the same number of data in segmented mode, a smaller number of PDUs is necessary. The number of PDUs can be calculated according to the protocol behaviour of ISO 15765-2:

$$n_{PDUs_segTransfer} = n_{FF} + \left\lceil \frac{d_{SDU} - d_{PL_FF}}{d_{PL_CF}} \right\rceil + n_{FC} + n_{Response} \tag{4.1-4}$$

It is assumed that only one *Flow Control* PDU (FC) is configured (Block size=0³⁴). In that case a data transfer of 4095 byte requires 1 *First Frame* (FF), 1 *Flow Control* (FC), 585 *Consecutive Frames* (CF) and 1 response *Single Frame* (SF) PDU. This results in 588 PDUs according to formula 4.1-4.

$$n_{PDUs_segTransfer} = 1 + \left\lceil \frac{4095 - 6}{7} \right\rceil + 1 + 1 = 588 \text{ PDUs}$$

Hence, focus for further analysis to accelerate data transfer via CAN will be on segmented data transfer only.

Impact of processing delays

The communication link performance is also influenced by processing delays because of software and hardware runtime. Both, sender and receiver nodes require processing time for the protocol handling. For the data transmission performance of the transport layer

³⁴ Block Size (BS) equal to zero requires no additional FlowControl PDU (refer to ISO 15765-2).

protocol according to ISO 15765-2 the two main influencing delay times are N_Cs and N_Br (refer to ISO15765-2). Both parameters depend on the implementation and therefore are defined to zero for the theoretical analysis in principle. The impact of those delay times is discussed in section 4.1.2.4.

4.1.2.1 Impact of Block Size parameter

The flow control parameter *Block Size (BS)* defines the number of *Consecutive Frame (CF)* PDUs that can be received by a receiver node in one block within a segmented data transmission. After reception of that block a *Flow Control (FC)* PDU has to be sent by the initial receiver node to signal the current flow state and to continue data transfer.

With focus on data transfer acceleration the *Block Size (BS)* represents an additional number of *Flow Control (FC)* PDUs (refer to figure 4.1-3). Hence, the *BS* parameter has a direct impact on the total number of TP-PDUs required to transfer the requested SDU and influences the data transfer rate. Below several formulas have been developed to calculate the total amount of PDUs and the corresponding transfer time. The total number of PDUs is calculated as depicted by the developed formula 4.1-5:

$$n_{\text{PDUs}} = \sum n_{\text{DataPDU}} + \sum n_{\text{FlowControlPDU}} \quad (4.1-5)$$

According to the definitions in ISO 15765-2 it has to be distinguished between two different cases to calculate the total amount of PDUs:

- a) *Block Size* equal to zero ($BS=0$) and
- b) *Block Size* between 1 and 255³⁵ ($1 \leq BS \leq 255$).

Case 1: BS = 0

ISO 15765-2 defines that “the BS parameter value zero (0) shall be used to indicate to the sender that no more FC PDUs shall be sent during the transmission of the segmented message. The sending network layer entity shall send all remaining consecutive frames without any stop for further FC PDUs from the receiving network layer entity” [ISO15765-2_2]. Hence, after the initial *FF* PDU only 1 *FC* PDU is required before the others are transmitted as *Consecutive Frame (CF)* PDUs. The total number of PDUs is calculated as depicted by formula 4.1-6.

$$n_{\text{PDUs}} = 2 + \left\lceil \frac{d_{\text{SDU}} - d_{\text{PL_FF}}}{d_{\text{PL_CF}}} \right\rceil \quad (4.1-6)$$

³⁵ The parameter *Block Size (BS)* is defined as an 8 bit value in ISO 15765-2.

Case 2: $1 \leq BS \leq 255$

A value of $1 \leq BS \leq 255$ indicates that the initial sender shall send the corresponding number of *Consecutive Frames* and shall then wait for a next *Flow Control* PDU [ISO15765-2_2]. Formula 4.1-7 depicts a fourth summand that represents the number of additional *Flow Control* PDUs depending on 1) the total number of *Consecutive Frame* (CF) PDUs which are required to transmit the SDU data and 2) the *Block Size* (BS).

$$n_{\text{PDUs}} = 1_{(\text{FF})} + \left\lceil \frac{d_{\text{SDU}} - d_{\text{PL_FF}}}{d_{\text{PL_CF}}} \right\rceil + 1_{(\text{FC})} + \left\lceil \frac{\left\lceil \frac{d_{\text{SDU}} - d_{\text{PL_FF}}}{d_{\text{PL_CF}}} \right\rceil}{BS} \right\rceil - 1$$

$$n_{\text{PDUs}} = 1 + \left\lceil \frac{d_{\text{SDU}} - d_{\text{PL_FF}}}{d_{\text{PL_CF}}} \right\rceil + \left\lceil \frac{\left\lceil \frac{d_{\text{SDU}} - d_{\text{PL_FF}}}{d_{\text{PL_CF}}} \right\rceil}{BS} \right\rceil \quad (4.1-7)$$

For the first theoretical research an ideal system with no additional delays (system runtime, software runtime etc.) during the protocol communication handling is assumed. The net data transfer rate $f_{\text{Data_net}}$ is calculated as depicted below:

$$f_{\text{Data_net}} = \frac{d_{\text{SDU_ISO15765-2}}}{n_{\text{PDUs}} \cdot d_{\text{PDU_ISO11898}}} f_{\text{Data_gross}} \quad (4.1-8)$$

ISO 15765-2 also distinguishes between four different address modes for communication via CAN [ISO15765-2_3]. The address mode has an impact on the PDU structure and on the communication performance, too. The different parameters for the analysis of the four cases are specified in table 4.1-3.

Table 4.1-3: Parameter definition for block size analysis

Parameter	Address mode [ISO 15765-2_3]				unit
	11 bit		29 bit		
	normal	mixed	normal	mixed	
SDU length (data to transmit)	4095 32760	4095 32760	4095 32760	4095 32760	byte bit
PDU length ISO 11898 $d_{\text{PDU_ISO11898}}$	123	123	145	145	bit
Payload <i>FirstFrame</i> (FF) $d_{\text{PL_FF}}$	6 48	5 40	6 48	5 40	byte bit
Payload <i>ConsecutiveFrame</i> (CF) $d_{\text{PL_CF}}$	7 56	6 48	7 56	6 48	byte bit

The data transfer rate is analysed for a bandwidth of 1.000 kbit/s, 500 kbit/s, 250 kbit/s and 125 kbit/s. These values are the most important gross data rates within the automotive area. 1.000 kbit/s is the maximum value defined for CAN's physical layer according to [ISO11898-1]. Table 4.1-4 depicts the calculated net data transfer values according to the different address modes (CAN identifier lengths and normal/mixed mode).

Table 4.1-4: $f_{\text{Data_Net max}}$ for different bandwidths

Address Mode		BlockSize	0	1	2	8	18	32	40
1000 kBit/s									
11 bit ID	Normal Address	Net Data Rate [kBit/s]	439.5	220.3	293.5	390.9	416.8	426.4	429.3
		Net Data Rate [kByte/s]	54.9	27.5	36.7	48.9	52.1	53.3	53.7
	Mixed Address	Net Data Rate [kBit/s]	377.2	189.0	252.0	335.5	357.8	366.0	368.0
		Net Data Rate [kByte/s]	47.1	23.6	31.5	41.9	44.7	45.7	46.0
29 bit ID	Normal Address	Net Data Rate [kBit/s]	375.9	188.4	251.0	334.3	356.4	364.7	367.1
		Net Data Rate [kByte/s]	47.0	23.6	31.4	41.8	44.6	45.6	45.9
	Mixed Address	Net Data Rate [kBit/s]	322.6	161.6	215.5	286.9	306.0	313.0	314.7
		Net Data Rate [kByte/s]	40.3	20.2	26.9	35.9	38.3	39.1	39.3
500 kBit/s									
11 bit ID	Normal Address	Net Data Rate [kBit/s]	219.8	110.2	146.8	195.5	208.4	213.2	214.6
		Net Data Rate [kByte/s]	27.5	13.8	18.3	24.4	26.1	26.7	26.8
	Mixed Address	Net Data Rate [kBit/s]	188.6	94.5	126.0	167.8	178.9	183.0	184.0
		Net Data Rate [kByte/s]	23.6	11.8	15.7	21.0	22.4	22.9	23.0
29 bit ID	Normal Address	Net Data Rate [kBit/s]	187.9	94.2	125.5	167.1	178.2	182.3	183.6
		Net Data Rate [kByte/s]	23.5	11.8	15.7	20.9	22.3	22.8	22.9
	Mixed Address	Net Data Rate [kBit/s]	161.3	80.8	107.7	143.5	153.0	156.5	157.4
		Net Data Rate [kByte/s]	20.2	10.1	13.5	17.9	19.1	19.6	19.7
250 kBit/s									
11 bit ID	Normal Address	Net Data Rate [kBit/s]	109.9	55.1	73.4	97.7	104.2	106.6	107.3
		Net Data Rate [kByte/s]	13.7	6.9	9.2	12.2	13.0	13.3	13.4
	Mixed Address	Net Data Rate [kBit/s]	94.3	47.3	63.0	83.9	89.5	91.5	92.0
		Net Data Rate [kByte/s]	11.8	5.9	7.9	10.5	11.2	11.4	11.5
29 bit ID	Normal Address	Net Data Rate [kBit/s]	94.0	47.1	62.8	83.6	89.1	91.2	91.8
		Net Data Rate [kByte/s]	11.7	5.9	7.8	10.4	11.1	11.4	11.5
	Mixed Address	Net Data Rate [kBit/s]	80.6	40.4	53.9	71.7	76.5	78.2	78.7
		Net Data Rate [kByte/s]	10.1	5.1	6.7	9.0	9.6	9.8	9.8
125 kBit/s									
11 bit ID	Normal Address	Net Data Rate [kBit/s]	54.9	27.5	36.7	48.9	52.1	53.3	53.7
		Net Data Rate [kByte/s]	6.9	3.4	4.6	6.1	6.5	6.7	6.7
	Mixed Address	Net Data Rate [kBit/s]	47.1	23.6	31.5	41.9	44.7	45.7	46.0
		Net Data Rate [kByte/s]	5.9	3.0	3.9	5.2	5.6	5.7	5.8
29 bit ID	Normal Address	Net Data Rate [kBit/s]	47.0	23.6	31.4	41.8	44.6	45.6	45.9
		Net Data Rate [kByte/s]	5.9	2.9	3.9	5.2	5.6	5.7	5.7
	Mixed Address	Net Data Rate [kBit/s]	40.3	20.2	26.9	35.9	38.3	39.1	39.3
		Net Data Rate [kByte/s]	5.0	2.5	3.4	4.5	4.8	4.9	4.9

Figure 4.1-4 and figure 4.1-5 depict the net data transfer rate $f_{\text{Data_net}}$ for different block sizes. Of course, the parameter *Block Size (BS)* is a discrete value. Hence, only the calculated (discrete) values shall be plotted within the diagrams. On the other hand the tendency of the BS curve is important and therefore a line between the discrete measurement points was plotted, too.

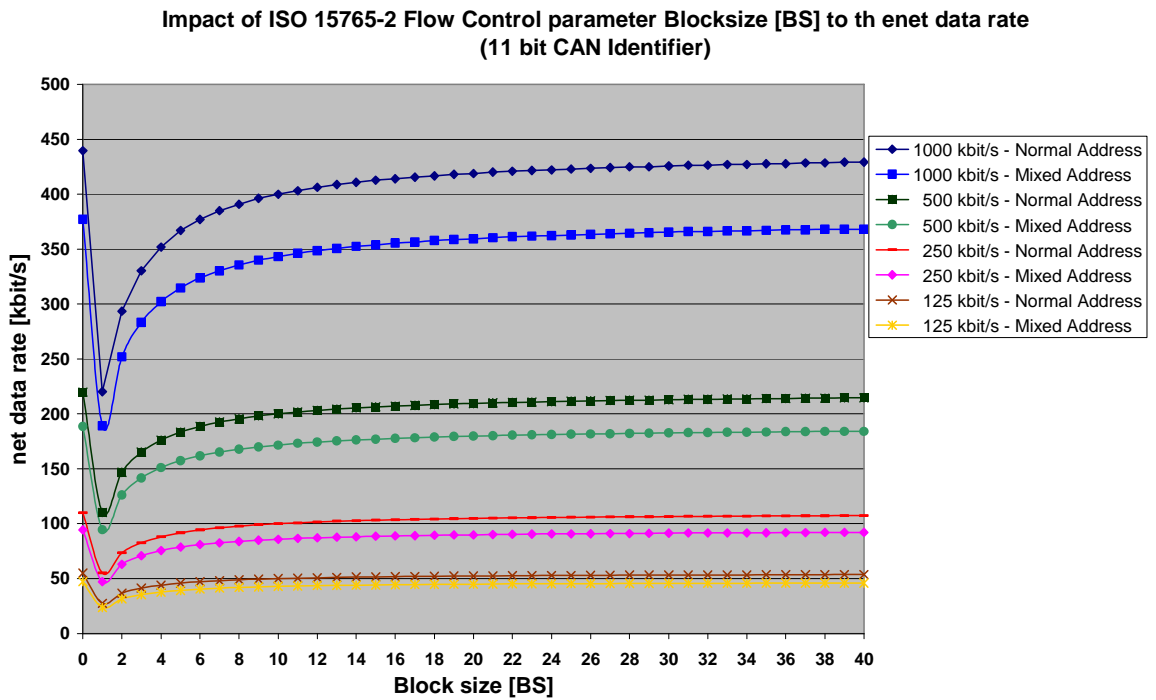


Figure 4.1-4: Block size analysis for 11 bit CAN identifier

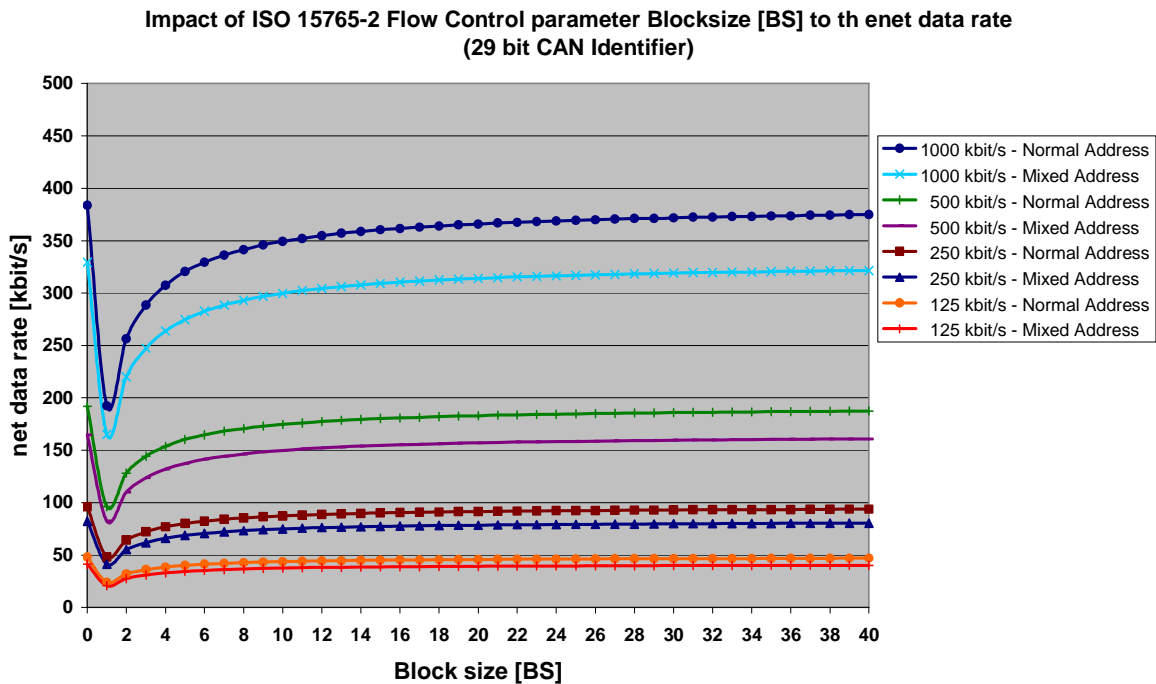


Figure 4.1-5: Block size analysis for 29 bit CAN identifier

Discussion

The ISO 15765-2 transport protocol’s flow control parameter *Block Size (BS)* has a significant impact to the net data rate. The tendencies of all diagrams are similar. The block size configuration generates at least 1 additional *Flow Control (FC)* PDU without any payload.

In worst case ($BS=1$) a *Flow Control* PDU without any payload data is required after each *Consecutive Frame (CF)*. This results in a decreasing net data transfer performance of 50.1% because the total number of PDUs to transmit the payload is doubled. If PCU's or ECU's processing time delay (e.g. system or software runtime etc.) is also taken into account, the net data rate reduction is additionally increasing (refer to case study).

The system performance depends significantly on *Block Size* parameter values in the range of $1 \leq BS \leq 12$ PDUs. A *Block Size* value more than 12 results in a saturation line. For a *Block Size* value more than 15 the differences between the calculation values are less than 0.4%. The maximum *Block Size* value of 255 results in only 2 additional *Flow Control* PDUs.

The *Block Size* configuration depends directly on the system's buffer resources. If enough buffer (buffer size = SDU_{max}) is available, a block size equal to 0 is possible. But here it has also to be distinguished for which scenario data transfer acceleration shall be configured. If a programming control unit (PCU) has a direct link to the ECU, a block size $BS = 0$ may be possible. If the PDU communicates via network, then the gateway's buffer resources could be a reason for controlling data flow via *Block Size* parameter, especially if several communication links are active in parallel. In that case a gateway could work nearly its RAM resource boundaries, and flow control based on *Block Size* is necessary to limit the maximum data transfer per block to protect against buffer overrun. As a result a good ratio between buffer resources and performance limitation is necessary and has to be taken into account during system or network design.

4.1.2.2 Impact of Minimum Separation Time parameter

ISO 15765-2 specifies a second *Flow Control* parameter that has an impact on the data transfer rate: The minimum *Separation Time (STmin)* defines the minimum delay between two *Consecutive Frames (CF)* in a segmented data transfer (refer to figure 4.1-2). The value can vary in the range of $0 \leq STmin \leq 127$ ms [ISO15765-2_4].

Of course, with focus on data transfer acceleration and the basic aim to generate 100% bus load any delays without any data transfer should be eliminated or at least the delay shall be minimised. Hence, this parameter shall be set to zero for the best data transfer performance. If *STmin* is set to zero some other system requirements have to be fulfilled: 1) a receiver must be able to receive data with no inter frame delay and 2) a sender must be able to perform such a data transmission. $STmin = 0$ means that the sender shall send as fast as possible. Therefore it is required that $STmin_{receiver} \leq STmin_{sender}$ to guarantee a stable connection link.

The time t_{STmin} is an additional time delay that enlarges the required time for the transmission of an SDU via protocol. To calculate the possible data transfer rate according to parameter $STmin$, formula 4.1-9 was developed from formula 4.1-8.

Formula 4.1-8:

$$f_{Data_net} = \frac{d_{SDU_ISO15765-2}}{n_{PDU_s} \cdot d_{PDU_ISO11898}} f_{Data_gross}$$

with

$$t_{bit} = \frac{1}{f_{Data_gross}}$$

$$f_{Data_net} = \frac{d_{SDU_ISO15765-2}}{(n_{PDU_s} \cdot d_{PDU_ISO11898} \cdot t_{bit}) + t_{totalSTmin}} \quad (4.1-9)$$

As depicted in figure 4.1-2 after the last *Consecutive Frame (CF)* of a block no further separation time ($STmin$) occurs because now the sender waits either for a *Flow Control (FC)* PDU or for transmission has been finalised. If the *Block Size (BS)* is equal to zero ($BS=0$), no additional *Flow Control (FC)* PDU is required. In that case ($BS = 0$) the overall additional time t_{STmin} is calculated as

$$t_{totalSTmin} = (n_{CF_PDU} - 1) \cdot t_{STmin} \quad (4.1-10)$$

If $1 \leq BS \leq 255$, on each end of a block no separation time ($STmin$) occurs because a *Flow Control (FC)* PDU is required with the exception of the last block. This results in

$$t_{totalSTmin} = \left(n_{CF_PDU} - \left\lceil \frac{n_{CF_PDU}}{BS} - 1 \right\rceil - 1 \right) \cdot t_{STmin} \quad (4.1-11)$$

The total data transfer rate is now calculated as depicted in formula 4.1-12 and 4.1-13:

For $BS = 0$:

$$f_{Data_net} = \frac{d_{SDU_ISO15765-2}}{(n_{PDU_s} \cdot d_{PDU_ISO11898} \cdot t_{bit}) + (n_{CF_PDU} - 1) \cdot t_{STmin}} \quad (4.1-12)$$

For $1 \leq BS \leq 255$:

$$f_{Data_net} = \frac{d_{SDU_ISO15765-2}}{(n_{PDU_s} \cdot d_{PDU_ISO11898} \cdot t_{bit}) + \left(n_{CF_PDU} - \left\lceil \frac{n_{CF_PDU}}{BS} - 1 \right\rceil - 1 \right) \cdot t_{STmin}} \quad (4.1-13)$$

The parameter ST_{min} is only represented in milliseconds (ms) in the range of 1 to 127 (\$01 - \$7F). For \$01 to \$09 the value has to be interpreted as microseconds (μs) in a division of $100\mu s$ [ISO15765-2_4]. The value of $t_{ST_{min}}$ is calculated by:

$$t_{ST_{min}} = (ST_{min} - \$F0) \cdot 100 \mu s \quad | \quad \$F1 \leq ST_{min} \leq \$F9 \quad (4.1-14)$$

Figure 4.1-6 depicts the graphical analysis for the minimum separation time parameter ST_{min} . It shows that the net data transfer rate is decreasing significantly, if ST_{min} is not equal to zero.

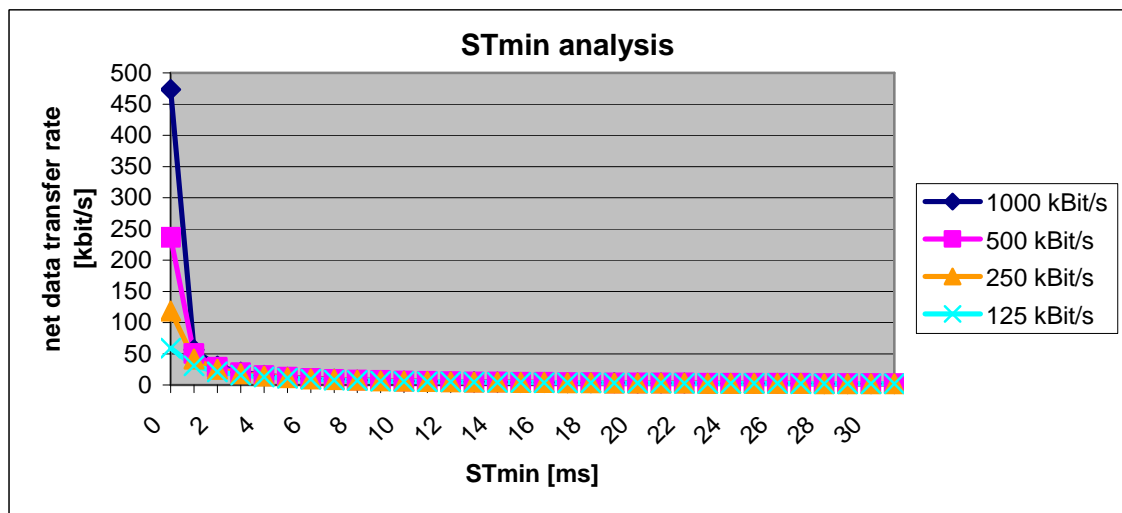


Figure 4.1-6: Data transfer rate depending on ST_{min}

Figure 4.1-6 shows also that the main effects occur in a range of $ST_{min} \leq 2$ ms. For $ST_{min} > 2$ ms the system approximates asymptotically to 0. Figure 4.1-7 depicts the range from $0 \leq ST_{min} \leq 1$ ms in more detail.

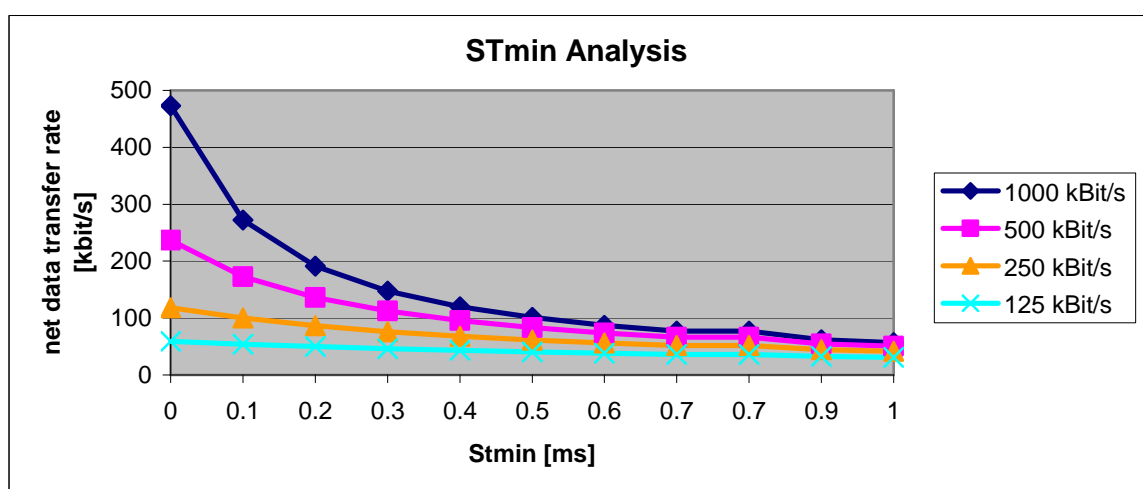


Figure 4.1-7: Data transfer rate depending on ST_{min} – detailed diagram

Discussion

Basically any separation time will decrease the data transfer performance. Within that delay no data transfer will be performed. Hence, the first finding to accelerate data transfer is to eliminate any delays by ISO 15765-2 protocol and configure the $STmin$ parameter equal or at least as close as possible to zero.

Figure 4.1-6 and figure 4.1-7 show that the net data rate impact of increasing $STmin$ values is higher the faster the basic bus system data rate is. The explanation is that the faster the basic bus system is the more PDUs can actually be transmitted during an $STmin$ time window. Any $STmin$ delay reduces the number of transmitted PDUs within that time and as a result the net data transfer rate will decrease significantly, too.

4.1.2.3 Comparative impact of $STmin$ and BS

Figure 4.1-8 compares the impact of the *Flow Control* parameters $STmin$ and *Block Size* (BS)³⁶. The *Flow Control* parameter *minimum Separation Time* ($STmin$) has a significantly higher impact on the net data transfer rate than the *Flow Control* parameter *Block Size* (BS). The BS parameter represents the receiver's buffer (RAM) resources whereas $STmin$ represents receiver's system performance. Buffer (RAM) resources for buffering of at least 8 CAN PDUs are typically available. A flashloader has normally access to the complete ECU RAM because application software is not active and requires no RAM resources. Hence, a *Block Size* equal to zero ($BS = 0$) is possible if all other network nodes (gateways) on that communication link provide equal buffer resources.

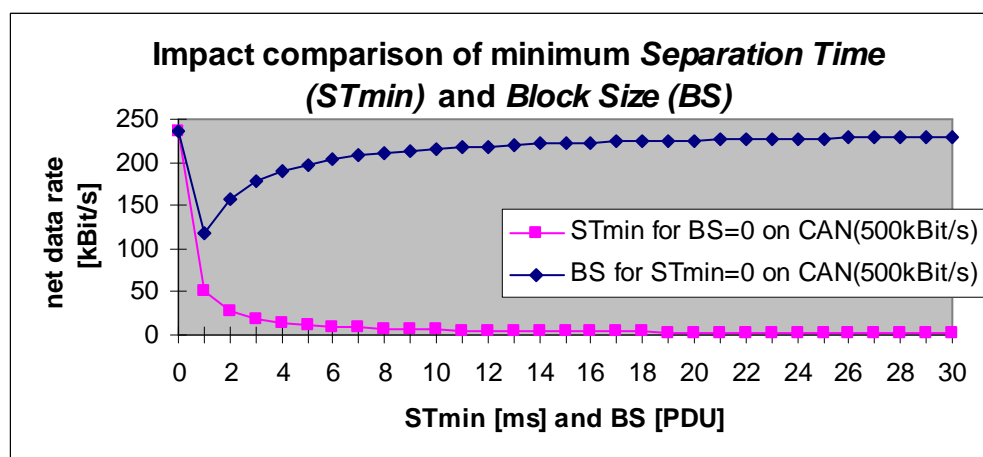


Figure 4.1-8: Comparison of impact of $STmin$ and BS

³⁶ Of course, the parameter *Block Size* (BS) is a discrete value. Hence, only the calculated (discrete) values shall be plotted within the diagrams. On the other hand the tendency of the BS curve is important and therefore it has been decided to plot a line between the discrete measurement points, too.

A minimum *Separation Time* (ST_{min}) greater than 0.1 ms reduces the data transfer performance that significantly, that all other parameters are negligible. Hence, it must be possible to design an embedded network which is able to communicate with $ST_{min} = 0$ via CAN and provides buffer resources for at least 12 or more CAN PDUs.

4.1.2.4 Processing delays

As discussed above additional delays based on the hardware or software runtime for protocol processing have to be taken into account. Based on formula 4.1-9 the additional processing delay has to be included in the transport layer performance calculation model.

$$f_{Data_net} = \frac{d_{SDU_ISO15765-2}}{(n_{PDU_s} \cdot d_{PDU_ISO11898} \cdot t_{bit}) + t_{totalSTmin} + t_{ProtocolProcessing}}$$

Figure 4.1-3 depicts that N_{Br} and N_{Cs} are the software processing timings of flow control handling. Hence, both delays occur if a *Flow Control* (FC) PDU is processed. The number of necessary *Flow Control* (FC) PDUs is calculated by a part of formula 4.1-7:

$$n_{FC_PDUs} = \left\lceil \frac{\left\lceil \frac{d_{SDU} - d_{PL_FF}}{d_{PL_CF}} \right\rceil}{BS} \right\rceil$$

The processing delay is calculated as

$$t_{processingDelay} = n_{FC_PDUs} (t_{N_Br} + t_{N_Cs}) = \left\lceil \frac{\left\lceil \frac{d_{SDU} - d_{PL_FF}}{d_{PL_CF}} \right\rceil}{BS} \right\rceil (t_{N_Br} + t_{N_Cs}) \quad (4.1-15)$$

The final formula to calculate data transfer performance is:

For $BS = 0$:

$$f_{Data_net} = \frac{d_{SDU_ISO15765-2}}{(n_{PDU_s} \cdot d_{PDU_ISO11898} \cdot t_{bit}) + (n_{CF_PDU} - 1) \cdot t_{STmin} + t_{N_Br} + t_{N_Cs}} \quad (4.1-16)$$

For $1 \leq BS \leq 255$:

$$f_{Data_net} = \frac{d_{SDU_ISO15765-2}}{(n_{PDU_s} \cdot d_{PDU_ISO11898} \cdot t_{bit}) + \left(n_{CF_PDU} - \left\lceil \frac{n_{CF_PDU}}{BS} - 1 \right\rceil - 1 \right) \cdot t_{STmin} + \left\lceil \frac{n_{CF_PDU}}{BS} \right\rceil (t_{N_Br} + t_{N_Cs})} \quad (4.1-17)$$

4.1.3 Complete reprogramming process based on UDS

As described in chapter 2, the ISO 14229 - USD protocol is the currently most common diagnostic protocol. Reprogramming of the ECU's application software is controlled by this standard. The communication is based on a diagnostic request – diagnostic response behaviour. Due to the UDS protocol all application data which shall be transferred are segmented to smaller partitions according to the underlying bus system transport layer protocol's maximum value. In case of the CAN bus system and the transport layer protocol according to ISO 15765-2 this value is limited to 4095 byte maximum.

To calculate the total download time all the PCU to ECU request times, the microcontroller's physical reprogramming times, the ECU to PCU response times and the PCU processing time for UDS have to be added.

$$t_{\text{Download}} = \sum t_{\text{Request}} + \sum t_{\text{PhysicalProgramming}} + \sum t_{\text{Response}} + \sum t_{\text{PCUprocessing}}$$

Download time (t_{DL}) for a segmented request and a unsegmented response

Based on formula 4.1-17 the delays for $t_{\text{PhysicalProgramming}}$, t_{Response} and $t_{\text{PCUprocessing}}$ have to be added.

$$t_{\text{DL}} = \left(\left(n_{\text{PDU}s} \cdot d_{\text{PDU_ISO11898}} \cdot t_{\text{bit}} \right) + \left(n_{\text{CF_PDU}} - \left\lfloor \frac{n_{\text{CF_PDU}}}{\text{BS}} - 1 \right\rfloor - 1 \right) \cdot t_{\text{STmin}} + \left\lfloor \frac{n_{\text{CF_PDU}}}{\text{BS}} \right\rfloor (t_{\text{N_Br}} + t_{\text{N_Cs}}) \right) + t_{\text{PhysProg}} + (1 \cdot d_{\text{PDU_ISO11898}} \cdot t_{\text{bit}}) + t_{\text{PCU_Process}} \quad (4.1-18)$$

If more data shall be transferred and reprogrammed than can be transmitted by a single segmented data transfer (CAN: max 4095 byte) the download sequence is repeated until all data are transmitted to the ECU. The request time is calculated as.

$$t_{\text{DL}} = \sum_{x=1}^{x=n-1} t_{\text{DL}(x)} + t_{\text{DL}(n)} \quad (4.1-19)$$

It has to be distinguished between all previously transmitted downloads and the last download, because the last download might have less than the maximum possible data to transmit. The number of repetitions (n_{DL}) is calculated by:

$$n_{\text{DL}} = \left\lceil \frac{d_{\text{PDU_ISO14229}}}{d_{\text{SDU_ISO15765-2}}} \right\rceil$$

The final request's SDU size is calculated by:

$$d_{SDU(n)} = d_{PDU_ISO14229} \bmod d_{SDU_Iso15765-2} \quad (4.1-20)$$

Hence the total download time via CAN and ISO 15765-2 is calculated by:

$$t_{DL} = \left[\frac{d_{PDU_ISO14229}}{d_{SDU_Iso15765-2}} \right] \cdot \left(\begin{array}{l} \left(n_{PDU_s(x)} \cdot d_{PDU_ISO11898} \cdot t_{bit} \right) + \\ \left(n_{CF_PDU(x)} - \left\lfloor \frac{n_{CF_PDU(x)}}{BS} - 1 \right\rfloor - 1 \right) \cdot t_{STmin} + \\ \left\lfloor \frac{n_{CF_PDU(x)}}{BS} \right\rfloor (t_{N_Br} + t_{N_Cs}) + \\ t_{PhysProg(x)} + \\ (1 \cdot d_{PDU_ISO11898} \cdot t_{bit}) + \\ t_{PCU_Process} \end{array} \right) + \left(\begin{array}{l} \left(n_{PDU_s(n)} \cdot d_{PDU_ISO11898} \cdot t_{bit} \right) + \\ \left(n_{CF_PDU(n)} - \left\lfloor \frac{n_{CF_PDU(n)}}{BS} - 1 \right\rfloor - 1 \right) \cdot t_{STmin} + \\ \left\lfloor \frac{n_{CF_PDU(n)}}{BS} \right\rfloor (t_{N_Br} + t_{N_Cs}) + \\ t_{PhysProg(n)} + \\ (1 \cdot d_{PDU_ISO11898} \cdot t_{bit}) \end{array} \right) \quad (4.1-21)$$

Note that after the final response no additional PCU processing time is required. The formula 4.1-21 depicts also, that the data transfer rate will decrease once more because of the additional delays $t_{PhysicalProgramming}$, $t_{Response}$ and $t_{PCUProcessing}$. The programming time can be compensated by the double buffered data transfer approach of chapter 3.

4.1.4 Conclusion

The analysis ahead provides several impact parameters for a fast data transfer via a CAN protocol stack based on ISO11898 and ISO15765-2. The analysis furthermore depicts that a singular optimisation of one parameter provides under some circumstances no benefit at all. To reach the optimum data transfer rate an optimised parameter set for all corresponding layers and protocols is necessary.

Increasing baud rate

Increasing the baud rate is a good approach to accelerate data transfer but if ISO 15765-2 *Flow Control* parameter *STmin* is greater than 0.4 ms, the increased bandwidth has no impact on the overall performance. If ISO 15765-2 *Flow Control* parameter *Block Size*

(BS) is less than 12 a higher CAN baud rate will provide only small effects. Finally, the maximum possible bandwidth depends also on the system's cable length, and the system's dimensions have to be taken into account.

The different system delays have an impact on the total data transfer rate. The longer those delays are the smaller the effect of increasing bandwidth is. This has to be taken into account.

CAN identifier and stuffing bits

The data transfer performance depends on the CAN identifier size. A 29 bit CAN identifier PDU requires 18+2 bit more protocol overhead compared to 11 bit identifier PDU ($\approx 15\%$). CAN identifier shall be configured to 11 bit if the number of possible addresses for the network is sufficient.

The stuffing bit mechanism has a big impact on the total PDU length but cannot be calculated because it depends also on the payload. Hence, for further discussion unique CAN PDUs are assumed with a total PDU length of 123 bit³⁷. In that case the net data rate is only 52.03%.

CAN identifier priority, arbitration and busload

The CAN identifier priority has an impact on the data transfer rate if software reprogramming communication is processed in parallel to normal system's communication. It depends at least on the total bus load and the identifier priority. To solve that problem it shall be possible to switch all other communication nodes into a silent mode³⁸. In that case the CAN identifier priority is not important for communication performance.

The critical evaluation of the feasibility to generate 100% busload shows that several commercial CAN communication interfaces (cards) are not able to generate that maximum busload for the upper bandwidths (500 kBit/s or 1.000 kBit/s). The real net data transfer rate will be less than the theoretical value because the PCU's CAN controller hardware is not able to transmit CAN PDUs with only 3 bit length inter-frame time (bus idle time). An idle time of less than 15 bit times is a realistic value. For that case the net bandwidth will decrease between 8.8% and 10.8% depending on the number of stuffing bits. Of course, if the ECU is part of a network this is also required for gateways.

³⁷ 123 bit is the average of a PDU (11 bit CAN identifier) without stuffing bits (111 bit) and a PDU with maximum stuff bits (135 bit).

³⁸ Within ISO 14229 – UDS protocol this is possible by the diagnostic service \$28 – Communication Control (refer to [ISO 14229])

Transport protocol

The impact of the transport protocol is significant. Optimisation effects on lower layers will be inoperative if the transport layer provides communication delays (e.g. separation times etc.). Low resources (e.g. buffer) also reduce the communication performance, because additional *Flow Control* PDUs are necessary to control the data flow. The impacts of the parameters *minimum Separation Time (ST_{min})* and *Block Size (BS)* configuration are more important than the initial baud rate on the physical layer.

Best results for communication via CAN are only possible if all above discussed parameters and their influences are taken into account during communication system's design.

4.2 FlexRay

FlexRay is currently the mostly used time triggered protocol within the automotive area. Within this chapter FlexRay is introduced and the FlexRay protocol stack (layer 1 - layer 4) is analysed with focus on software reprogramming purpose. The possibilities to accelerate the data transfer via FlexRay are discussed as well as the main influencing parameters for the net data transfer rate.

4.2.1 FlexRay (FlexRay Specification 2.1)

"The FlexRay Communications System is a robust, scalable, deterministic, and fault-tolerant digital serial bus system designed for use in automotive applications" [Fle11]. FlexRay is a time triggered protocol specified by the FlexRay Consortium³⁹. The protocol is specified for a bandwidth up to 10 Mbit/s. The FlexRay Communications System Specification 2.1 was released in 2005. The revision 3.0 is currently standardised within the ISO. W. Zimmermann and R. Schmidgall provide a technical introduction in [Zim10-9]. M. Rausch also gives an introduction to FlexRay with detailed information also on hardware implementations and synchronisation mechanisms [Rau07].

Time triggered mechanism

Within a time triggered communication protocol data transmission is only possible within a well defined time slot. Only one exclusive sender is allowed to transmit data within a time slot. If two senders try to get concurrent access to the network within the same slot, a communication error occurs (data collision). To prevent the network from data collisions a

³⁹ "FlexRay Consortium: A cooperation of leading companies in the automotive industry, from the year 2000 to the year 2009. The FlexRay Consortium has concluded its work with the finalization of the FlexRay Communications System Specifications Version 3.0" [Fle11].

network wide common communication plan (communication schedule) defines the sender-to-slot arrangement.

A precondition for time triggered communication systems is a network wide common clock. FlexRay provides mechanisms to synchronise this clock on each node via the network.

A FlexRay communication slot is uniquely defined by its slot identification number (slot ID). Only if the defined slot ID occurs, a node is allowed to transmit a PDU whereas a transmitted PDU can be received by all connected network nodes.

FlexRay Schedule

Figure 4.2-1 depicts an abstract FlexRay schedule. The FlexRay schedule is divided into four sub-segments: 1) a *static segment*, 2) a *dynamic segment*, 3) a *symbol window segment (SW)* and 4) a *network idle time (NIT)*.

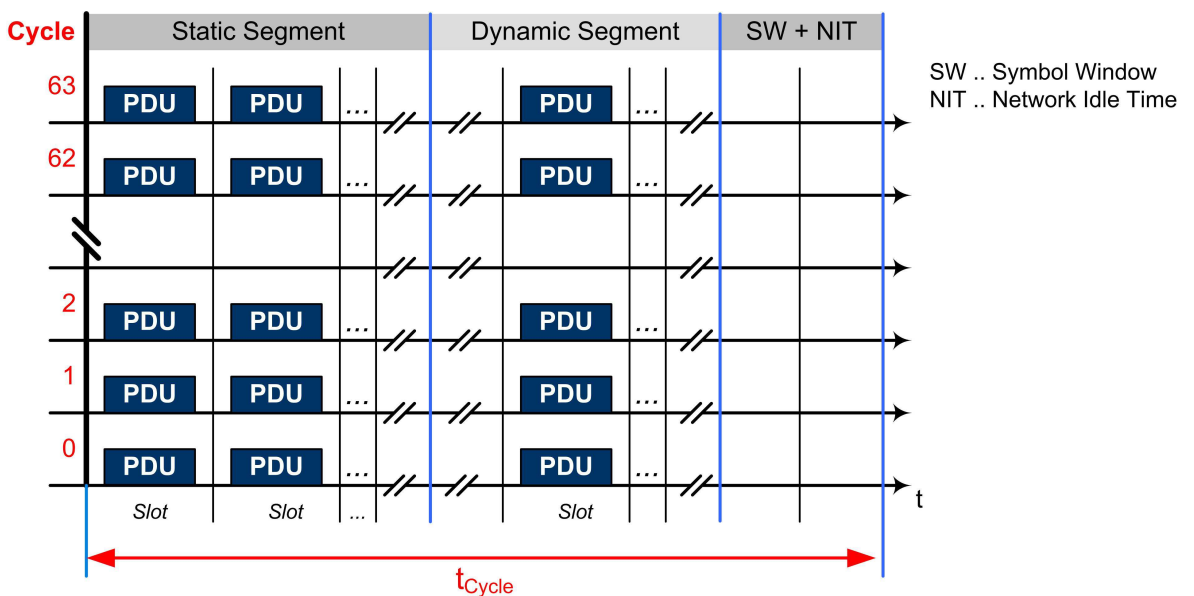


Figure 4.2-1: FlexRay Schedule

The static and the dynamic segments are defined for data communication. The symbol window segment (SW) is optional configurable and is for network function monitoring. The network idle time segment (NIT) is reserved for the nodes to calculate and applied clock correction.

All segments are transmitted within one communication cycle. The complete schedule defines 64 consecutive communication cycles.

Data transfer is only possible within the static and the dynamic segments. Hence, in the following sections and figures only these segments are illustrated. The SW segment and

the NIT segment are not used for data transfer and have therefore not been taken into account for data transfer acceleration.

Within the static segment a PDU shall be sent within each slot. The idea behind the static segment is to provide a deterministic communication system with equidistant data transmission.

Within the dynamic segment a PDU is only sent if data for transmission are available. If no transmission shall be processed the sender node transmits no PDU and after a defined timeout all connected network nodes will switch to the next slot ID. This segment is basically defined for event triggered communication as given within the CAN protocol (refer to section 4.1).

Static and dynamic segments are subdivided into small communication sections (slots). Within each slot a FlexRay PDU can be transmitted whereas each slot within the global communication schedule is exclusively allocated to exactly one sender node.

FlexRay PDU

All FlexRay PDUs have the same structure and are able to transmit up to 254 byte payload. The PDU structure in detail is given in [Zim10-10] and in [Fle05]. Table 4.2-1 depicts the different parts of a FlexRay PDU for static and dynamic segment's communication and their corresponding length.

Table 4.2-1: FlexRay PDU length

Definition	length	unit
Header	40	bit
Trailer	24	bit
Transmission Start Sequence TSS	3..15	bit
Frame Start Sequence FSS	1	bit
Frame End Sequence FES	2	bit
Payload	0..2032	bit
Sum	70 .. 2104	bit

The total length on a FlexRay PDU is calculated as [Zim10-9]:

$$n_{FR-PDU} = \frac{10}{8} (n_{Header} + n_{Trailer} + n_{Payload}) + n_{TSS} + n_{FSS} + n_{FES}$$

The factor 10/8 is necessary because the physical layer insert a 2 bit long *Byte Start Sequence* (BSS) between each byte of the FlexRay PDU. For the following discussions in summary 10 bit are assumed for n_{TSS} , n_{FSS} and n_{FES} . Hence, the calculation for the FlexRay PDU length is:

$$n_{FR-PDU} = \frac{10}{8}(64 + n_{Payload}) + 10 \quad (4.2-1)$$

Table 4.2-2 depicts the length of a FlexRay PDU depending on the payload size ($n_{Payload}$) and the corresponding PDU runtime T_{FR-PDU} within a 10 MBit/s FlexRay network.

Table 4.2-2: FlexRay PDU length and PDU runtime for $f_{bit}=10$ MBit/s

Payload per PDU ($n_{Payload}$)		FlexRay PDU Length (n_{FR-PDU})	FlexRay PDU runtime (T_{FR-PDU})
8 Byte	64 bit	170 bit	17 μ s
16 Byte	128 bit	250 bit	25 μ s
32 Byte	256 bit	410 bit	41 μ s
42 Byte	336 bit	510 bit	51 μ s
64 Byte	512 bit	730 bit	73 μ s
128 Byte	1024 bit	1370 bit	137 μ s
254 Byte	2048 bit	2630 bit	263 μ s

FlexRay network communication and addressing mode

The FlexRay addressing mechanism is the slot/cycle assignment within the common FlexRay schedule. As described above a sender is only allowed to transmit data if the corresponding slot ID occurs within the correct cycle number. Each ECU knows the slot/cycle combination of the relevant senders and receives the data within these slots. No additional addressing mechanism is defined within the FlexRay specification 2.1. As a result of that mechanism, the bandwidth for a single communication link is limited. For each communication link slots have to be allocated for the sender node, which are not available for another communication link, e.g. for reprogramming communication between a PCU and the corresponding ECU. If several ECUs are connected to the FlexRay network, several slots have to be allocated for PCU's reprogramming communication link. Hence, the maximum possible net data rate for the communication between the PCU and the ECU depends significantly on the number of allocated slots for that link within the global schedule.

FlexRay net data transfer rate

As depicted in Figure 4.2-1 the available or usable bandwidth depends on the overall system schedule and the number of slots allocated for the data transfer link of a network node. A formula was developed to calculate the net data transfer rate:

$$f_{\text{Data_net}} = \frac{\sum_{x=0}^{x=63} n_{\text{Payload_Cycle}_x}}{64 \cdot t_{\text{CycleTime}}} \quad (4.2-2)$$

If the FR-PDUs have the same length the formula can be simplified:

$$f_{\text{Data_net}} = \frac{n_{\text{Payload}} \cdot X_{\text{FR-PduPerCycle}}}{t_{\text{CycleTime}} \cdot r_{\text{PduCycleRepetition}}} \quad (4.2-3)$$

FlexRay distinguishes between the static and the dynamic segment. Hence the formula above can be used only within strict timing boundaries:

(1) For communication within the static segment:

$$\frac{n_{\text{FR-PDU}}}{f_{\text{Bit}}} \cdot X_{\text{FR-PduPerCycle}} < t_{\text{staticSegment}} \quad (4.2-4)$$

(2) For communication within the dynamic segment:

$$\frac{n_{\text{FR-PDU}}}{f_{\text{Bit}}} \cdot X_{\text{FR-PduPerCycle}} < t_{\text{dynamicSegment}} \quad (4.2-5)$$

Summary

The FlexRay schedule definition has a significant impact on the performance of the different communication links. With focus on software reprogramming best performance occurs if 100% busload on a single communication link between the PCU and an ECU is given. Unfortunately, a slot can be allocated only once for exactly one sender, and the number of slots is limited. That means an allocated slot for another link is not usable for the software reprogramming communication. In contrast to the CAN bus system where additional CAN nodes can be added to an existing network without high effort, an additional FlexRay node can only be included if the communication behaviour is considered within the global schedule. If the slots for data transmission are not allocated at schedule design time, the introduction of a new FlexRay node is only possible if also a new schedule is introduced (at least for the corresponding communication partners). That means that each communication link performance is defined at schedule definition time and can not be changed within the finalised schedule.

Therefore two different questions are discussed:

- 1.) What is the maximum performance of a FlexRay communication link and what are the main influencing parameters?
- 2.) Are there possibilities to optimise a communication link performance within an existing system?

4.2.1.1 Communication performance

With focus on software reprogramming the crucial question is how to generate the maximum net data rate on a communication link. Formula 4.2-3 provides four possible approaches to accelerate data transfer and increase the net data rate:

- 1) Configure a cycle repetition equal to one for all PDUs
- 2) Increase FlexRay-PDU's payload
- 3) Increase the number of PDUs per cycle
- 4) Decrease the cycle time

Cycle Repetition

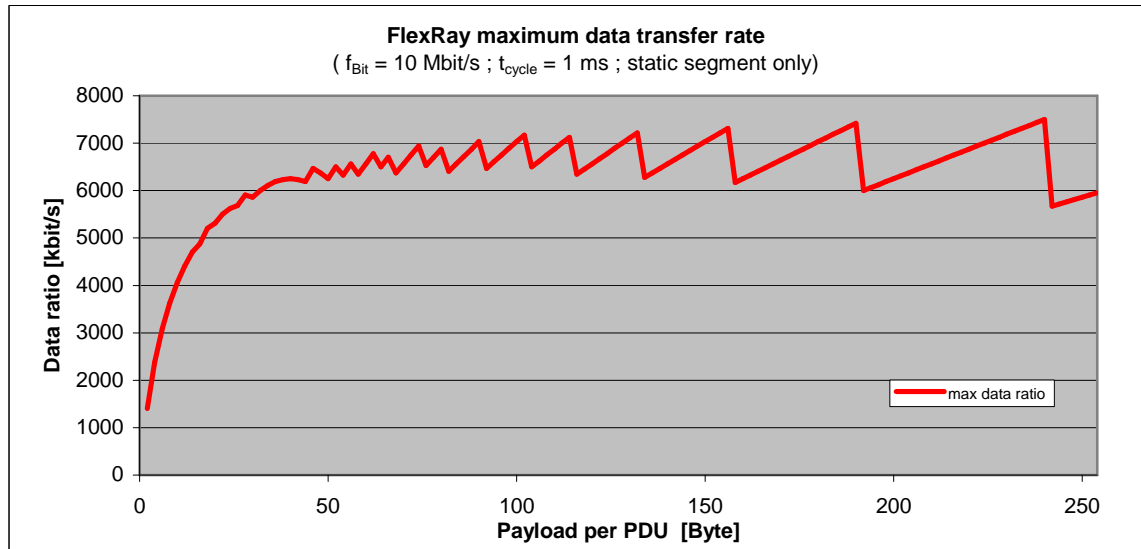
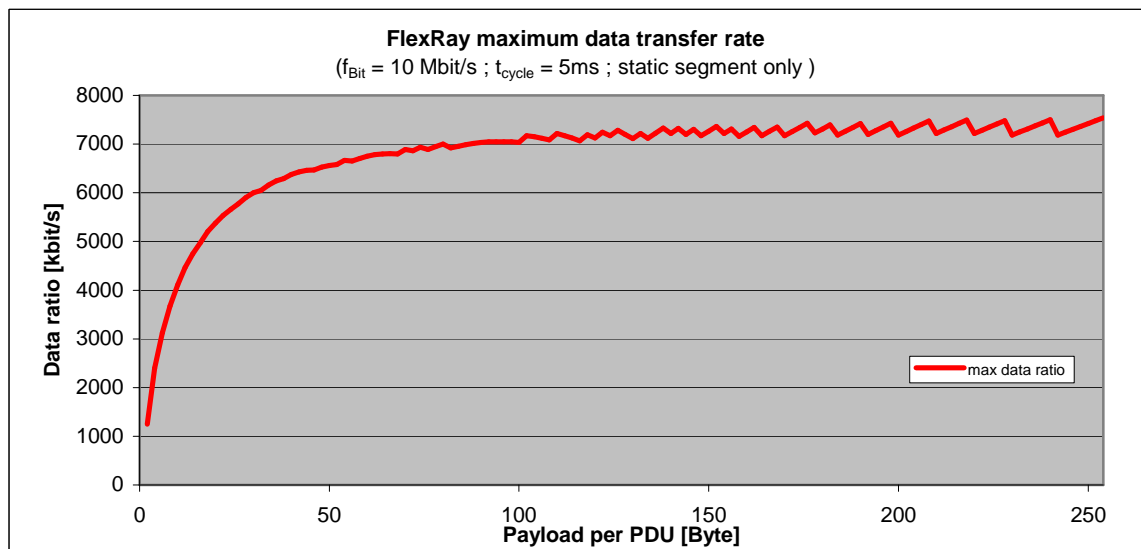
As depicted in figure 4.2-1 the global FlexRay schedule provides 64 cycles. If the schedule is designed to transmit a PDU not within each cycle (*cycle repetition* = 1), a data transmission gap will occur with a delay of $t_{\text{TransmissionDelay}} = (r_{\text{PduCycleRepetition}} - 1) * t_{\text{Cycle}}$. Due to the crucial requirement to generate 100% busload for software reprogramming communication, the cycle repetition for a data transfer PDU shall be configured equal to one ($r_{\text{PduCycleRepetition}}=1$). Hence it is guaranteed that data transfer within each communication cycle is possible.

FlexRay PDU payload

A FlexRay PDU is able to transmit at least 2 Byte and in maximum 254 Byte payload. A cycle repetition of 1 is assumed for all PDUs (data transfer within each cycle is possible). Furthermore it is assumed that all slots of the communication cycle are allocated for this data transfer (e.g. from a PCU to an ECU) whereby for a simplification only a static segment is configured⁴⁰.

Figure 4.2-2 and figure 4.2-3 depict the maximum net data rate for the different possible payload lengths (2 Byte – 254 Byte). This approach has to be analysed within the limits of formula 4.2-4 and 4.2-5. The cycle time is varying from 1ms to 5 ms.

⁴⁰ This is only a simplification to illustrate the impact of the different payload lengths for the overall net data transfer rate.

Figure 4.2-2: Maximum FlexRay net data rate ($t_{\text{cycle}} = 1 \text{ ms}$)Figure 4.2-3: Maximum FlexRay net data rate ($t_{\text{cycle}} = 5 \text{ ms}$)

In both figures the same effect is visible. The ratio between payload and protocol overhead results in a lesser data transfer rate even though all available slots are in use. The peaks occur because of the time limitation of the cycle time. If the last PDU requires more transmission time than the residual cycle time, the PDU transmission is skipped. Hence, the resulting data transfer rate is less than the rate with the previous payload length. There is a direct dependency between the maximum payload length and the communication cycle time. Especially figure 4.2-2 depicts that the best result is not obligatory given by the maximum payload length.

This effect is less important if the cycle time is higher (refer to figure 4.2-3). Best performance will be given if 1) the PDU's payload is configured to 254 Byte (this reduces the

impact of protocol overhead) and 2) the communication cycle time is configured to exactly n-times of PDU runtimes⁴¹.

Both configurations depict a net data transfer rate upper limit at nearly 7500 kbit/s.

FlexRay PDUs per cycle

As discussed above, best performance is given if each slot per cycle can be used for data transmission. The impact of the ratio between communication cycle time, the payload length and the number of usable PDUs (slots) per cycle is depicted in figure 4.2-4 and figure 4.2-5.

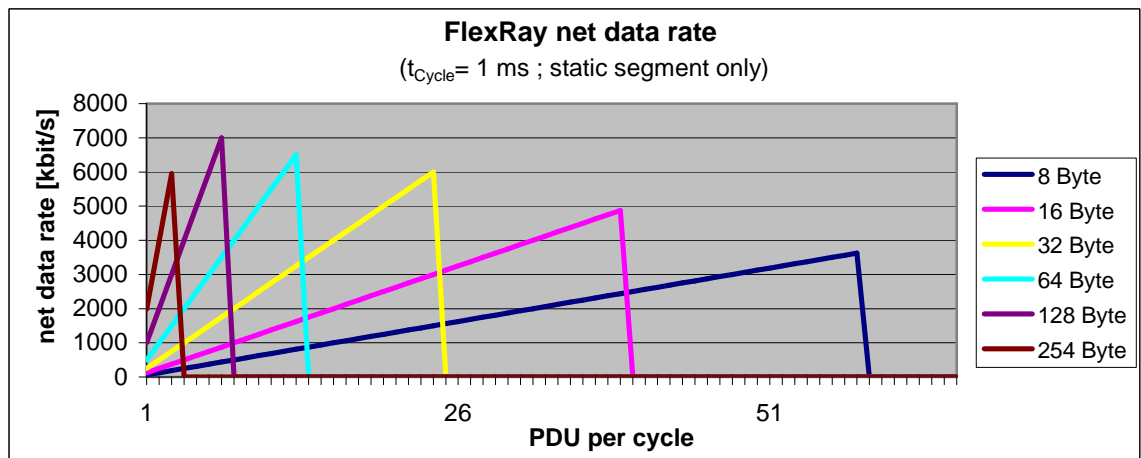


Figure 4.2-4: FlexRay net data rate ($t_{\text{cycle}} = 1 \text{ ms}$)

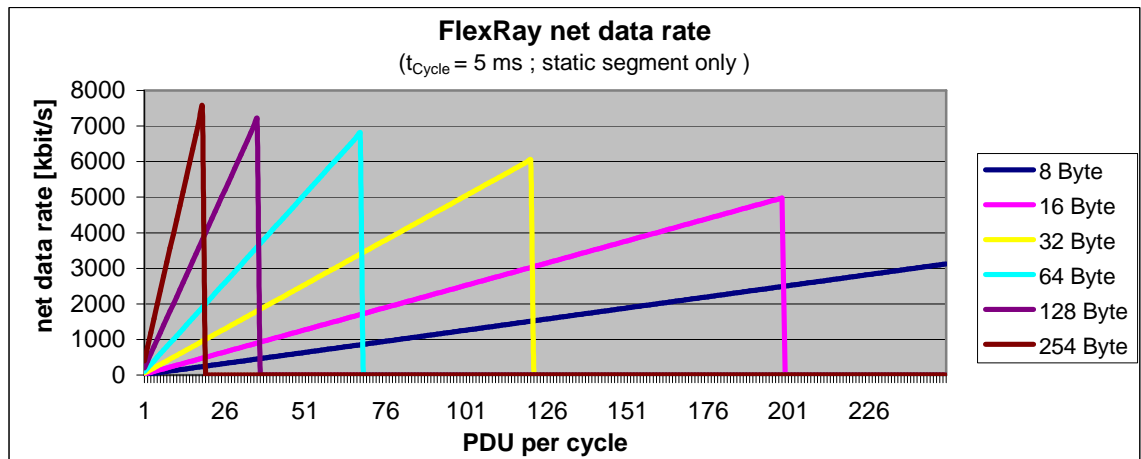


Figure 4.2-5: FlexRay net data rate ($t_{\text{cycle}} = 5 \text{ ms}$)

Depending on formula 4.2-4 and 4.2-5 limitations occur because the communication cycle time allows only the usage of a limited number of PDUs.

⁴¹ For the communication cycle time calculation the symbol window segment time and network idle time have also to be taken into account.

Figure 4.2-4 depicts a configured communication cycle time of 1ms. Within that time slot only 3 PDUs with 254 byte payload can be transmitted. If the payload is 128 byte, seven PDUs can be transmitted and the 8th PDU is skipped by time restriction. Hence, the configuration of less payload PDUs provides better performance on that communication cycle time configuration (1ms.). If the communication cycle time is increased (e.g. 5ms) the large payload PDUs provide better performance.

Conclusion

Even though the FlexRay data link layer provides a 10 MBit/s bandwidth the net data rate on that layer has a wide spreading and could be in worst case significantly lower.

The PDU payload size, the number of PDUs per cycle, the cycle repetition time for a PDU and the communication cycle length are the essential parameters. As depicted in formula 4.2-3 and discussed above, several different parameter sets will provide equal performance values.

The most important parameter for schedule design is the number of connected nodes and their required transmission slots. By this value and the required cycle time the payload length limitation for each slot is given. For the schedule design several degrees of freedom exist. Different value combinations of PDUs per cycle and slot's cycle repetition will provide equal performance results.

To generate maximum performance it is important to have no transmission gap at a segment's end because PDU's runtime is out of segments time. Hence, PDU's total runtime, i.e. combination of PDU's payload length and the number of allocated slots, shall be in relation to the cycle or segment time.

In best case configuration a net data transfer rate of 7.500 kBit/s is possible (75% of gross data rate). For a CAN bus system in comparison this net data rate is only on 52% (refer to section 4.1.4).

Theoretically, the overall performance of FlexRay could be doubled if the second channel was available as defined within FlexRay specification. In that case data transmission could be processed via channel A as well as via channel B. Unfortunately, currently there is no microcontroller available that supports a second channel. The theoretical maximum net data transfer ratio of ≈ 15.000 kBit/s (7.500 kBit/s on channel A and B) cannot be evaluated in practice.

4.2.1.2 Schedule optimisation

As mentioned above, FlexRay's schedule design has to be done within the network development phase. If the schedule is fixed, changes on the schedule result in high effort.

Also the available slot resources have to be shared between normal ECU's functional communication and the reprogramming communication from a PCU to the corresponding ECU.

An approach to accelerate the reprogramming data transfer is to switch into a second, optimised schedule. That means that each ECU supports two different schedules, one for the normal ECU functional communication and one for the special scenario of software reprogramming. A well defined trigger event (e.g. reception of a diagnostic message etc.) initiates the schedule switching process. A second trigger event (e.g. a second diagnostic service, power-on reset) initiates switching back to the default schedule.

Figure 4.2-6 depicts possible approaches to optimise a FlexRay schedule with the aim to accelerate data transfer for software reprogramming purpose.

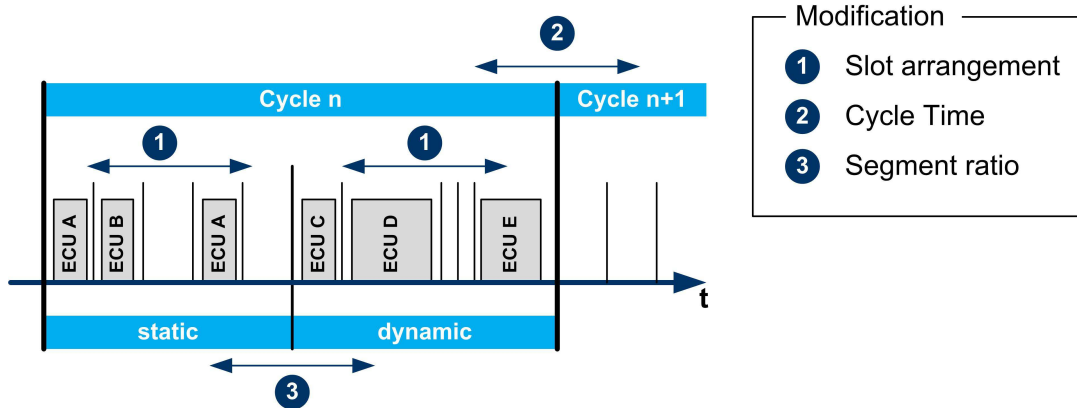


Figure 4.2-6: FlexRay schedule optimisation

There are three possible approaches to modify a FlexRay's schedule:

- 1) Slot arrangement modification
- 2) Cycle time modification
- 3) Relation of static to dynamic segment

Approach 1: slot arrangement modification

Slot arrangement modification will change the assignment of slots to the corresponding ECUs and the length of the slots. For the reprogramming scenario the normal application communication slots are not necessary and therefore they could be skipped from the schedule. The resulting bandwidth is used to expand the remaining slots. By this approach a) more slots could be used for reprogramming communication and b) within these slots it is possible to transmit more data. The risk to shut down the communication system is less because only the interpretation mask for the received PDUs has to change and no global timing values are influenced or modified. Especially for FlexRay each PDU signal the start and the end of a frame by a special bit pattern (*Start-Of-Frame*, *End-Of-Frame*).

Hence, the FlexRay communication controller hardware can identify the real PDU length. In case the software analysis detects an invalid length for the expected PDU this mismatch could be handled without any impact on FlexRay's physical communication.

Approach 2: cycle time modification

The modification of the communication cycle time, i.e. the basic schedule timings, provides the possibility to design a schedule that maps perfectly to the given reprogramming scenario. As described in section 4.2.1.1 it might be possible that the full bandwidth is not used because last PDU's length requires more time to transmit than the remaining cycle time. If the schedule is optimised to the corresponding PDU lengths the maximum performance (net data rate) is possible.

The risk of this approach is higher compared to the slot arrangement approach because the communication system has to be shut down, reinitialised for the new cycle time and restarted. If one or more ECUs do not restart with the new cycle time, a synchronisation is not possible and the network will not get a stable state.

Approach 3: relation of static and dynamic segment

Whereas the upper approaches have been common to all time triggered communication systems, the third approach is FlexRay specific and modifies the relation of static and dynamic segment.

This approach solves the main disadvantage of the FlexRay specification: the addressing mode which is given only by the slot-ID and communication cycle number. As discussed above, best performance is given if all available slots are mapped to one communication link. Even if the schedule is modified, either the different links to each connected ECU are defined in that new schedule, or for each communication link to an ECU an individual schedule is defined. As a result the PCU or the gateway, in case the PCU is not connected directly to FlexRay bus, has to deal with several schedules. This challenge is partly solved if a small static segment (to guarantee system's synchronisation) and a large dynamic segment are defined. The dynamic segment based on the event driven approach requires no permanent PDU transmission. If the sender (PCU or Gateway) does not transmit data within a dynamic slot, the FlexRay system, i.e. all ECUs on the bus, switch to the next slot-ID after a small delay. In opposite to the static segment, where a PDU with full PDU-length has to be sent, the communication in the dynamic segment reduces the unusable bandwidth to the minimum defined by physical protocol.

Conclusion

Data acceleration approaches by switching to an optimised schedule for software reprogramming scenarios provides high potentials and the net data rate can be significantly

increased. On the other hand the modification of basic schedule timings provides a high risk if communication system's synchronisation gets lost if a communication node does not switch to the new schedule. The approach to modify only slot arrangement provides an increasing performance without the risk of losing synchronisation because basic system timings are not modified.

4.2.2 FlexRay Transport Protocol (ISO 10681-2)

This section introduces the FlexRay transport layer protocol according to ISO 10681-2, analyses different approaches to accelerate data transfer via this protocol and discusses the impact of the different protocol parameters.

Introduction

As described in section 4.2.1 the basic communication schedule has a big impact on the overall data transfer performance. Software reprogramming communication shall work on the same network as normal ECU communication but should not require resources (slots) when not in use. Unfortunately, for time triggered systems it is necessary to reserve bandwidth within the basic communication schedule that is not usable for normal ECU communication. With focus on software reprogramming purpose based on the diagnostic protocol UDS as defined in ISO 14229, at least one slot has to be reserved for a diagnostic service request to the ECU (exclusive access for PCU to ECU communication) and at least one slot for the diagnostic service response (exclusive access for ECU to PCU communication). Hence each slot is exclusively dedicated to a PCU-ECU connection. For a network with e.g. 20 nodes at least 40 slots have to be reserved within the global schedule. This example shows the main disadvantage of the FlexRay system: the address mechanism based on the slot-ID and cycle number combination. Even if diagnostic communication is processed only within the dynamic segment a slot could not be shared between different communication links. It was one of the main challenges for FlexRay communication to find a mechanism that provides the necessary performance for software reprogramming as well as reduces the required slot resources. In 2007/2008 an ISO standardisation group was established with the aim to standardise a FlexRay transport layer protocol that fulfils the above described communication requirements (TC22/SC3/WG1/TF13 ISO TP on FlexRay)⁴². The transport layer protocol for FlexRay

⁴² R. Schmidgall is a member of this ISO standardisation group TC22/SC3/WG1/TF13 "ISO TP on FlexRay". The ISO 10681-2 specification was introduced in AUTOSAR 4.0 (refer to [AUTOSAR]). R. Schmidgall took the document ownership for the FlexRay Transport Protocol (FrTp) according to ISO 10681-2 (AUTOSAR document ID 029). In 2010 the AUTOSAR steering committee has decided that ISO 10681-2 shall be available also for AUTOSAR 3.2 (previous AUTOSAR version to AUTOSAR 4.0).

was standardised in 2009 as international standard [ISO10681-2]. The protocol is called “communication layer” protocol, because it provides services and mechanisms of layer 4 as well as of layer 3. It is also independent from the underlying physical layer FlexRay protocol 2.1 or 3.0 (refer to section 4.2.1). The communication protocol works for communication in the static segment as well as for communication in the dynamic segment.

PDU according to ISO 10681-2

For software reprogramming large data have to be transmitted. The maximum payload length of a FlexRay PDU can be configured up to 254 byte. Hence, a transport protocol is necessary to transmit data packages with more than the configured PDU payload length. Also a flow control is required to control the data stream on the bus. The maximum payload length that can be transmitted by a segmented data transfer is up to 65.535. The limit is given by the 16 bit *MessageLength* parameter within the *StartFrames*'s PCI.

According to the initial requirements of a more flexible addressing mechanism the FlexRay communication protocol defines a target and a source address field. The *PCI* field and the payload are also part of the PDU. Figure 4.2-7 depicts the PDU according to ISO 10681-2 specification.

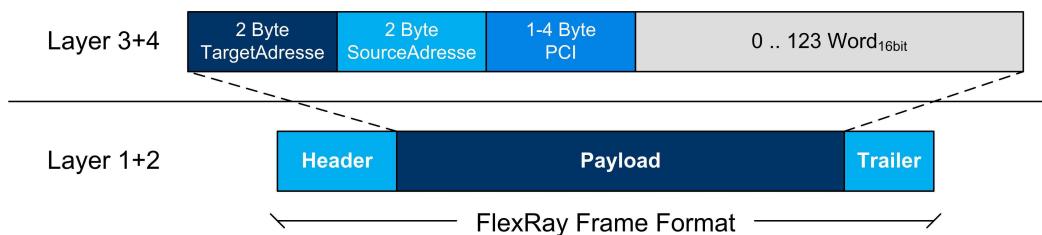


Figure 4.2-7: FlexRay communication protocol PDU format

The communication layer PDU is mapped to the FlexRay's payload field. Table 4.2-3 depicts the different PDU-types and the corresponding address information fields, the PCI length and payload lengths.

Table 4.2-3: ISO 10681-2 PDU overview

PDU Type	Address	PCI	Max. possible payload n_{PL}
StartFrame (STF)	4 Byte	4 byte	$FR\text{-}PDU\text{-}8 \leq n_{PL_SF} \leq 246$ Byte
ConsecutiveFrame (CFx)	4 Byte	2 byte	$FR\text{-}PDU\text{-}6 \leq n_{PL_CF} \leq 248$ Byte
LastFrame (LF)	4 Byte	4 byte	$FR\text{-}PDU\text{-}8 \leq n_{PL_LF} \leq 246$ Byte
FlowControl (FC)	4 Byte	1 byte	---

The protocol overhead is at maximum 8 bytes. Therefore the maximum payload that can be transmitted by one FlexRay PDU is up to 246 byte (123 words_{16bit}). ISO 10681-2 uses nearly the same protocol mechanisms as the ISO 15765-2 CAN Transport Protocol.

Address mechanism

By the new defined source and target address fields it is possible to define several different connection links mapped on the same slot-IDs. For example, a PCU allocates four slots for data transmission. All connected ECUs are configured to receive data on these slots (broadcast connection). If the target address matches with their own address, they have to process the received PDU. If the address doesn't match, the PDU shall be skipped. However, a response slot must be configured for each ECU. By this method the required number of slots could be reduced significantly because all slots of the 1:n connection (e.g. PCU to ECUs) could be shared. Figure 4.2-8 depicts the possible communication scenarios.

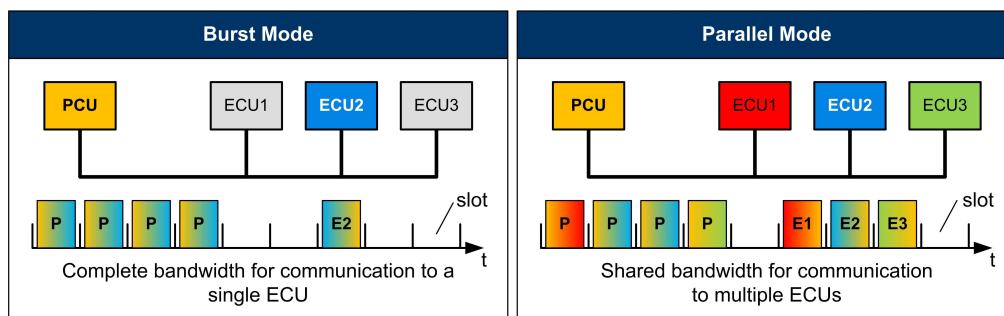


Figure 4.2-8: FlexRay communication layer scenarios

The specified address mechanism supports a burst mode (all bandwidth is used for communication with one ECU) as well as a parallel mode (shared bandwidth for multiple ECUs). A burst mode could be necessary if only one ECU within the network should be reprogrammed. The parallel mode is used if several ECUs shall be reprogrammed. Without the flexible bandwidth assignment the bandwidth for a burst mode to each ECU has to be statically allocated within the basic FlexRay schedule. The bandwidth could be assigned dynamically and flexible in a range as depicted in figure 4.2-9.

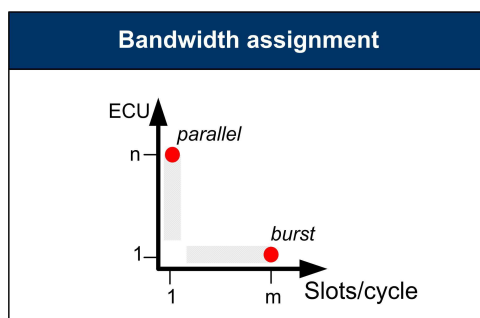


Figure 4.2-9: FlexRay Bandwidth assignment

4.2.2.1 Communication performance

The communication protocol according to ISO 10681-2 supports unsegmented and segmented data transfer. Figure 4.2-10 depicts those scenarios.

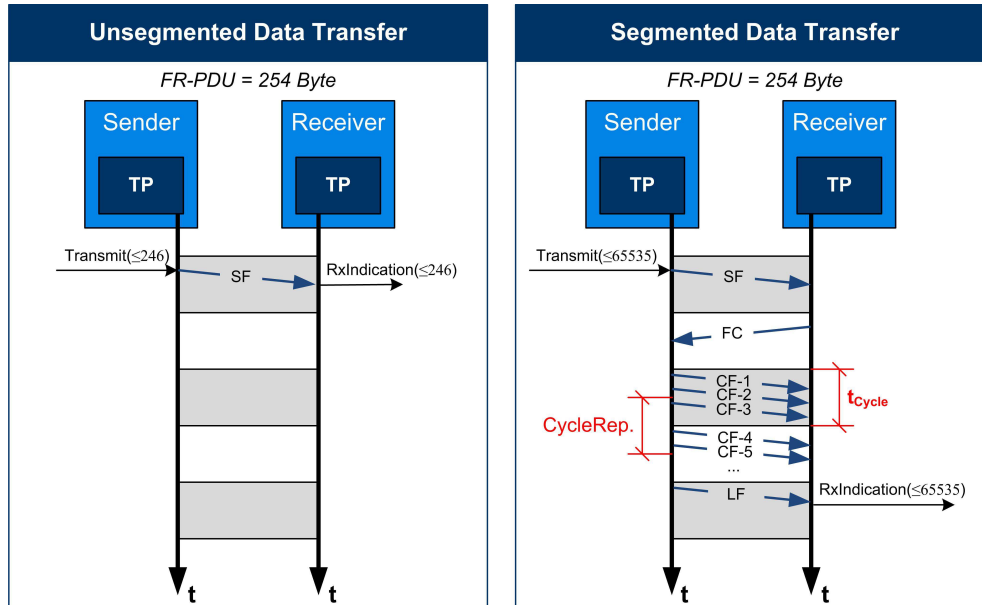


Figure 4.2-10: Data transfer according to ISO 10681-2

For unsegmented data transmission only a *Start Frame (SF)* is transmitted. A segmented data transfer requires more PDUs within a defined sequence. After a connection link has been established by a PCU's *Start Frame (SF)* the ECU has to send a *Flow Control (FC)* PDU before the PCU continues the data transfer by sending *Consecutive Frame (CF)* PDUs. In contrast to the CAN TP a final *Last Frame (LF)* PDU terminates the connection. In spite of the smart address method the basic FlexRay communication schedule has an impact on the protocol's communication performance especially for segmented data transmission. The flow control PDU is sent by the initial receiver node and until no flow control PDU is received by the initial sender no additional consecutive PDU is transmitted. Therefore ECU's response slot position and the cycle repetition within the global FlexRay schedule is very important. The communication layer protocol supports several protocol configuration possibilities for data transmission. To accelerate the data transfer some conditional protocol parameters have to be fixed:

- a) No additional *Flow Control* PDU but the first one shall be sent. Hence, the possible delay time without data transmission is reduced.
Therefore, the configuration option to transmit a PDU type *ConsecutiveFrame_EOB* (End of Block) must be disabled.
- b) The *Cycle Repetition (CR)* for all PDUs is equal to 1. This guarantees a data transmission within each cycle.

Total data transfer time for a single, segmented data transfer

For total data transfer time calculation some formulas have been developed which takes the ISO 10681-2 protocol behaviour into account.

The net data rate is calculated by the SDU size divided by the transfer time.

$$f_{\text{netDataRate}} = \frac{n_{\text{SDU}}}{t_{\text{Transfer}}}$$

The transfer time t_{Transfer} is the sum of all FlexRay cycle times required to transmit all the data according to the ISO 10681-2 protocol.

$$t_{\text{Transfer}} = x \cdot t_{\text{Cycle}}$$

The number of cycles (x) is calculated by the initial *Start Frame (SF)* cycle plus the required cycles to transmit the *Flow Control (FC)* PDU plus the required cycles to transmit all the payload data by *Consecutive Frame (CF)* PDUs and the *Last Frame (LF)* PDU. The *Cycle Repetition (CR)* is assumed equal to 1. The *Maximum Number Of PDUs Per Cycle (MNPC)* parameter depends on the basic schedule time and shall be within the limitations as defined in section 4.2.1.

$$t_{\text{Transfer}} = \left(1_{(\text{SF})} + \text{CR}_{(\text{FC})} + \left\lceil \frac{\frac{n_{\text{SDU}} - n_{\text{PL-SF}}}{n_{\text{PL-CF}}}}{\text{MNPC}} \right\rceil \cdot \text{CR}_{(\text{CF})} \right) \cdot t_{\text{cycle}} \quad (4.2-6)$$

Hence, the net data rate can be calculated as:

$$f_{\text{netDataRate}} = \frac{n_{\text{SDU}}}{\left(1_{(\text{SF})} + \text{CR}_{(\text{FC})} + \left\lceil \frac{\frac{n_{\text{SDU}} - n_{\text{PL-SF}}}{n_{\text{PL-CF}}}}{\text{MNPC}} \right\rceil \cdot \text{CR}_{(\text{CF})} \right) \cdot t_{\text{cycle}}} \quad (4.2-7)$$

To illustrate the result a system is assumed with $t_{\text{Cycle}} = 5\text{ms}$; $\text{CR} = 1$; FR-Payload length = 254 Byte; $\text{MNPC} = 18^{(43)}$; $n_{\text{SDU}} = 65.535$ Byte. It requires 17 FlexRay communication cycles to transmit all the data. The data rate is 771 kByte/s (6.168 MBit/s).

⁴³ Refer to table 4.2-2: FR-PDU with 254 byte payload has a runtime of 263 μs . In 5ms cycle time 19 PDUs could be scheduled. One response slot for a FC-PDU is required. Hence, a MNPC of 18 is possible and assumed.

Impact of Flow Control Parameter

The flow control configuration of ISO 10681-2 has a deep impact on the overall data transfer performance. Five different *Flow States (FS)* are defined by protocol but only the flow state *Clear To Send (CTS)* with the parameter *Bandwidth Control (BC)* influences the data transfer performance directly. The flow state *Wait (WT)* is not relevant to these considerations because that state only occurs when the system resources (buffer etc.) are in a critical state and communication can not be continued without a delay for data processing. Table 4.2-4 depicts the different Flow Control PCI bytes.

Table 4.2-4: ISO 10681-2 Flow Control (FC) PCI Overview [ISO 10681-2]

Name	Byte 1		Byte 2	Byte 3	Byte 4
	Bits 7-4 C_PCIType	Bits 3-0			
FlowControl (FC)	8	FS=CTS	BC		BFS
FlowControl (FC)	8	FS=ACK_RET	ACK		BP
FlowControl (FC)	8	FS=OVFLW			
FlowControl (FC)	8	FS=WT			
FlowControl (FC)	8	FS=ABT			
NOTE Grayed out cells are not utilized for PCI information					

Source: ISO10681-2:2009 – Table 17

Table 4.2-5: ISO 10681-2 Definition of Bandwidth Control (BC) values [ISO 10681-2_2]

Hex value		Description
Bit 7-3 MNPC	Bit 2-0 SCexp	
0	don't care	No bandwidth control In case no bandwidth control mechanism is required, bit 7-3 is set to zero (0).
1-1F	0 - 7	SCexp The sub-parameter 'Separation Cycle Exponent' represents the exponent to calculate the minimum number of 'Separation Cycles' (SC) the sender has to wait for the next transmission of a C_PDU. Formula: $SC = (2^n) - 1$ (with $n = SCexp$) which results in the following separation cycles: 0, 1, 3, 7, 15, 31, 63, 127 MNPC The sub-parameter 'Maximum Number of PDUs per Cycle' limits the number of C_PDUs the sender is allowed to transmit within a FlexRay cycle either immediately following FC C_PDU or following SC cycles after the sender has sent the previous C_PDU of the message. The adherence of MNPC prevents an overflow on the receiver side (e.g. due to Rx-buffer restraints in the receiver). Note: Both parameter can be set independently

Source: ISO10681- 2:2009 – Table 21

The *Bandwidth Control (BC)* parameter is divided into two values. The *Maximum Number Of PDUs Per Cycle (MNPC)* “limits the number of C_PDUs⁴⁴ the sender is allowed to transmit within a FlexRay cycle ..” [ISO 10681-2, table 3.2-10]. The *Separation Cycle Exponent (SCexp)* “represents the exponent to calculate the minimum number of 'Separation Cycles' (SC) the sender has to wait for the next transmission of a C_PDU”

⁴⁴ C_PDU is the nomenclature for “communication layer protocol data unit” within ISO 10681-2

[ISO 10681-2, table 3.2-10]. As depicted in formula 4.2-7 the net data rate is influenced by *MNPC* and by the cycle repetition value *CR*. *CR* is calculated by (refer to table 4.2-5):

$$CR = SC + 1 = 2^{SC_{exp}}.$$

A quantitative evaluation of formula 4.2-6 shows that a large *MNPC* results in a smaller number of FlexRay communication cycles to transmit the SDU. On the other hand a large *SC_{exp}* results in a large value for the *Cycle Repetition (CR)* and finally in a large number of required FlexRay cycles. The best performance will occur if *MNPC* has the maximum value that is possible by the given schedule and a cycle repetition of 1 which results in a *SC_{exp}* value of 1. A large cycle time t_{cycle} must not generally result in less performance because a large cycle time provides the possibility to have many PDUs per cycle if the schedule is correspondingly configured.

Summary

The FlexRay communication layer protocol ISO10681-2 solves a basic problem of time triggered communication systems: dealing with the limited resource of time slots (FlexRay slots) for communication. By introducing a source and target address field into the PDU the possibility of dynamic bandwidth assignment was given in spite of the fix bandwidth allocation in the global communication schedule. In between both boundary scenarios, burst mode (all bandwidth for communication to one ECU) and parallel mode (shared bandwidth for communication with several ECUs) an optimised bandwidth assignment for a required scenario could be configured. A disadvantage of this method is the high CPU⁴⁵ load because in [FlexRay2.1] specification no address filtering mechanism in hardware is defined. As a result the address evaluation must be implemented in software. Typically the received PDU must be processed through all underlying software layers before the address evaluation can be executed in the communication layer. The next FlexRay specification will solve that problem by the definition of a hardware filtering mechanism. This method will decrease the CPU load if the ECU is not addressed.

The impact of the underlying basic FlexRay communication schedule to the data transmission performance is apparent (refer also to section 4.2.1). Nevertheless, the influence of communication protocol configuration is important, as well, especially the number of required *Flow Control* PDUs and the flow control parameters influencing the communication net data rate.

Best results are possible if only one flow control PDU is required and the separation cycle is configured equal to one.

⁴⁵ CPU.. Central Processing Unit

4.2.3 Complete reprogramming process based on UDS

As described in chapter 2, the ISO 14229 - USD protocol is the currently most common diagnostic protocol. Reprogramming of the ECU's application software is controlled by this standard. The communication is based on a diagnostic request – diagnostic response behaviour. Due to the UDS protocol all application data which shall be transferred are segmented to smaller partitions according to the underlying bus system transport layer protocol's maximum value. In case of the FlexRay bus system and the transport layer protocol according to ISO 10681-2 this value is limited to 65535 byte maximum.

To calculate the total download time all the PCU to ECU request times, the microcontroller's physical reprogramming times, the ECU to PCU response times and the PCU processing time for UDS have to be added.

$$t_{\text{Download}} = \sum t_{\text{Request}} + \sum t_{\text{PhysicalProgramming}} + \sum t_{\text{Response}} + \sum t_{\text{PCUprocessing}}$$

Download time (t_{DL}) for a segmented request and a unsegmented response

As discussed in section 4.2.2.1 the communication layer's option to optimise the download performance by communication layer' configuration (disable sending of *ConsecutiveFrame_EOB* PDUs) is here assumed, too. Based on formula 4.2-6 the delays for $t_{\text{PhysicalProgramming}}$, t_{Response} and $t_{\text{PCUProcessing}}$ have to be added.

$$t_{\text{DL}} = \left(1 + \text{CR}_{(\text{FC_Req})} + \left[\frac{\frac{n_{\text{SDU}} - n_{\text{PL-SF}}}{n_{\text{PL-CF}}}}{\text{MNPC}} \right] \cdot \text{CR}_{(\text{CF_Req})} \right) \cdot t_{\text{Cycle}} + t_{\text{PhysProg}} + (\text{CR}_{(\text{STF_Res})} \cdot t_{\text{Cycle}}) + t_{\text{PCU_Process}} \quad (4.2-8)$$

If more data shall be transferred and reprogrammed than can be transmitted by a single segmented data transfer (FlexRay: max 65535 byte) the download sequence is repeated until all data are transmitted to the ECU. The request time is calculated as

$$t_{\text{DL}} = \sum_{x=1}^{x=n-1} t_{\text{DL}(x)} + t_{\text{DL}(n)} \quad (4.2-9)$$

It has to be distinguished between all previously transmitted downloads and the last download, because the last download might have less than the maximum possible data to transmit.

The number of repetitions (n_{DL}) is calculated by:

$$n_{DL} = \left\lceil \frac{d_{PDU_ISO14229}}{d_{SDU_Iso10681-2}} \right\rceil$$

The final request's SDU size is calculated by:

$$d_{SDU(n)} = d_{PDU_ISO14229} \bmod d_{SDU_Iso10681-2} \quad (4.2-10)$$

Hence the total download time via FlexRay and ISO 10681-2 is calculated by:

$$t_{DL} = \left\lceil \frac{d_{PDU_ISO14229}}{d_{SDU_Iso15765-2}} \right\rceil \cdot \left(\left(1 + CR_{(FC_Req)} + \left\lceil \frac{n_{CF_PDU(x)}}{MNPC} \right\rceil \cdot CR_{(CF_Req)} \right) \cdot t_{Cycle} + \right. \\ \left. \left(t_{PhysProg} + (CR_{(STF_Res)} \cdot t_{Cycle}) + t_{PCU_Process} \right) \right) + \\ \left(\left(1 + CR_{(FC_Req)} + \left\lceil \frac{n_{CF_PDU(n)}}{MNPC} \right\rceil \cdot CR_{(CF_Req)} \right) \cdot t_{Cycle} + \right. \\ \left. \left(t_{PhysProg} + (CR_{(STF_Res)} \cdot t_{Cycle}) + t_{PCU_Process} \right) \right) \\ t_{DL} = \left\lceil \frac{d_{PDU_ISO14229}}{d_{SDU_Iso15765-2}} \right\rceil \cdot \left(\left(1 + CR_{(FC_Req)} + \left(\left\lceil \frac{n_{CF_PDU(x)}}{MNPC} \right\rceil \cdot CR_{(CF_Req)} \right) + CR_{(STF_Res)} \right) \cdot t_{Cycle} + \right. \\ \left. \left(t_{PhysProg} + t_{PCU_Process} \right) \right) + \\ \left(\left(1 + CR_{(FC_Req)} + \left(\left\lceil \frac{n_{CF_PDU(n)}}{MNPC} \right\rceil \cdot CR_{(CF_Req)} \right) + CR_{(STF_Res)} \right) \cdot t_{Cycle} + \right. \\ \left. \left(t_{PhysProg} \right) \right) \quad (4.2-11)$$

Note that after the final response no additional PCU processing time is required. To save FlexRay slot resources it is possible to use the node's transmission slots twice: a) to send communication layer's *FlowControl* PDUs in a segmented data transfer and b) to send UDS protocol's response PDUs. In that case the cycle repetition for the request *FlowControl PDUs* ($CR_{(FC_Req)}$) and the cycle repetition for the UDS response PDU ($CR_{(STF_Res)}$) are equal.

To accelerate the data transfer the programming time $t_{PhysicalProgramming}$ can be compensated by the double buffered data transfer approach of chapter 3. Delays that occur because of cycle repetition values unequal to one can not be compensated because they are either part of the request or the response is delayed and the PCU is not allowed by UDS protocol to start the next request without previously response reception.

4.2.4 Conclusion

The time triggered FlexRay bus system provides benefits for deterministic data communication e.g. for control and regulation systems which require signals within equidistant time slots. FlexRay has disadvantages for event driven communication. The main problem is the static, non-sharable allocation of bandwidth (slots) for each signal because of FlexRay's static address mechanism based on the relation of slot-ID and cycle number. With focus on software reprogramming bandwidth for each diagnostic request has to be allocated as well as bandwidth for the diagnostic response. This bandwidth is not usable for normal ECU communication. On the other hand the ECU normal communication is not usable for software reprogramming.

The communication layer protocol according to ISO 10681-2 solves the address problems partially but reducing the maximum PDU's payload length by additional four bytes (source address and target address). The address mechanism provides benefit for 1:n connections of an exclusive sender like a PCU or a gateway. The diagnostic requests can be sent in a broadcast slot. All connected ECUs will receive that PDU and evaluate the addresses. For diagnostic responses this mechanism provides no benefit because each ECU requires an exclusive slot for diagnostic response transmission.

The discussion above depicts that best performance is only possible if the basic communication schedule and the communication layer protocol are well concerted. Optimisations only on one layer (protocol) might not be sufficient. Nevertheless, the main influencing factor is the basic communication schedule design. Hence, to accelerate data transfer for software reprogramming purpose some aspects have to be taken into account during communication schedule design:

- 1.) Communication for software reprogramming purpose shall be allocated within the dynamic segment. If a communication link is currently not active only the minimum time delay as required by the protocol will occur before the FlexRay system switches to the next slot.
- 2.) The cycle repetition for all allocated slots shall be set to 1. This configuration guarantees that within each cycle communication is possible.
- 3.) For PCU's communication more than one slot shall be allocated. Because of the additional address information of ISO 10681-2 the PCU could manage the bandwidth depending on the required scenario, i.e. burst mode or parallel mode.
- 4.) If schedule switching is possible the re-organisation of the slot arrangement provides a lower risk than the timing modifications (static and dynamic segment length and cycle

time). The PDU's payload length shall be modified depending on the communication cycle length to prevent a gap at the end of the cycle because of a too long final PDU.

A generic evaluation about best configuration and a corresponding absolute maximum of data transfer ratio for an individual network is not possible. At least, the number of connected ECUs is influencing the number of available slots and varies from network to network.

A final approach to accelerate data transfer is the usage of the second FlexRay channel if hardware is available that supports the second channel. Some open topics can currently not be discussed finally, because the FlexRay communication controller interface is unknown. For communication on two channels the handling of the data to transmit is interesting. Shall a data transfer block be transmitted on only one channel or shall both channels be used to accelerate the data transfer? In the last case how the flow control shall be handled? On the other hand, if each channel transfers a complete block individually, is it possible to transfer 64 kByte⁴⁶ on each channel? Does the microcontroller provide sufficient RAM resources?

With the approaches as discussed above, it is possible to accelerate software reprogramming communication on a FlexRay bus system. Even if the second channel is currently not available the system provides good performance if all protocols are optimally configured.

4.3 Summary

In this chapter approaches to accelerate the data transfer via CAN and FlexRay bus system protocol stacks are discussed. These bus systems are currently the most popular bus systems for automotive networks.

CAN bus system

For the CAN bus system, as a representative for CSMA media access a data transfer rate, it is theoretically possible to generate 100% bus load for a single communication link between a PCU and an ECU, also within a network of several communication nodes. The maximum performance for data transfer via transport layer protocol is less than 50% of the gross bit rate (maximum at 1000 kBit/s). Delays on the transport protocol layer have a significant impact to the overall system performance.

⁴⁶ 64 kByte is the maximum payload that can be transmitted via ISO 10681-2 protocol.

With focus on software reprogramming time this bus system shall only be used for ECUs with a memory size less than 50 MByte and used for simple or complex control assignment (refer to chapter 1 – figure 1.2-2).

FlexRay

For FlexRay, as a representative for time triggered systems (TDMA), it is almost impossible to generate 100% bus load, particularly within a network with several other communication nodes. Due to the basic idea of time triggered protocols fixed time slots have to be allocated for each network node. Those allocated slots are only usable for the well defined communication link to exactly that node. Hence, the bandwidth is not usable for other communication links. For FlexRay in particular, the communication cycle division in a static and dynamic segment reduce the bandwidth, too.

Therefore, the communication schedule design has a significant impact to the overall communication performance. The schedule design is always a consideration between data transfer rate and bus resources (slots). Due to the high gross bit rate of 10 MBit/s in maximum an acceptable data transfer rate can be configured for ECUs with a memory size less than 50 MByte and used for simple or complex control assignment (refer to chapter 1 – figure 1.2-2). If the schedule can be reorganised during runtime also ECUs with more memory can be connected.

The effort to use time triggered protocols for spontaneous communication (as given for software reprogramming) is quite high, because the bandwidth for spontaneous communication has to be allocated statically within the schedule.

5 Data size reduction

Content

5.1 Partitioning	97
5.1.1 Analysis	97
5.1.2 Discussion	98
5.2 Fill byte skipping	99
5.2.1 Analysis	99
5.2.2 Discussion	100
5.3 Data compression	103
5.3.1 Analysis	103
5.3.2 LZ77 and LZSS Algorithm	104
5.3.3 Discussion	106
5.4 Differential file	107
5.4.1 Analysis	108
5.4.2 Discussion	111
5.5 Conclusion	112

One approach to accelerate the software reprogramming process is to reduce the data size which has to be transmitted from a programming control unit (PCU) to an electronic control unit (ECU). The total reprogramming process time will be reduced because of a shorter data transmission time. There are two basic approaches to reduce the data size:

- 1) Reduce file size
 - a) Partitioning - build partitions to divide the application software into several parts to reprogram only those parts that have to be adjusted
 - b) Skip fill bytes - skip unused bytes within an embedded software file

- 2) Skip redundancies
 - a) Compression – transmit data without redundancies
 - b) Differential file – transmit only the differences between the new and the previous file

The power of these approaches is very different as well as the effort to implement a stable process. Within this chapter different methods for data size reduction are evaluated and the possibility is discussed to use these methods for automotive systems.

5.1 Partitioning

5.1.1 Analysis

A simple, but powerful approach to reduce the file size is to divide the initial file into several partitions, e.g. operating system, sensors, actuators, drivers, characteristic curves, communication protocol stacks etc. In case the software has to be changed for any reason (bug fixing, parameter set optimisation, software enhancement etc.) not the complete application software, but only the corresponding partition has to be reprogrammed.

For example, if an ECU supports 2 partitions with an equal size the reduction might be 50%. An absolute value for size reduction can not be given because it depends on ECU's usage and the possibility to divide the source code into several partitions. Nevertheless, if not all partitions have to be reprogrammed, and a data reduction and therefore, a transfer time reduction will occur.

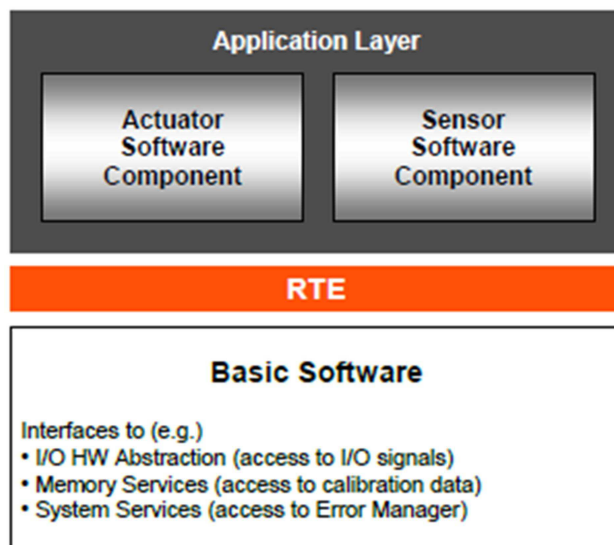


Figure 5.1-1: Application layer partitions [AUT11]

Figure 5.1-1 depicts an example with two application partitions and the basic software partition within an AUTOSAR⁴⁷ software architecture model.

5.1.2 Discussion

The division of software into several different partitions has to be done initially during software development process and is part of the software architecture decisions. The AUTOSAR software architecture model regards to the partitioning requirement and provides methods to divide software into several logical components. On the other hand this method increases system complexity and therefore, some additional aspects have to be taken into account.

Microcontroller aspects vs. number of partitions

The linking process has to take care that the software partition is linked to an own physical memory block within microcontrollers memory. This block shall be allocated exclusively for that partition to be able to erase and reprogram this block individually from all other partitions. As a consequence unused memory of this partition shall not be used for another software part. Hence, the number of possible and reasonable different partitions depends on a) the number of microcontroller's physical memory blocks⁴⁸ and b) the given size of the corresponding blocks. Of course, the definition of many partitions will reduce reprogramming time but requires a microcontroller derivate with a corresponding memory layout that physically supports those many partitions. As a result the cost reduction by reprogramming time requires a more expensive microcontroller to support the partitioning.

Software partition management and compatibility

It is a basic aim to be able to reprogram each partition independently of other partitions. For the pure physical reprogramming process this is normally given because each partition allocates a unique defined memory space with a unique address range. Nevertheless, dependencies between the different software partitions of an ECU are available (e.g. interfaces etc.) and therefore, their compatibility has to be managed. It has to be guaranteed that different partitions' software, which can be reprogrammed individually are compatible after reprogramming. A flashloader based on the HIS specification provides a special diagnostic service to check the compatibility (refer to section 2.2.3.2). Typically hardware-to-software compatibilities and software-to-software compatibilities have to be

⁴⁷ “[AUTOSAR] (AUTomotive Open System ARchitecture) is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers“.

⁴⁸ Within different microcontroller specifications several terms to describe contiguous memory fragments are used: e.g. memory section, memory bank, memory page, memory block etc.

checked. The complexity of the different compatibility dependencies is increasing the more partitions are defined.

The approach is useless if the address ranges of partitions are moved because of software expansion of another partition. In that case all reallocated partitions have to be reprogrammed to get compatibility. Hence, memory reserves have to be taken into account during system design to prevent the ECU for additional reprogramming activities because of address shifts.

5.2 Fill byte skipping

5.2.1 Analysis

Typically gaps within the binary code's allocated address range are filled with so-called fill bytes during the linking process. An approach to reduce data size and therefore, transfer time is to skip such fill bytes and transfer only pure compiled binary code from PCU to ECU. As a result gaps occur within the pure binary code. The corresponding address information of each allocated memory space, i.e. start address and length information have to be transmitted for each pure binary code part. With another view the pure binary code gap ranges have to be transmitted because the ECU needs the memory position information where the data transfer has to be continued.

Of course, it requires some time even to transmit this gap information as well as transmit the fill bytes. Hence, the break even point of both methods has to be calculated.

$$t_{\text{FillByteTransfer}} = t_{\text{GapTransfer}} \quad (5.2-1)$$

Based on the approach of formula 5.2-1 it is possible to calculate the number of fill bytes that can be transferred within the time slot where the gap address information is transferred (refer to the discussion below). Based on the calculation the distinction is possible whether fill byte transmission or gap information transmission provides the faster and more efficient solution.

The break even point depends on the individual communication link performance between the PCU and the ECU. The bus system's bandwidth and communication protocol stack's performance have to be taken into account. The value has to be calculated individually for each communication link especially within heterogeneous networks where different protocol conversions are necessary.

5.2.2 Discussion

To show the influencing aspects a data transfer via UDS⁴⁹ on ISO 15765-2 (Transport Protocol) and CAN is assumed. By the developed generic formula the system performance can be calculated to decide whether gap transfer provides benefits compared to fill byte transfer.

Data Transfer based on UDS, ISO15765-2 and CAN

Figure 5.2-1 depicts the two scenarios of fill byte transfer and gap transfer.

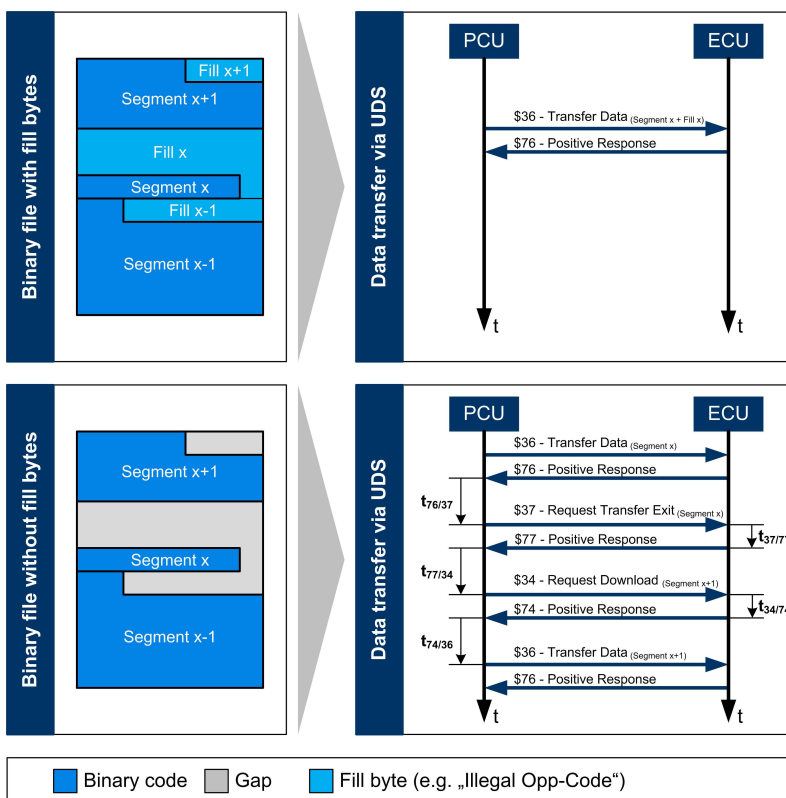


Figure 5.2-1: Data transfer with and without fill bytes

Fill byte transfer is processed by the UDS defined diagnostic service \$36 – *Transfer Data*. In contrast the transfer of gap address information requires the two additional UDS defined diagnostic services \$37 - *Request Transfer Exit* and \$34 - *Request Download* as well as their positive responses with the *Service Identifiers (SID)* \$77 and \$74. The diagnostic requests with the *SID* \$37, \$77 and \$74 are transferred as unsegmented *Single Frame – PDUs* (refer to ISO-15765-2 in chapter 3), each transmitted by a single CAN-PDU. The diagnostic service request \$34 – *Request Download* requires segmentation because the complete diagnostic service request with the additional request parameters *start address information* and *data length information* is in sum longer than 7 bytes.

⁴⁹ UDS – Unified Diagnostic Services [ISO 14229]. Refer to chapter 2.5.4.

Hence, based on ISO 15765-2 transport layer protocol three CAN-PDUs are necessary (*First Frame, Flow Control, Consecutive Frame*). In sum 6 CAN-PDUs are required to transfer the gap address information via the UDS protocol, the transport layer protocol ISO 15765-2 and the CAN protocol.

According to figure 5.2-1 and formula 5.2-1 the break even point can be calculated as:

$$\begin{aligned}
 t_{\text{FillByteTransfer}} &= t_{\text{GapTransfer}} \\
 t_{\text{Transfer36}} &= t_{76/37} + t_{\text{Transfer37}} + t_{37/77} + t_{\text{Transfer77}} + t_{77/34} + t_{\text{Transfer34}} + \\
 &\quad t_{34/74} + t_{\text{Transfer74}} + t_{74/36}
 \end{aligned} \tag{5.2-2}$$

With PCU's processing time $t_{\text{PCUProcessing}} = t_{76/37} = t_{77/34} = t_{74/36}$ and

with ECU's processing time $t_{\text{ECUProcessing}} = t_{37/77} = t_{34/74}$

$$\begin{aligned}
 t_{\text{Transfer36}} &= 3t_{\text{PCUProcessing}} + 2t_{\text{ECUProcessing}} + t_{\text{Transfer37}} + t_{\text{Transfer77}} + t_{\text{Transfer34}} + \\
 &\quad + t_{\text{Transfer74}} \\
 t_{\text{Transfer36}} &= 3t_{\text{PCUProcessing}} + 2t_{\text{ECUProcessing}} + 6t_{\text{CANPDURuntime}} \\
 t_{\text{Transfer36}} &= 3t_{\text{PCUProcessing}} + 2t_{\text{ECUProcessing}} + 6 \frac{\text{PDU}_{\text{Length}}}{\text{BitRate}}
 \end{aligned} \tag{5.2-3}$$

Within the time $t_{\text{Transfer36}}$ a well known number (x_{PDU}) of ISO15765-2 defined *Consecutive Frames* PDUs can be transferred. Each *Consecutive Frame* PDU transmits $n_{\text{BytePerPDU}}$ payload (e.g. 7 byte payload for ISO10761-2). The number of fill bytes that are possible to transfer can be calculated as depicted below:

$$\begin{aligned}
 \text{FillBytes} &= x_{\text{PDU}} \cdot n_{\text{BytePerPDU}} \\
 &= \frac{t_{\text{Transfer36}}}{t_{\text{PDU}}} \cdot n_{\text{BytePerPDU}} \\
 &= \frac{3t_{\text{PCUProcessing}} + 2t_{\text{ECUProcessing}} + 6 \frac{\text{PDU}_{\text{Length}}}{\text{BitRate}}}{\frac{\text{PDU}_{\text{Length}}}{\text{BitRate}}} \cdot n_{\text{BytePerPDU}}
 \end{aligned} \tag{5.2-4}$$

Table 5.2-1 depicts the number of fill bytes according to formula 5.2-4 that can be transferred depending on different CAN bus system bandwidths, the ideal PDU's runtime and different PCU and ECU processing times⁵⁰.

⁵⁰ The processing time of PCU and ECU depends on several influencing parameters. For an ECU it is important, whether the data reception is done in ECU's interrupt mode or if the ECU is polling to the receiver in a task mode. Therefore, only some exemplary values are given in table 5.2-1 to illustrate the wide range of that approach. Real processing time values are measured in the case study of chapter 10.

Generally, the number of possible transferred fill bytes (represents the gap size) is higher, the faster the bus system is. As a logical consequence the number of fill bytes is increasing too, the slower PCU's and ECU's processing time is.

Table 5.2-1: Break even calculation

CAN Bandwidth [Bit/s]	Runtime [ms]		Processing Time [ms]			Total Time (for gap address information transfer) [ms]	Number of possible transmitted fill bytes
	single PDU	all PDUs	on PCU	on ECU	total		
1.000.000	0.123	0.738	1	1	5	5.738	322
1.000.000	0.123	0.738	3	3	15	15.738	889
1.000.000	0.123	0.738	5	5	25	25.738	1463
1.000.000	0.123	0.738	10	10	50	50.738	2884
500.000	0.246	1.476	1	1	5	6.476	182
500.000	0.246	1.476	3	3	15	16.476	462
500.000	0.246	1.476	5	5	25	26.476	749
500.000	0.246	1.476	10	10	50	51.476	1463
250.000	0.492	2.952	1	1	5	7.952	112
250.000	0.492	2.952	3	3	15	17.952	252
250.000	0.492	2.952	5	5	25	27.952	392
250.000	0.492	2.952	10	10	50	52.952	749
125.000	0.984	5.904	1	1	5	10.904	77
125.000	0.984	5.904	3	3	15	20.904	147
125.000	0.984	5.904	5	5	25	30.904	217
125.000	0.984	5.904	10	10	50	55.904	392

The main influencing timing factors are changing depending on PCU's and ECU's processing time. In consequence for high bandwidth bus systems the processing time is a very important factor for the distinction whether a fill bytes transfer approach or a gap information transfer approach shall be implemented.

Within the automotive area processing times of $1 \text{ ms} \leq t_{\text{Processing}} \leq 10 \text{ ms}$ are possible. As a result a gap shall be more than 1463 byte on a CAN bus with 500 kBit/s gross data transfer rate. If the gap is smaller than that value, it requires more time to transmit the gap information than the fill bytes.

As an additional non-communication but safety aspect the usage of "illegal operation code" for fill byte values within binary code is best practice. If the microcontroller read from that addresses, e.g. if a memory calculation operation failed, the illegal operation code forces the microcontroller to a safe state (e.g. reset). In contrast, gaps in data areas (e.g. characteristic curves or diagrams etc.) will often be filled with zero.

System design aspects

Knowledge about the complete communication system's processing delays are a precondition to decide, whether fill bytes shall be skipped or not. For fast systems with small delays this utilisation of that approach is not recommended.

5.3 Data compression

Data compression is a possible approach for data transfer acceleration. The aim is to reduce the total number of data that shall be transmitted. Data compression is a branch of information theory and was published the first time by C. E. Shannon in 1948 [Sha48]. K. Savood provides an overview about data compression and explains most currently known compression methods and algorithms [Say05].

5.3.1 Analysis

Lossy and lossless compression

Data compression methods can be divided into two basic types: 1) lossy compression and 2) lossless compression. With focus on software reprogramming only lossless compression has to be taken into account because the transferred data file shall be restored completely without loss of information after the compression, transmission and decompression process. Changing even a single bit within the initial binary code cannot be tolerated. A brief introduction to “mathematical preliminaries for lossless compression” is given by K. Savood in [Say05-1]. Lossless compression is divided then again into two different fundamental approaches: 1) *statistical data compression* or 2) *substitutional data compression*.

Statistical data compression methods use the symbol probability of the different characters within a file to reduce the data length. Popular compression methods based on the statistical approach are *Arithmetic coding* (introduction is given by A. Said in [Sai04]), *Huffman coding* [huffma] and *Shannon-Fano* [shanno] *coding*.

Substitutional data compression methods replacing parts of the uncompressed symbol string by references to a dictionary. Popular compression methods based on substitutional compression are *LZ77*⁵¹ algorithm [Ziv77] and *LZSS*⁵² algorithm [Sto82].

For selection of a compression method, running on an embedded system's microcontroller some additional basic requirements have to be taken into account:

Resource restrictions

Embedded systems have resource restrictions. The size of available RAM to allocate large dictionaries is limited. As a result it might be possible that algorithms based on large dictionaries could not provide their full performance because of dictionary size limitations.

⁵¹ Jacob Ziv, Abraham Lempel, known as Lempel-Ziv algorithm, published 1977. Refer to [Ziv77]

⁵² James Storer, Thomas Szymanski, published 1982. Refer to [Sto82]

Clock frequency

Microcontroller's clock frequency is significantly lower than the typical known clock frequencies within the PC environment. Hence, it is desirable to have fast de-compression whereas compression on PCU's or offboard site could be relatively slower.

Statistically based approaches for embedded source code

Compared to typical human languages for microcontroller's source code it is not easy to identify the most used character. Source code in a binary file represents assembler commands and depends on microcontroller's ALU (Arithmetic Logical Unit) implementation and design.

Patents

A non-technical but commercial aspect is the question whether the algorithm is patent-protected. If a licence fee for the usage is demanded, it has to be distinguished whether the strength of that algorithm justifies the financial disadvantage compared to patent free approaches. Compression algorithms that consider the previously discussed aspects are the *LZ77* and *LZSS* algorithm.

5.3.2 LZ77 and LZSS Algorithm

Jacob Ziv and Abraham Lempel published the LZ77 compression method in [Ziv77]. This approach is a simple but efficient method where not longer the probability entropy of characters was coded. This method uses the repetition of characters within a data string. It is based on a sliding window method where a buffer is split into two parts: a) a history buffer and b) a look-ahead buffer. The look-ahead buffer contains the next character that shall be coded. The history buffer contains characters that have been coded previously. The algorithm compares the look-ahead buffer's characters with characters of the history buffer and searches for the position of the longest matching string pattern. The position within the history buffer, the length of the matching string and the next character after matching string within the look-ahead-buffer is coded. After that step the window of both buffers is shifted by the matching length + 1 towards the look-ahead buffer. A detailed description of that algorithm is given in [dataco].

The LZSS algorithm is based on the LZ77 algorithm. J. Storer and T. Szymanski published the method in [Sto82] in 1982. This approach replaces the window buffer mechanism by a ring buffer. Also the coding for matching pattern was changed. Figure 5.3-1 depicts an overview.

A single bit indicates whether the original character from the look-ahead buffer is coded or if a matching string was found in the history buffer. In the last case the position of the matching string within the history buffer is coded. Because of the ring buffer mechanism

the position of the data stream within the history buffer doesn't change. Access by an index is possible and a small dictionary tree could be build up very fast. As a result the coding speed increases. A detailed description of that algorithm is given in [dataco].

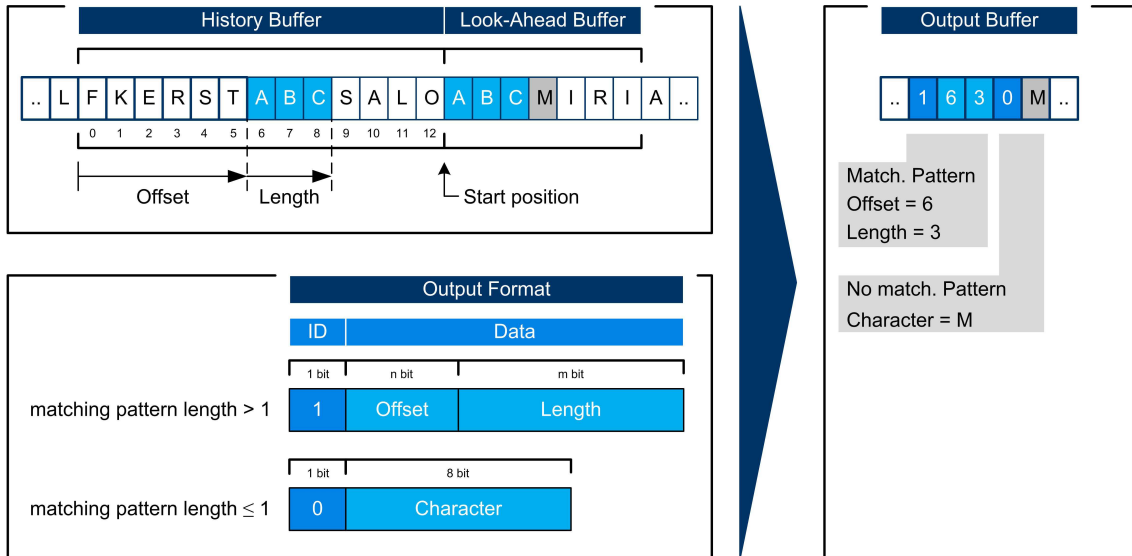
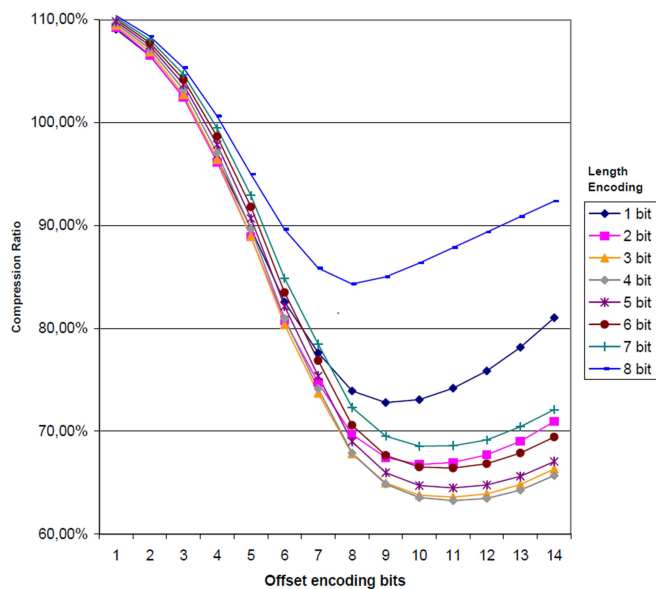


Figure 5.3-1: LZSS algorithm

F. Hees has done research on the implementation of compression algorithms for Vector's⁵³ Flashloader in 2004 [Hee04]. He analysed the parameter for offset and length coding within an output stream. Figure 5.3-2 depicts the compression ratio depending on the length encoding and the offset encoding.



Source: [Hees2004]

Figure 5.3-2: LZSS Compression Results [Hee04]

⁵³ VECTOR Informatik GmbH, Germany [Vector]

A good compression ratio is possible if:

- a) Length encoding is between 3 and 5 bits (best: 4bit).
- b) Offset encoding is between 10 and 12 bits (best: 11 bit).

For other parameter combinations compression ratio will decrease. F. Hees discussed this effect: Because “a larger history buffer size increases the possibility to find a matching phrase. But the number of bits used for the offset encoding increases as well. At a certain point, this offset encoding requires more bits than the larger history buffer saves” [Hee04]. An equal effect will occur for the length encoding. “If the number of bits for length encoding is increased, it will be possible to encode longer phrases. At a certain point, this length encoding requires more bits than the larger look-ahead buffer saves” [Hee04].

The parameter pair of 4 bit length encoding and 12 bit offset encoding provides a compression ratio of approx. 64%. As a consequence the number of bits to be transferred from PCU to ECU is reduced too and 35% of transfer time is saved.

5.3.3 Discussion

Data compression seems to be a good approach to accelerate data transfer and in the end software reprogramming, too. The data ratio diagram of figure 5.3-2 depicts that the LZSS compression algorithm, which is possible to be implemented on an embedded system, provides nearly 35% data reduction. An important estimation parameter is the runtime of that algorithm. If the time reduction based on data compression is compensated by the de-compression runtime only additional resources have been wasted. Hence, it is necessary that

$$t_{\text{CompDataTransfer}} + t_{\text{Decompression}} < t_{\text{UncompDataTransfer}} \quad (5.3-1)$$

De-compression time depends on microcontroller’s clock frequency, resource availability, de-compression algorithm etc. If formula 5.3-1 is not fulfilled compression provides no benefit.

An important part for LZSS optimisation is the problem of aligned buffer access for microcontrollers. The LZSS algorithm example above provides good results if the offset encoding parameter has 12 bits and the length encoding parameter has 4 bits. Together with the single bit of the Identifier (refer to figure 5.3-1) a matching pattern requires 13 bits for encoding. A single character requires 9 bit. For both scenarios a microcontroller has to shift the compressed data string to get data into a byte aligned format. F. Hees provides an optimisation for that problem in [Hee04]: As depict in figure 5.3-3 an optimisation is possible if the identifier bits are grouped into 16 bit tuples ahead of a group of 16 compression patterns within the data stream. The advantage is a byte aligned access to the

data stream. The bit by bit shifting of the compressed data stream is not longer necessary and acceleration of de-compression is possible.

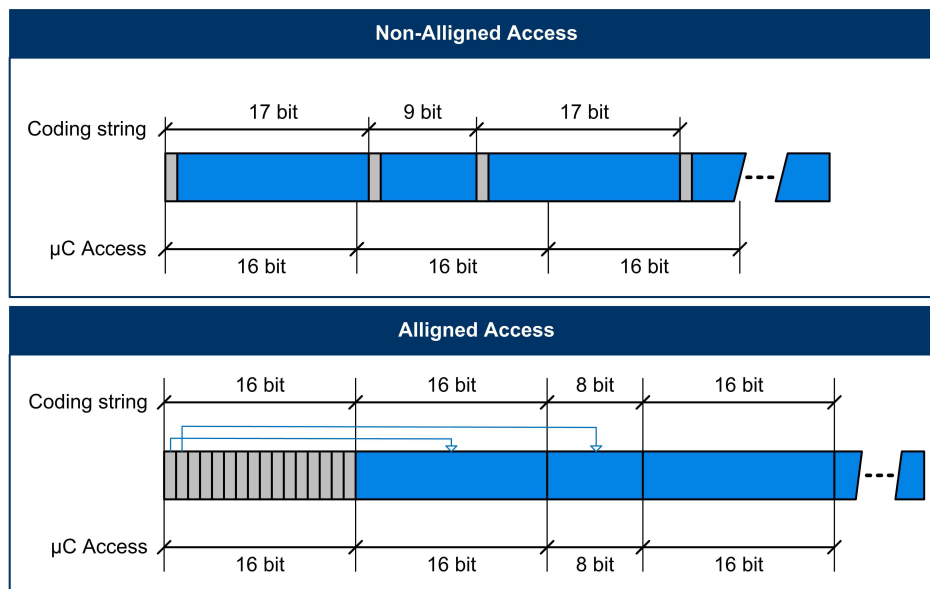


Figure 5.3-3: LZSS Optimisation

Of course, optimisation of the discussed compression algorithm to get better compression ratio might be possible. But this requires some additional research with focus on algorithm optimisation. The scope of these research activities was on data transfer acceleration based on data size reduction for embedded systems. Nevertheless, as a conclusion each compression algorithm and its optimisations have to be proofed, if it fulfills formula 5.3-1 and if the resource requirements are within tolerable boundaries.

5.4 Differential file

Another method to compress data for their transmission is to transmit only the differences of two files. This approach is used within PC operating systems. Microsoft, for example, uses the *Binary Delta Compression (BDC)* technology to “reduce the download size of software update packages for Windows operating systems” [Pot05]. The aim is to create “smaller software update packages that require less time and network bandwidth to install” [Pot05]. The BDC compression ratio could be significantly higher than all other file compression approaches⁵⁴.

⁵⁴ BDC compression ratio: 10:1 up to 1.000:1. It depends on the real differences of two files and their size [Pot05].

Embedded system's software bug fixing is an issue for differential file update

One reason for re-programming embedded software is bug fixing. In most cases embedded software does not change completely when fixing a bug (e.g. changing a value of a constant or some parameters within a characteristic curve etc.). As a percentage of the total volume of an application the source code modifications required and the resulting OP-code changes, required for bug fixing is often very small. Typical errors in the source code like wrong exit conditions in loops or wrong statements for a comparison are only a few characters. Changes in characteristic curves implemented as arrays covers only a few bytes, too.

Thus an assumption that 80% of bug fixings result in less than 1 kByte OP-code changes and 20% in more than 1 kByte is a realistic figure. As a result of this assumption only a few bytes within a memory sector or partition needs to be changed. Figure 5.4-1 depicts the small OP-code difference within a software partition.

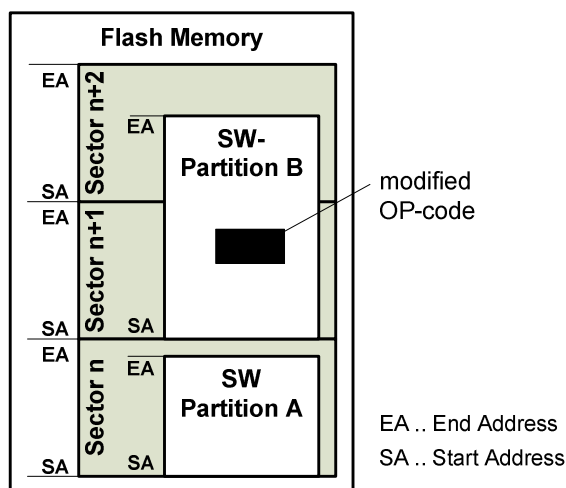


Figure 5.4-1: Modified Op-Code in case of bug fixing

5.4.1 Analysis

Method

The method of differential file calculation seems to be very simple. The difference between the old file and the new file is calculated. This differential file is transmitted to the ECU. In the ECU's memory the old file is stored. With the received differential file and the available old file the new file could be re-calculated. Figure 5.4-2 depicts the differential file process. Of course, mapping that approach to the embedded world suggest a strong reduction of data to be transmitted. But here some additional environmental requirements have to be taken into account to benchmark that method.

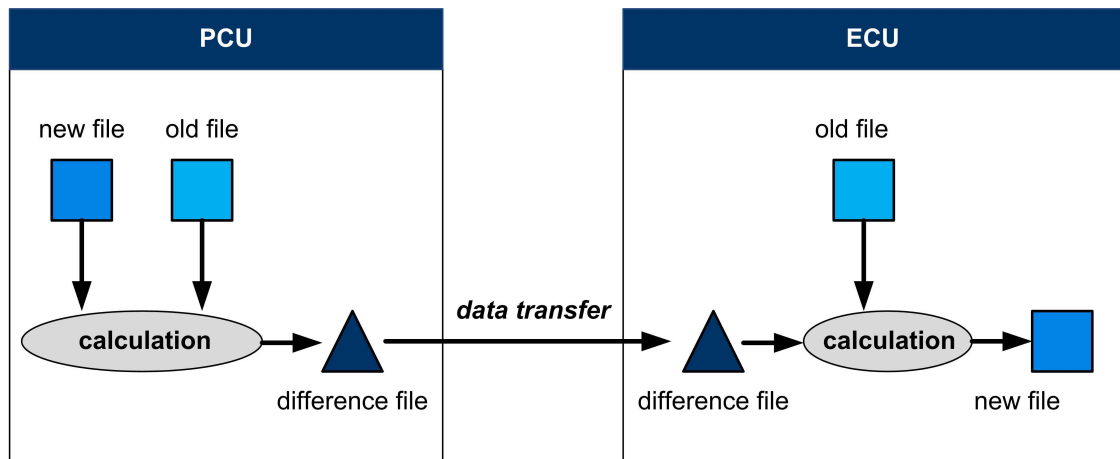


Figure 5.4-2: Differential file update

Embedded system's flash memory impact

The Flash memory technology characteristics are described in section 2.2.2. Due to those facts, some impacts occur and some restrictions are given to the differential file update method:

- 1) Flash memory cells can not just be overwritten. The memory cell has to be erased previously before the new information can be programmed.
- 2) Flash memory can not be erased byte-wise. Typical Flash memory devices provide micro pages, blocks or sectors. These are consecutive ranges of several memory cells. The size of such a memory block depends on the flash memory technology and the overall flash memory size. Depending on the fact of 1) usually such a complete block must be erased and reprogrammed even if just one bit within that memory area has to be changed.
- 3) In contrast to the PC world where software is stored file oriented and virtual addressed, the embedded system's microcontroller work physical address oriented. Typically memory access to operation code elements (e.g. variables or arrays in RAM, OP-code or constants in Flash, jump's target addresses) is in relative address mode (basic start address and offsets). Within the PC world a single file can be changed and the memory managing system is able to allocate it on a free memory space if the new file is expanded. A microcontroller provides neither a memory managing system nor that much memory to squander memory space. This is why during the embedded software generation process all source code elements (e.g. in C-language: functions, arrays etc.) are linked consecutively without any larger gaps within the address space. Consequently, if a routine expands all other compiled elements will change the allocation address. In that

case the differences between a previous file and a new compiled and linked file will be quite high while the source code changes are only a few lines of code.

Issue 1) and 2) are microcontroller's internal reprogramming process optimisations. Issue 3) requires a change within the typical software generation process to generate the final output binary file (INTEL-Hex-Record⁵⁵ or MOTOROLA S-Record⁵⁶).

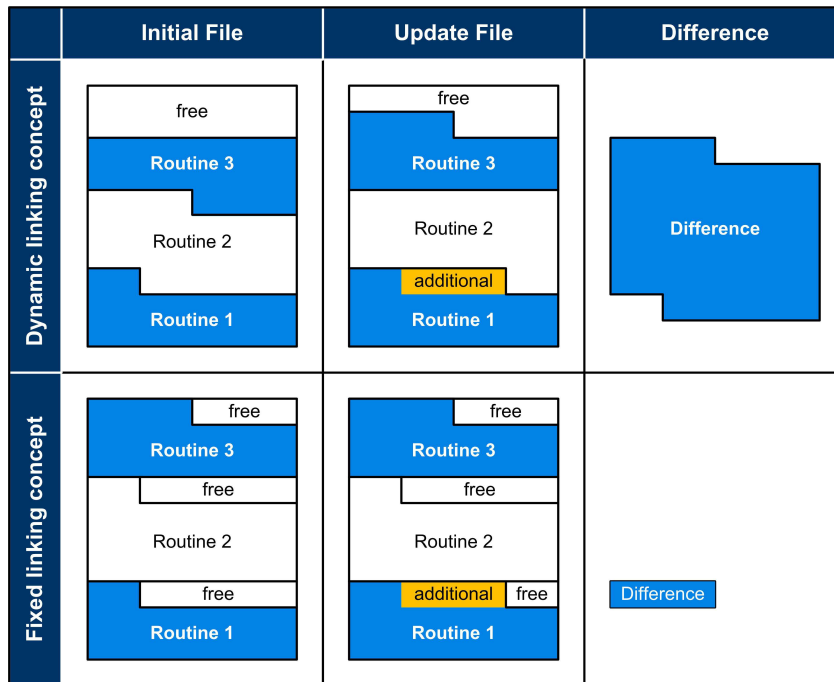


Figure 5.4-3: Differential file update

To be able to reduce the differences of the embedded software files it is necessary to allocate all the software parts always on the same position (address). In that case also the correct functionality because of microcontroller's memory access by relative addresses is guaranteed. This requires a fixed linking concept as depicted in figure 5.4-3. If this is not guaranteed the smallest possible difference of both files can not be calculated. However, a link process with fix addresses can be implemented by different approaches.

- 1) A fix position for at least each source code module (e.g. c-file, object-file etc.) must be configured within the linker command file.
- 2) Best results provide the fix allocation on source code function level. Here each function or array etc. is allocated on a fix position.

⁵⁵ INTEL Hex Record: Hexadecimal object file format initially for the Intel-architecture based micro-controllers [Int88].

⁵⁶ MOTOROLA S-Record: Hexadecimal object file format initially for the MOTOROLA 6800 architecture based microcontrollers [Mot92].

If these environmental requirements have been taken into account, the abstract sequence to reprogram flash memory by differential file updates can be processed as listed below:

- a) The flashloader identifies the memory section that includes the requested byte(s) that shall be changed.
- b) The flashloader copies the complete memory section temporarily into RAM. This requires that the microcontroller provides enough additional RAM for this step.
- c) The flashloader overwrites the old byte value by the new byte value transmitted via the differential file.
- d) The flashloader erases the original memory section within the Flash memory.
- e) The flashloader programs the new values from RAM into the Flash memory sector.

Step b) and c) could be done within the copy process to optimise runtime. If not enough RAM is available, the microcontroller shall provide an additional, usually unused Flash memory sector for the copy process. This solution is less powerful than the RAM access method, but in some cases this might be the only possibility.

The differential file structure can have equal structure as the currently given INTEL-Hex-Record or MOTOROLA S-Record file. Within both file types address information and the corresponding data of the identified differences can be stored.

5.4.2 Discussion

Differential file update is a very strong approach to accelerate data transfer as well as the total reprogramming process. The performance depends on the differences of the initial file and the new file and on microcontroller's memory technology.

Microcontroller aspects

The internal mechanisms are complex to reprogram a differential file to the Flash memory by a flashloader. It requires large RAM resources to copy the initial, currently active code from Flash memory's internal section. Because of the Flash memory's restrictions (previously erase process before programming) this step is necessary to overwrite the corresponding positions with the new data values in RAM. But if sufficient RAM is available and the memory micro pages are small enough the total reprogramming time could be reduced significantly.

Memory space vs. cost aspects

A disadvantage of this new method that has to be discussed is the necessity of gaps (free memory spaces) in between the different source code elements for further use. These gaps have to be included if code elements will enlarge in the future. The granulation optimum has to be set individually and could be on programming language object

level (c-functions, arrays etc.), on source code module level (c-file or object file) or functionality level (communication stack, driver software, sensor software etc.). On the other hand this expands the total file size and initially requires more memory space on microcontroller's memory device. As discussed in chapter 1 automotive microcontroller provide strong memory resource restrictions. Hence, the commercial relation of higher costs for a larger memory vs. reprogramming time and cost reduction has to be taken into account. Because of that cost pressure the break even point of this solution has to be calculated individually. Also the final risk is always given, that the gap is too small for the necessary changes and therefore, the complete memory section changes.

Stringent version and compatibility control management

Especially within the automotive industry a stringent version and compatibility control management are required because software on a car is only reprogrammed if it is in a repair shop. Because of the large service intervals of modern vehicles it might be possible that several software versions are in between the current vehicle software and the current OEM software.

MRAM technology

As written above, today's established Flash memory technology provides the elementary disadvantage that a byte-wise erase and write access is not possible.

With focus on differential file updates the erasing of complete physical memory sectors is required and a complete writing of those sectors is the consequence. However a real improvement will be possible if the established flash memory technology in currently available microcontrollers is replaced by the new MRAM technology (*Magnetoresistive Random Access Memory*). In contrast to other memory technologies MRAM semiconductors store the information not by electrical, but by magnetic load elements.

In chapter 9 a short introduction as well as a discussion to benefits of this new memory technology is given. Until today there are no experiences with MRAM based microcontrollers because those systems are not available. However, the theoretical discussion of MRAM depicts the high potential of that memory technology (refer to appendix A).

5.5 Conclusion

There are several approaches for data size reduction with the aim to reduce data transfer time and as a result reduce total reprogramming time. However, the methods provide significant differences with focus on automotive usage.

Theoretical case study to compare the approaches

The power of the different approaches is compared by a theoretical case study based on the following assumptions and on typical data for an ECU that processes complex control assignments, e.g. diver assistance systems:

Assumption:

Total file size:	32 MByte
Modified OP code size:	1 kByte
Memory partitions:	2 (results in 2 x 16 MByte)
Compression ratio:	75%
CAN Payload:	8 Byte / PDU (no transport protocol etc.)
Approximate frame length:	123 bit (11 bit CAN Identifier)
CAN bit rate:	125 kbit/s, 500 kbit/s, 1 Mbit/s

The data transfer time t_{transfer} is calculated by formula 4.5-1. Additional protocols have not been taken into account at this quantitative method comparison. The aim is to illustrate the power of the different approaches.

$$t_{\text{transfer}} = \frac{\text{DataVolume}}{\text{Payload}} \cdot \text{FrameLength} \cdot \frac{1}{\text{bitrate}} \quad (5.5-1)$$

Table 5.5-1 depicts an overview of the data transfer time on different CAN bus systems.

Table 5.5-1: Data transfer time via CAN

Description	File Size (Data to transmit)	Data Transfer Time on CAN			unit
		125	500	1000	kbit/s
Original file (complete)	32 MByte	4127.2	1031.8	515.9	s
Compression (-25%)	24 MByte	3095.4	773.8	386.9	s
2 Partitions	16 MByte	2063.6	515.9	257.9	s
Partitioning and Compression	12 MByte	1547.7	386.9	193.5	s
Differential File	1 kByte	0.1	0.031	0.016	s

Table 5.5-1 depicts that if only the differences of two files will be transferred the data transfer time is significantly reduced compared to conventional data size reduction methods e.g. partitioning and compression. The data transfer time is a function of the data size to be transmitted. The experiment depicts the power of the differential file transfer approach. Of course, the model is simplified and additional overhead of upper layer protocols, additional processing delays, the erase time and the programming time will decrease the performance. Nevertheless, these performance reduction parameters are similar to all approaches.

Effort vs. data size reduction

Especially the relation between effort and the typically possible data size reduction value depicts those differences. Figure 5.5-1 depicts this relation in a diagram.

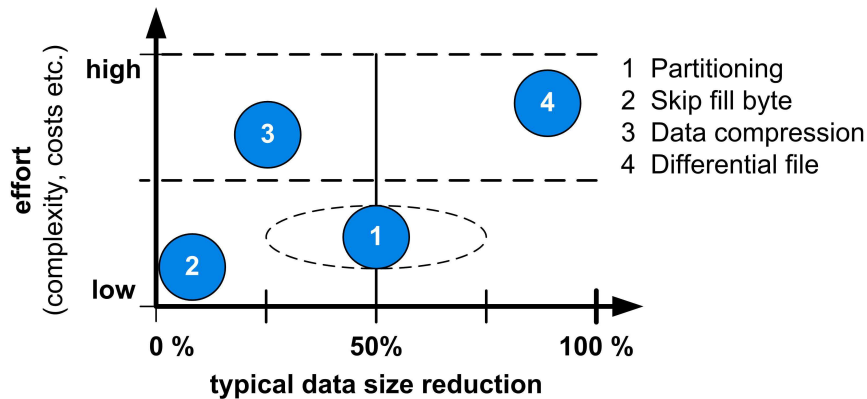


Figure 5.5-1: method's complexity vs. typical data size reduction

Partitioning is from technical point of view a simple method and provides good data size reduction results in a very special case. Transfer time reduction is given if at least one of these partitions is not reprogrammed. If all partitions shall be reprogrammed no benefit is given by this method. The ECU's internal implementation of partitioning is simple. On the other hand the effort for compatibility management of the different partitions is increasing because the software compatibility of the different partitions has to be verified.

Skipping fill bytes is the easiest way to reduce data because this can be configured for embedded software's linking process. But that method is less powerful. In the worst case the fill byte separation will result in a longer reprogramming time because the transfer of the gap addresses during the reprogramming process requires more time than the continuous data transfer inclusive the fill bytes.

The power of data compression algorithms is limited by the given microcontroller resources as well as the fact of only lossless methods can be used. Especially the RAM limitation avoids the usage of dictionary based algorithms. Compared to the other methods the effort for implementation and the off-board processes is high.

Differential file update could provide best results for data size reduction. However, this is only possible if the software development process is modified for that approach. The necessity of fixed module start addresses to avoid a general address offset for the whole code requires more memory, which results in higher costs. On the other hand if several megabytes of binary code could be reduced to a few bytes the time benefit is very high. The break even point of cost benefit relation for that method has to be evaluated individually for an ECU. Nevertheless, with focus on the continuously increasing automotive software sizes this method might be the best solution to solve the timing problems in future.

Only by a size reduction of 90% and more, acceptable reprogramming times will be possible. Additionally, this method can be combined with all other discussed methods.

Outlook

The next evolutionary step in embedded memory technologies will be *Magnetoresistive Random Access Memory (MRAM)*. The advantages of MRAM based systems are quite evident. The main advantages of MRAM vs. Flash memory technology with a focus on reprogramming activities are the byte-wise access and the possibility to overwrite data without an initial memory erase phase. Then reprogramming by the differential file approach will become the best solution, because the implementation of ECU's internal data handling loses complexity. Due to the reduced amount of data to transfer, the data transfer time and the physical programming time could be reduced significantly (refer to appendix A).

6 Microcontroller Hardware Optimisation

Content

6.1 Memory status information	116
6.1.1 Analysis	117
6.1.2 Discussion	119
6.2 Doubling interrupt vector tables.....	121
6.2.1 Analysis	121
6.2.2 Discussion	122
6.3 Conclusion.....	123

Within this chapter approaches are discussed to accelerate the physical memory programming process on an electronic control unit (ECU) by implementing functionality in microcontroller's hardware rather than in software as it is currently done.

The main advantage of a hardware implementation in contrast to a software implementation is the concurrent execution. Software implementations are typically organised in tasks and interrupts. The operating system schedules the tasks within a task cycle time. Task activities, e.g. calculations, communication processing etc., can only be executed if the task is active. If the task is inactive, e.g. in case an interrupt task is executed, a delay for the activity occurs. By an implementation in hardware neither task time nor interrupt runtime are required. The action can be executed concurrently to other hardware or software operations.

6.1 Memory status information

An implementation of a memory status information register within microcontroller's hardware provides trustable information about the current memory state (erased or programmed). Based on this information it is possible to accelerate the reprogramming

process. In case, an ECU is reprogrammed within the assembly line, the erase memory step can be skipped, if the memory status signals an erased memory.

6.1.1 Analysis

ECU's functionality is the summary of hardware functionality (e.g. periphery elements like drivers, actuators, sensors etc.) and the corresponding software functionality to handle the hardware. Differentiations in functionality will be solved by different ECU software because different hardware variants (e.g. non-placement of components etc.) are too expensive. Hardware variations split the number of equal parts and therefore increase the hardware costs per part. A good illustration is an engine control module (ECM). Typically an engine will be integrated into several vehicle model lines. The ECM software implements the individual adaptation between engine, gear box and the corresponding vehicle but the hardware is always equal.

Finally, an ECU is always a combination of hardware and software parts and these combinations result in individual ECUs (variants). Nevertheless, the point in time when the combination is executed (ECU variant setting) has a deep impact on handling as well as on the final costs and process complexity.

Reduction of hardware / software variants

The cost pressure within the automotive industry requires optimisations within the ECU variation building process. Figure 6.1-1 depicts the different possibilities.

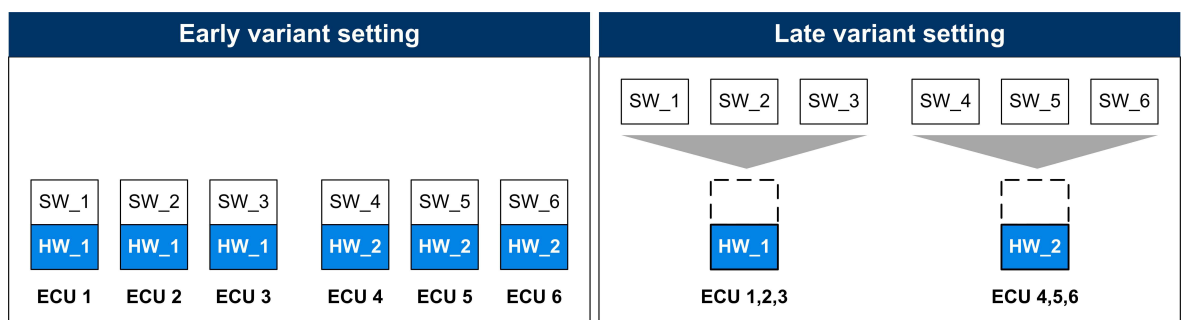


Figure 6.1-1: Early variant building vs. late variant building

If the ECU variant is built within ECU manufacturer's assembly line (early variant setting), the different ECU types have to be handled during the complete logistic process (e.g. ordering, delivery etc.) for manufacturing as well as for the after sales market spare parts. Therefore, a trend is visible in an increasing manner: ECUs won't be delivered fully programmed to the OEM's production line. Especially for those ECUs with many software variants, the tendency is to deliver them without application software (only with flashloader) to the OEM and program them within the assembly line during vehicle's

manufacturing process (late variant setting). The advantage of that approach is to reduce hardware / software combination variants. In consequence this results in a lean logistical process starting from the ordering process up to the allocated area for material boxes on the assembling line because only one (hardware) part has to be handled instead different ECUs for all combinations. The probability to produce an error vehicle because of a wrong ECU selection during manufacturing process is reduced, too.

An equal effect is visible for the after sales market spare parts. For late building variants only the hardware parts must be stored in the central logistic centres. In contrast to the early set variants of hardware and software combination this method requires less different stock grounds.

Skipping the “erase memory” process

Unfortunately, the currently used microcontroller’s flash memory technology requires the erasing of a flash memory cell before the cell can be programmed (refer to section 2.4). Especially for microcontrollers with large memory this behaviour results in a long erase time. Table 6.1-1 depicts the erase time for Infineon’s TC1197 microcontroller [TC1197].

Table 6.1-1: Infineon TC1197 Flash Parameter [TC1197]

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Program Flash Retention Time, Physical Sector ¹⁾²⁾	t_{RET} CC	20	–	–	years	Max. 1000 erase/program cycles
Program Flash Retention Time Logical Sector ¹⁾²⁾	t_{RETL} CC	20	–	–	years	Max. 100 erase/program cycles
Data Flash Endurance (64 KB)	N_E CC	30 000	–	–	cycles	Max. data retention time 5 years
Data Flash Endurance, EEPROM Emulation (4 × 16 KB)	N_{EB} CC	120000	–	–	cycles	Max. data retention time 5 years
Programming Time per Page ³⁾	t_{PR} CC	–	–	5	ms	–
Program Flash Erase Time per 256-KB Sector	t_{ERP} CC	–	–	5	s	$f_{CPU} = 180$ MHz
Data Flash Erase Time for 2 × 32-KB Sectors	t_{ERD} CC	–	–	2.5	s	$f_{CPU} = 180$ MHz
Wake-up time	t_{WU} CC	–	–	$4000/f_{CPU} + 180$	μs	–

1) Storage and inactive time included.

2) At average weighted junction temperature $T_j = 100^\circ\text{C}$, or the retention time at average weighted temperature of $T_j = 110^\circ\text{C}$ is minimum 10 years, or the retention time at average weighted temperature of $T_j = 150^\circ\text{C}$ is minimum 0.7 years.

3) In case the Program Verify feature detects weak bits, these bits will be programmed once more. The reprogramming takes additional 5 ms.

The Infineon TC1197 microcontroller provides currently 2 MB on chip flash memory bank [TC1197-1]. It results in a total erase time of 40s⁵⁷. If the late variant building process is used, the ECU is delivered without the specific application software only with the flashloader software to process reprogramming. If trustable information about the current memory state is available the erase process can be skipped, if the ECU's memory is currently erased. The initial erase process can be done either by the ECU supplier or by the microcontroller manufacturer.

Optimisation in case of programming process restart

The implementation of a memory state monitoring system enables additional possibilities to accelerate the reprogramming process: In case a currently executed reprogramming process is interrupted, e.g. by a communication interruption, it is not necessary to erase the complete memory again. If the flashloader is able to inform the PCU about the last successfully programmed sector, the PCU could restart programming process at exactly that position. Depending on the instant of interruption time, this approach will reduce the process time of the second programming sequence.

6.1.2 Discussion

To store the memory status of a physical memory sector⁵⁸ a single bit is sufficient (sector erased / not erased). It should be stored within non volatile memory.

Software approach

A first approach is to monitor the memory status by software and store that information within a non-volatile memory (NVM), e.g. an EEPROM. If the memory sector is completely erased, the corresponding flag shall be modified within the NVM. If the first memory cell is programmed within that memory sector, the flag shall be modified again. However, this simple implementation provides some disadvantages:

- (1) If the microcontroller has been changed the memory status information might be wrong. At least it is fortuity, if the values will match.
- (2) If the EEPROM's bit will toggle because of ageing or environmental influences the status will not be correct.

In both cases a reprogramming of a not erased memory is possible. In good case some memory cells will not have the required state (logical '1' or '0'). This will be detected by the memory check, e.g. by calculating a CRC sum. In worst case all cells provide the required

⁵⁷ 2048 kByte total size / 256 kByte sector size = 8 sectors → 8 x 5 sec = 40 sec.

⁵⁸ Current available Flash memory supports only erasing of complete sectors.

value but the programming quality is not good enough to fulfil the data retention time⁵⁹ (“i.e. the time after which stored data can still be retrieved” [TC1197]) as specified in the data sheet. In that case the memory cell will lose the programmed value over the years.

- (3) If one of the components was changed a software implementation could check the memory cells to synchronise the corresponding status. Unfortunately it is not possible to detect whether \$FF⁶⁰ was programmed or the cell has \$FF as its initial erase state. A sector with \$FF cell values could not unambiguously be detected as erased.

Hardware approach

Another approach is to handle the memory status information in hardware⁶¹. The microcontroller’s memory control unit implements a register with read-only access. For each memory sector a corresponding flag exists that is updated each time the sector is either completely erased or the first memory cell is programmed. Figure 6.1-2 depicts an abstract overview of a memory status information register implementation in hardware. The main advantage is that the information is more trustable than a simple software solution.

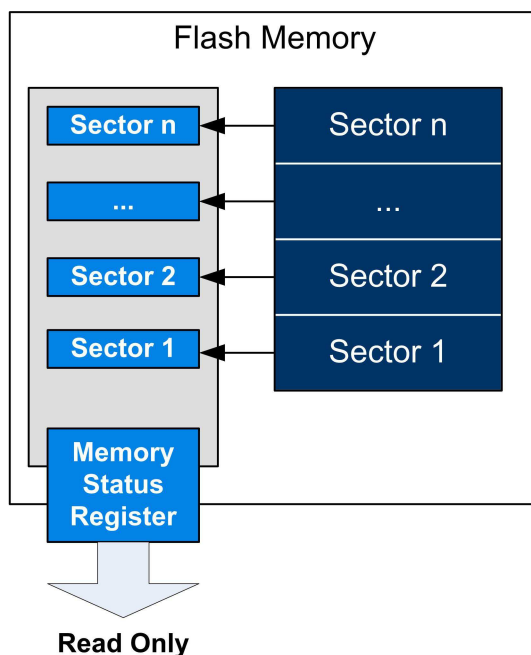


Figure 6.1-2: Memory Status Information Register

⁵⁹ Data retention for currently used flash memory cells is up to several years. For Infineon’s microcontroller TC1197: min. 20 years (refer to table 6.1-1).

⁶⁰ Hexadecimal nomenclature: \$FF represents ‘1111 1111’ in binary nomenclature

⁶¹ Submitted for patent: IP-Number P813194/DE/1 (06.10.2008) – Document Number 1455555 by Ralf Schmidgall / Daimler AG

6.2 Doubling interrupt vector tables

“In computing, an interrupt is an asynchronous signal indicating the need for attention (...)” [wikipede]. Hardware interrupts are typically triggered by events within the hardware like the reception of a PDU at the communication interface or an expired hardware timer. If an interrupt occurs the microcontroller suspends the current software execution and starts the interrupt service routine (ISR) execution. If the ISR has been finalised, the microcontroller continues the normal software execution. Interrupts are a powerful approach to react very fast to hardware or software events. With focus on software reprogramming typical events are the reception of data on the communication interface or internal timers that have expired. Especially for concurrently executed work (e.g. data decompression etc.) interrupt managed software execution provides execution speed benefits. Nevertheless, today’s typical microcontroller hardware is not able to select different interrupt services depending on a currently active software mode like application mode or flashloader mode. This selection has to be done by software and therefore requires code size and runtime. A selection of different interrupt service routines based on the currently active software mode directly by the microcontroller’s hardware will provide some advantages.

6.2.1 Analysis

Today a microcontroller supports only a single interrupt service routine vector table. This table stores the start address of the corresponding interrupt service routine that shall be executed if the interrupt of the corresponding interrupt source occurs. If the ECU software is divided into different functional parts which are not concurrently active (e.g. application mode or flashloader mode), multiple ISR are necessary.

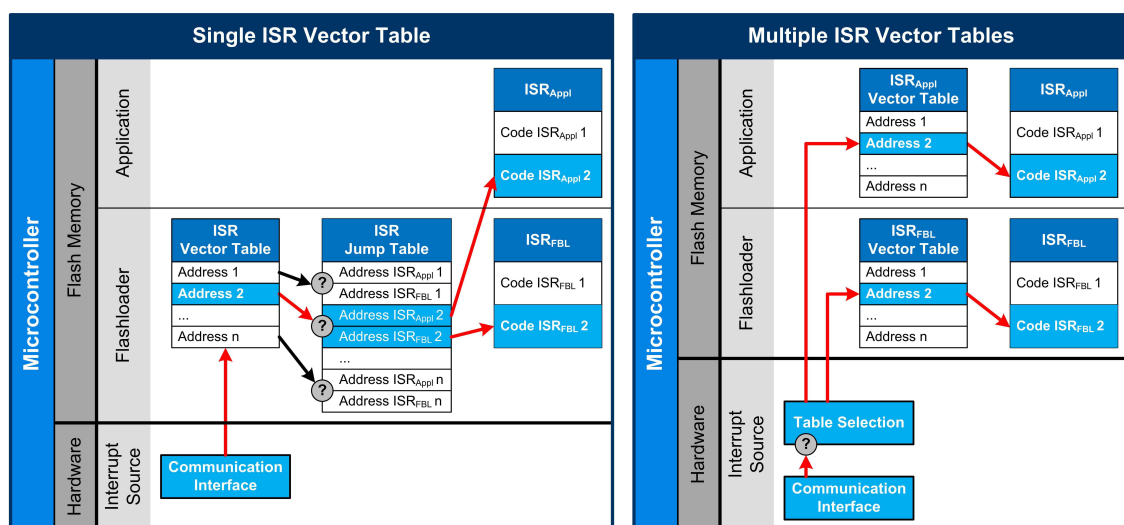


Figure 6.2-1: Single ISR vector table vs. multiple ISR vector tables

Software approach based on single ISR vector table

An approach to solve the problem is the implementation of an additional ISR jump table. Figure 6.2-1 depicts the general software based method for a single ISR vector table system.

For each ECU mode different ISRs are implemented. The base address of each ISR allocated in the different, mode specific areas (application or flashloader) is stored within an additional ISR jump table. The base address for the ISR selection within the jump table is stored in the ISR vector table. The execution of an interrupt is processed in the following manner (refer to figure 6.2-1):

- 1) An interrupt source (e.g. a communication interface etc.) initiates an interrupt.
- 2) The microcontroller hardware selects the corresponding base address of the ISR jump table entry.
- 3) Within the ISR jump table the corresponding start address of the ISR is selected depending on the currently active ECU mode.
- 4) The ISR is executed within the currently active software.

Hardware approach based on multiple ISR vector tables⁶²

The support of multiple ISR vector tables requires a table selection mechanism implemented in microcontroller's hardware. In that case today's typically implemented ISR selection mechanism is still usable. The ISR selection mechanism is working in the following manner (refer to figure 6.2-1):

- 1) After microcontroller's basic initialisation the currently active ECU mode is set in a table selection register. The ISR vector table is allocated in the memory section of the corresponding ECU mode's software.
- 2) An interrupt source (e.g. a communication interface etc.) initiates an interrupt.
- 3) The microcontroller hardware selects the corresponding base address of the ISR vector table based on the ISR table selection information.
- 4) Within the ISR vector table the corresponding start address of the ISR is selected and the corresponding ISR is executed within the currently active software.

6.2.2 Discussion

Today ISR selection by hardware is state of the art for all microcontrollers. The innovation is the selection of different ISR vector tables based on an additional selection

⁶² Submitted for patent: IP-Number P813195/DE/1 (11.10.2008) – Document Number 102008051390.3 by Ralf Schmidgall / Daimler AG

information which is processed in hardware. Of course, the software solution will also work but the hardware approach provides some benefits.

ISR vector table allocation

If only one ISR vector table is available and shall be used by both software parts (application and flashloader), this basic ISR vector table has to be allocated within the flashloader memory area. This is necessary to guarantee functionality also if no application software is programmed (refer to figure 6.2-1).

Programming flexibility

The software solution provides less flexibility for the software development process. If an ISR start address for the application software moves by any reason, new Flashloader software is necessary to modify the ISR address within the jump table. The hardware approach solves that problem. A re-allocation of the ISR results in a new address value of the ISR vector table which is also part of the application software. Hence, all relevant address information are allocated within the same memory space.

Execution speed

The software solution requires a two jump strategy to execute an ISR. This might be a runtime disadvantage (or problem) for critical software where a very fast reaction to an event is required (e.g. airbag activation etc.). If the ISR selection is executed by the currently available hardware mechanisms, the address evaluation (ISR address vector) for the interrupt processing is very fast by direct register access. The selection of the second ISR vector table is done by adding an offset to the basic ISR vector table address.

6.3 Conclusion

With focus on ECU's embedded software reprogramming process the above discussed hardware solutions provide two advantages compared to the corresponding software implementations: 1) higher signal or information integrity and 2) higher execution speed.

Signal integrity

If erasing the memory is not necessary during a reprogramming sequence (refer to section 2.4), skipping this erase process results in a total reprogramming time reduction of several seconds. The decision to skip or to execute the erasing process is based on the memory status information. A misinformation will have important effects because microcontroller's memory retention time can not be guaranteed. Hence, if this information is provided by microcontroller's memory hardware, it might be a more trustable information than a software implementation, because the possibility to falsify the signal is reduced.

This is also correct for the interrupt source detection in hardware and a direct jump to the ISR vector table.

Execution speed

With focus on software reprogramming the direct effect of the increasing execution speed is small because there are only a few time critical activities. Nevertheless, the benefit of hardware interrupt usage is visible because this provides the possibility to do concurrent activities, e.g. data reception via communication interface and decompression of the previously received data block.

Potential for increasing performance

The potential to speed up the reprogramming process of both discussed approaches is quite different. Also the effort to implement the discussed approaches is significantly different, too. As discussed in chapter 1, cost pressure within the automotive area is high and therefore the relation between potential vs. effort has to be discussed, too. Figure 6.3-1 depicts the relationship.

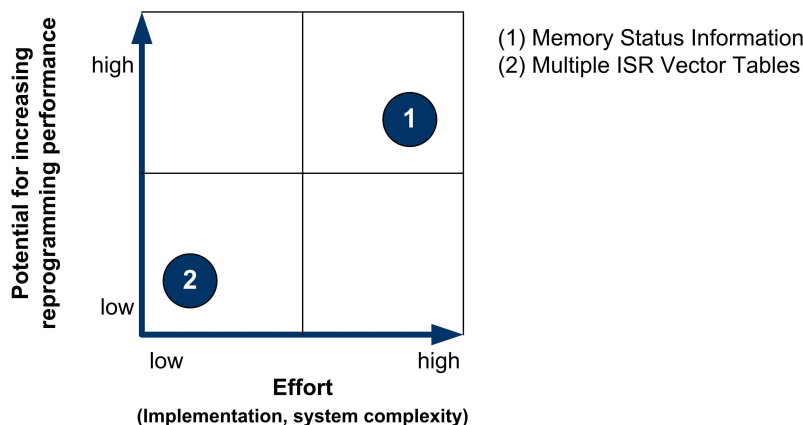


Figure 6.3-1: Potential vs. effort of hardware implementation

The memory status information provides high potential but requires high effort. If not only the memory status, but also the conditions to guarantee data retention shall be analysed, the effort is quite evident. On the other hand, a trustable signal will accelerate the erase process.

The effort to implement multiple ISR vector tables seems to be low because a simple offset to the address calculation might be sufficient. The offset addition is based on the information about the currently active software. On the other hand, the potential for process acceleration is smaller. Nevertheless, the implementation in hardware provides at least a simplification of the flashloader software and reduces software's complexity and therefore the possibility to make mistakes.

7 Network architecture

Content

7.1 Introduction	127
7.1.1 Networking issues	127
7.1.2 Network types	128
7.2 Routing nodes (Gateways).....	129
7.3 Routing strategy.....	130
7.3.1 Analysis	130
7.3.2 Discussion	131
7.4 Conclusion.....	135
7.4.1 Routing strategy.....	135
7.4.2 Network design	136
7.4.3 Summary	139

This chapter is intended to discuss the influence of the network elements on the total reprogramming process performance.

Typical vehicle networks are organised depending on the different functionalities that are necessary within a modern car. The global network is divided into several domains (refer to figure 2.1-2). Each domain encapsulates special vehicle functionalities like power train systems, infotainment systems, driver assistance systems etc. The ECUs within a domain communicate via a bus system that fulfils communication requirements to solve the domain specific functional assignments sufficiently, e.g. via low speed CAN within the comfort or body domain, via FlexRay for driver assistance and regulation systems etc. The domains are coupled by gateways. If ECU's signal is necessary within another domain, the gateway is routing that signal into the corresponding domain.

In the past, CAN was the established automotive bus system. The different domains differed only by the used CAN bandwidth. Due to the smaller number of ECUs within a car in the past, networks were flat and the different domains were decoupled by only one gateway. Of course, software reprogramming was also an issue in the past, but because of the smaller software size the resulting reprogramming time was not critical. Today vehicle networks are quite complex for several reasons:

The vehicle functionality has been increased during the last years. Due to that the number of ECUs has been increased and the communication demand increased, too. Sub-networks have become necessary to handle the communication's bus load and to guarantee stable communication. The sub-networks are coupled by gateways.

Due to new innovative functionality the requirements on data communication have been increased, too. For complex driver assistance systems, busses with guaranteed latency times are necessary. FlexRay was established for regulation systems.

Due to the cost aspects as discussed in chapter 1 the same network is used for software reprogramming purpose as for normal vehicle system's functional communication. Unfortunately the dedicated bus systems, that solve ECU application software's functional communication requirements, are not optimised for software reprogramming aspects. This topic was discussed in chapters 3, 4 and 5, where possibilities were discussed to accelerate data transfer on field bus systems.

During the software reprogramming process, the PCU and the ECU are exchanging data. If both components are directly interconnected, the reprogramming process performance depends on a) PCU's bus access performance, b) ECU's bus access performance and c) the bandwidth of the communication link (performance of the protocol stack). In case the ECU is part of a network and more than two (field) bus systems are part of the communication link between PCU and ECU, the performance of that network also has to be taken into account. In that case an overall communication link has to be divided into several sub-links on the corresponding (field) bus systems which are connected by coupling (routing) nodes like bridges or gateways. Thus, the overall communication performance and therefore the reprogramming process performance depends on a) the bus access performance of the PCU and the ECU, b) the different bus systems' bandwidth (performance of the protocol stack) as well as c) the data routing performance of the routing nodes.

With focus on ECU's application software reprogramming the network performance has to be analysed, because data routing delays are delays in data transmission and enlarge the overall reprogramming process (refer also to appendix B) [Sch10].

7.1 Introduction

The term “*network*” defines the physical interconnection of two or more nodes (e.g. ECUs). A.S. Tanenbaum et al. define that “two computers are said to be interconnected if they are able to exchange information” [Tan10-1].

As discussed in chapter 1 the design of today’s automotive networks is driven by requirements for application’s functional communication and costs. Software reprogramming aspects have not been an issue in the past. Due to increasing ECU’s software and the resulting extension of reprogramming times with increasing costs reprogramming aspects become more interesting. Now the communication network components as a part of the communication link between PCU and ECU have to be analysed and discussed with the aim to accelerate data transfer. Derived from the analysis, design rules for future automotive network design are possible to fulfil ECU’s application function communication as well as ECU’s software reprogramming.

7.1.1 Networking issues

In [Cou01-1] James Coulouris et al. define the following network issues: performance, scalability, reliability, security, mobility, quality of service and multicasting. With the intention of reprogramming automotive ECUs and data transfer acceleration within an automotive network only performance and reliability are important. Scalability is an independent design issue, mobility is not in focus for automotive networks and security is currently not relevant for data transfer acceleration. Security aspects will become an issue for software reprogramming if secured data transfer, i.e. encrypted data, is required and the data encryption results in additional protocol overhead that reduce the net data transfer ratio. The quality of service is given because if the ECU is reprogrammed, the Flashloader has only the assignment to reprogram the ECU and therefore multicasting is not necessary.

Performance

The performance parameters of primary interest were those affecting the data transfer speed: data transfer rate and latency [Cou01-1].

Data transfer rate and the influencing factors (Protocols etc.) are discussed in chapters 3 and 4. The latency is discussed here and is visible as the time delay for transmitting received messages from a source bus system to a target bus system.

Reliability

Reliability of networks based on automotive field bus systems is typically given and not a problem. Communication failures usually occur due to receiver’s or sender’s application software (e.g. protocol implementation, buffer handling etc.) rather than network errors.

For example, CAN or FlexRay implements complex CRC to identify bit errors within the data stream.

7.1.2 Network types

For embedded systems an “embedded network” couples at least two field bus systems. Two different network types are classified: homogeneous networks and heterogeneous networks. Figure 7.1-1 depicts an overview.

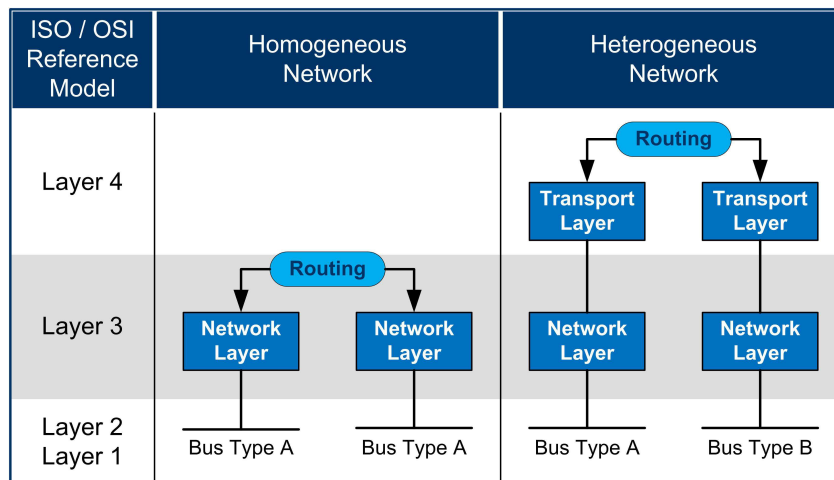


Figure 7.1-1: Network Classification

Both network types have some characteristics which have a significant impact to the network performance.

Homogeneous Networks

Homogeneous networks couple equal bus types. With focus on embedded systems, that means equal field bus systems (refer to section 2.5.1). Within homogeneous networks it is possible to route a PDU directly on ISO/OSI reference model layer 3 (refer to figure 7.1-1). This is possible because the network layer PDUs have an identical format and therefore only address information have to be analysed. A PDU received from a source bus system could be send without further activities or PDU modifications on layer 3 or upper layers on the target bus system.

Heterogeneous Networks

Heterogeneous networks couple different bus types. In the context of embedded systems that means different field bus systems (refer to section 2.5.1). Because of unequal network layer PDUs (e.g. different number of payload, different address methods etc.) the data (SDU) have to be received completely (i.e. reassembled on layer 4 – transport layer) before a new transmission on the other bus system can be initiated. Hence, routing is placed on ISO/OSI reference model layer 4 (refer to figure 7.1-1).

7.2 Routing nodes (Gateways)

There is no uniform naming convention for routing elements. Within embedded systems (especially within the automotive area) network coupling elements are always named as *gateways* because they are able to decide whether a message or signal from a sender must be routed to another bus system or not. The term *router*, as known from PC networks, is typically not used in the automotive area. In [Zim10-13] W. Zimmermann and R. Schmidgall classified three different bus coupling elements: (1) *transceiver* for the ECU bus access, (2) *repeater* to enlarge the physically length limits of a bus system (e.g. FlexRay star or coupling of truck and trailer bus systems) and (3) *gateways*. Nevertheless, automotive *repeaters* are typically implemented as *gateways* because in many cases not all signals and messages⁶³ are transferred to the other bus systems or domains. Hence, a selection method is implemented to process a kind of selective routing. For the following discussion a more detailed definition for *gateways* is necessary. A possible differentiation criterion is the highest layer (refer to figure 7.1-1), on which routing will be performed. Hence, we distinguish between layer-3-gateways⁶⁴ and layer-4-gateways to classify the routing strategy. Within the AUTOSAR layered software architecture routing is always executed by the *PDU-Router (PduR)* module, but on top of different layers (refer to figure 7.2-1).

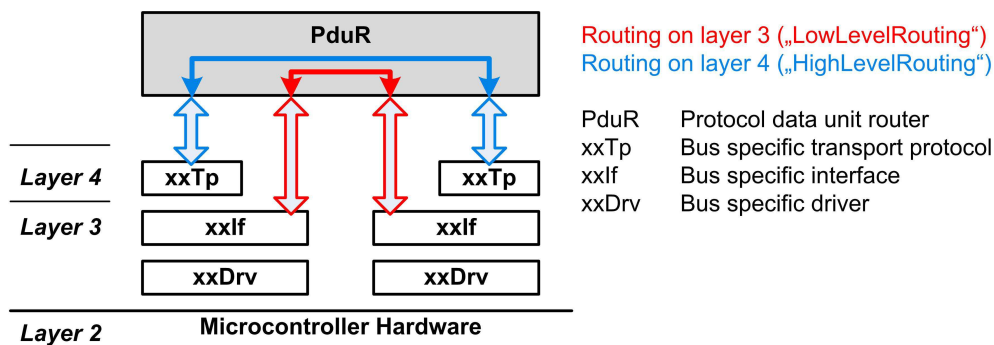


Figure 7.2-1: Routing within the AUTOSAR layered software architecture

⁶³ Within a modern vehicle several thousand signals and messages are emitted by the ECUs' application software. But not all signals are relevant for all other ECUs in all domains. For gateway configuration only the signals for a receiver from other domains are relevant. For example, the ignition status (on/off, clamp 15 or clamp 30 etc.) as well as the speed signal are important information for all ECUs and are therefore routed into all domains. On the other hand internal signals of the power train domain are not relevant for the body domain and are therefore blocked and not routed.

⁶⁴ Within the automotive industry the differentiation between layer 3 and layer 4 are not always clearly structured. Because of resource limitations, implementations for embedded systems have combined both layers (refer to chapter 3). The CAN transport layer protocol as defined in [ISO 16765-2_3] specifies the mapping of network layer PDUs. Nevertheless, the basic idea is to route ahead of layer 3. That means that only address aspects have to be taken into account but not protocol control information (PCI).

7.3 Routing strategy

Depending on the coupled network types different routing strategies are required. Figure 7.3-1 depicts an overview.

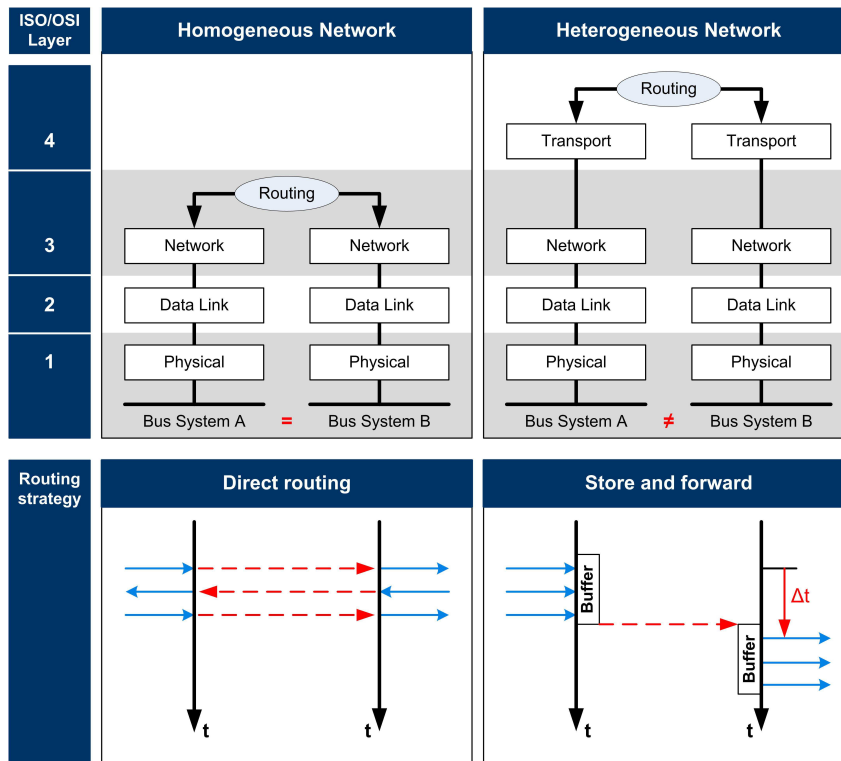


Figure 7.3-1: Routing strategy

7.3.1 Analysis

Typically routing within networks is basically the problem of finding the shortest path between two nodes. Many of today's common algorithms are based on the shortest path algorithm published by R.E. Bellman [Bel57] and the algorithm for large networks published by Ford and Fulkerson [For62]. Compared to typical problems of routing information through LAN networks or the internet, the problem within automotive networks is very simple. Especially for software reprogramming purpose the network aspects could be significantly simplified: (1) the connection links are stable, i.e. not floating, and (2) there is exactly one way for the diagnostic connection link, i.e. there are no alternative possibilities to establish a communication link from the PCU to the ECU and vice versa. Hence, no complex algorithms are necessary to calculate the best path because there is only one.

Therefore, for automotive networks the routing problem is reduced to the question of how fast received data can be processed and transmitted onto the target bus. But here the routing strategy has a significant influence on the routing performance.

Direct routing (routing “*On-the-fly*”)

Within the direct routing strategy a received PDU is immediately processed and routed from the receiver side to the sender side. The PDU is neither analysed nor modified. The PDU is only copied to the transmission buffer.

This method is possible if the physical bus systems on receiver and sender side are equal (same type), e.g. CAN to CAN routing. In that case the network layer PDUs are equal on both bus systems and the network-layer-gateway or layer-3-gateway can copy the PDUs without further activities. Protocol conformity is given by FIFO⁶⁵-buffers for the PDUs on transmission or sender side.

Within the automotive industry the term “*routing-on-the-fly*” was introduced for layer-3 routing. In the normal communication or network terminology of network nodes this is typically done by a simple repeater but if there are other PDUs in the network which are not routed to the other connected bus systems, a selective routing is given and therefore it is a layer-3-gateway. Figure 7.3-1 depicts an abstract view to that routing mechanism.

Store and forward

Compared to a layer-3 gateway a layer-4 gateway is necessary if the physical bus systems differ (e.g. CAN to FlexRay routing). In that case the network layer PDU can not be simply copied to the transmission buffer because typically the network layer PDUs are different. Hence, a routing is possible only on the top of layer 4 (transport layer). As a consequence the complete transport layer protocol has to be processed on the receiver side. If the layer 4 SDU is completely received the routing process can be executed and the SDU is transmitted via target bus system’s transport layer protocol. In that case the differences on the network layer are not relevant.

7.3.2 Discussion

Due to the coupling of different networks different routing mechanisms are necessary and different strategies are possible. As discussed in section 7.1.1 routing performance is an important criterion for gateways and has a significant impact for data transfer acceleration and reprogramming performance.

Performance

A gateway’s routing performance within a network can be calculated as the time which is necessary to forward data received on a source bus system to a target bus system. Figure 7.3-2 depicts an abstract overview.

⁶⁵ FIFO... First In First Out – A mechanism for data buffering within a queue.

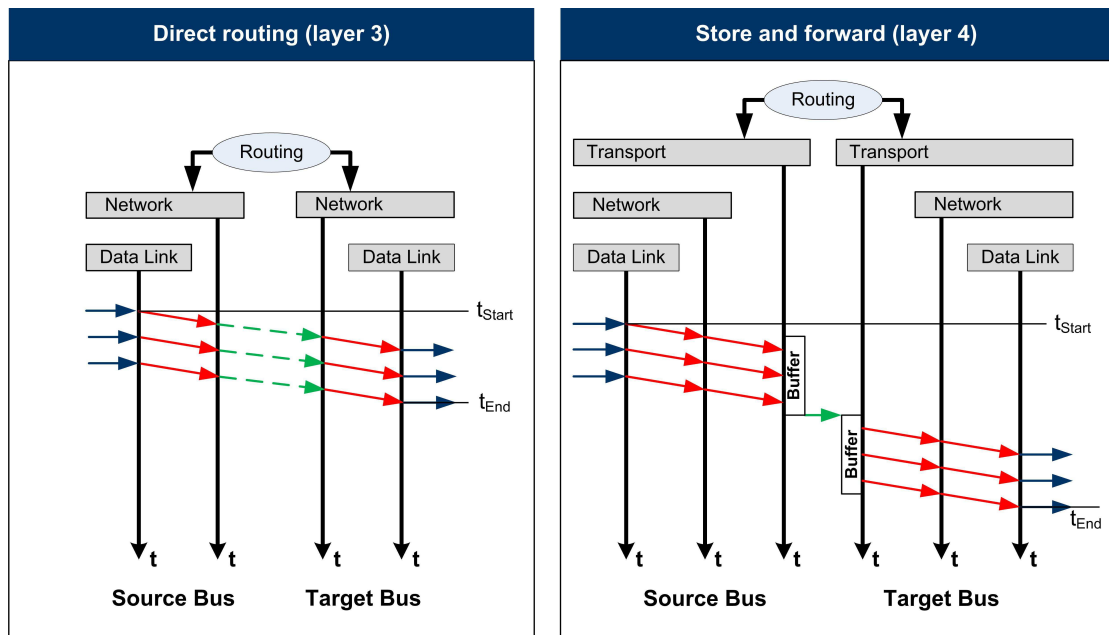


Figure 7.3-2: Routing performance

The routing performance is calculated as

$$\text{Routing Performance} = \frac{\text{Data Length}}{t_{\text{End}} - t_{\text{Start}}} \quad (7.3-1)$$

For the routing performance discussion and the comparison of routing strategies it is assumed that source and target bus systems have equal bandwidths. In that case figure 7.3-2 depicts that a direct routing strategy on layer 3 has a higher performance because the delay based on routing processing is very short. *Store and forward* strategies require more time because data reception on layer 4 and therefore the complete transport layer protocol handling has to be finalised until routing could be performed. After the single routing step transport layer protocol handling for the target bus system is required. In sum the time for reception on the source bus and transmission of data on the target bus on a “*store and forward routing*” system is longer than on a “*direct routing*” system.

Resources

The different routing strategies also have an impact on the required gateway resources. As depicted in figure 7.3-2 the CPU load for a “*store and forward*” mechanism is higher than for a direct routing. This is because of the additional layer interaction between layer 3 and layer 4 and the processing of the transport layer protocol.

For data routing a buffer is an important resource. Routing on layer 3 requires buffer especially if the bandwidth of the target bus system is less than the source bus system (e.g. routing from $CAN_{500\text{kBit/s}}$ to $CAN_{125\text{kBit/s}}$). In that case an adequate queue for the data to transmit is necessary. Depending on the differences between the bandwidths the queue

depth could vary. For layer 4 routing a complete transport layer SDU must be stored during data reception before routing can be processed. For CAN transport protocol according to ISO 15765-2 this are 4095 bytes in maximum. For FlexRay communication layer protocol according to ISO 10681-2 this are 65.535 bytes in maximum. For communication in parallel those buffer sizes have to be allocated for each concurrently active channel.

As illustrated in chapter 1 the cost pressure for automotive systems is high and therefore resource management is an important issue. Due to that fact several hundred kByte of RAM for routing buffers are not a realistic scenario. On the other hand the coupling of different physical bus systems via layer 4 routing is state of the art. Hence, a combination of routing strategies is necessary to reduce required RAM size for economic aspects and fulfil the technical requirements for coupling bus systems via transport layers.

Partly store and forward

A *partly store and forward* approach reduces the demand for buffer resources, as well as increases system's performance in case a layer-4 routing is required. If a defined data volume (threshold) is received, routing is processed. Figure 7.3-3 depicts the details.

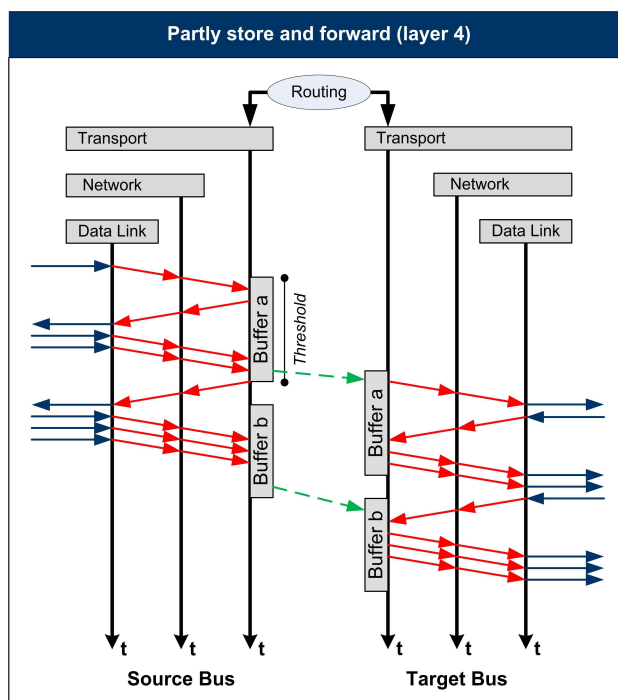


Figure 7.3-3: Partly store and forward routing strategy

If the defined threshold value is less than the maximum possible SDU length of the transport layer protocol, the required buffer resources could be reduced significantly. The data flow on the source bus system has to be controlled by the transport layer protocol's flow control mechanism and depends primarily on the availability of a free buffer. On the

other hand, a smaller buffer (small threshold) results in more flow control PDUs (PDUs without payload) and reduces performance.

The overall system's performance is increasing if more than one buffer is available and these buffers can be used alternatively. The number of different buffers depends on the bandwidth relation between source and target bus system. This is necessary if the bandwidths of the source and target bus systems are different. The system has to be configured in such a way that no additional delay or communication gap is visible on the slower bus system. The relation is given as:

$$r_{\text{Bandwidth}} = \frac{f_{\text{TargetBus}}}{f_{\text{SourceBus}}} \quad (7.3-2)$$

If the bandwidth of a source bus system does not equal the one of the target bus system, the relation $r_{\text{Bandwidth}}$ is not equal to 1 ($r_{\text{Bandwidth}} \neq 1$). In that case two buffers are sufficient because until the slower bus system has processed the first buffer (reception or transmission) the second buffer is processed by the faster bus system (transmission or reception). In case both bus systems have an equal bandwidth, $r_{\text{Bandwidth}}$ is equal to 1 ($r_{\text{Bandwidth}} = 1$). In that case only two buffers are not sufficient. It is possible that a buffer is completely processed before the other system has freed the other buffer (mutual exclusion principle). Therefore a third buffer is necessary to provide a free buffer if the currently faster bus system is requesting a new buffer. If no free buffer is available, the system has to interrupt data transmission by flow control processing and continue later on. Figure 7.3-4 depicts the ideal number of buffers.

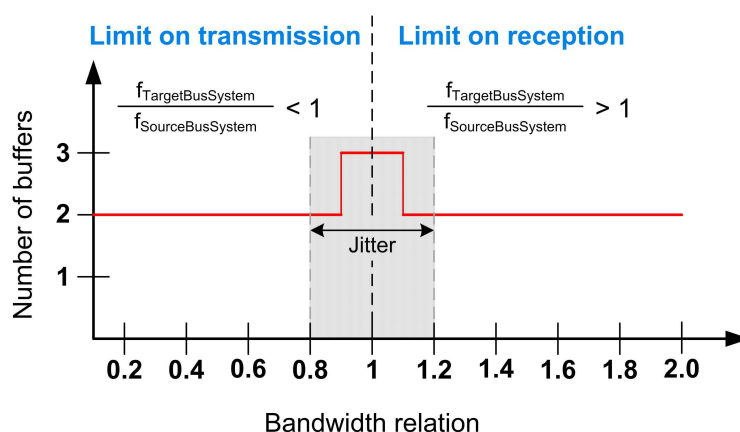


Figure 7.3-4: Ideal number of buffers

The jitter and the resulting boundary for the step from two buffers to three buffers have to be analysed for each gateway individually. It depends on internal criteria like microcontroller performance, interrupt service processing runtime, task management etc. and could not be calculated in a generic way.

“Store and forward” method vs. tunnelling method

Another approach known from the telecommunication industry is the tunnelling method. A PDU from one system is integrated completely as payload (SDU) into the PDU of a second system. For example, a CAN protocol stack's network layer PDU is directly forwarded to the FlexRay protocol stack's transport layer. However this approach provides only benefits if the target bus system that is tunnelled is (a) significantly faster than the source bus system and (b) the source bus system is only a transfer bus where as the destination node is a gateway. Typically automotive networks provide no mere transfer bus systems and therefore tunnelling is not possible.

Cascaded sub-networks

Within an automotive network it is possible that more than two different networks are coupled via gateways for a link between PCU and ECU. In that case data are packed and repacked several times by the corresponding transport layer protocols. Maximum performance is given if 100% bus load is processed on the slowest sub-bus system. The number of necessary buffers (as discussed above) has to be calculated for each gateway.

7.4 Conclusion

An additional influencing parameter of automotive embedded system's reprogramming performance is the network performance. The network performance depends on two parameters:

- a) The data transfer performance (protocol stack performance) of the coupled sub-links (refer to chapters 3, 4 and 5).
- b) The gateway's routing performance depending on the routing strategy based on the network type (heterogeneous and homogeneous)

7.4.1 Routing strategy

The network type (heterogeneous or homogeneous) is based on the type of coupled bus systems. The routing performance of a gateway is influenced by the network type, i.e. the coupling of equal or different bus systems. For equal bus systems a coupling on layer 3 - network layer is possible (homogeneous network). Different bus systems are typically coupled on layer 4 - transport layer (heterogeneous networks). Due to the routing layer different routing strategies are possible. Figure 7.4-1 depicts different routing strategies in relation to possible routing performance and the required resources.

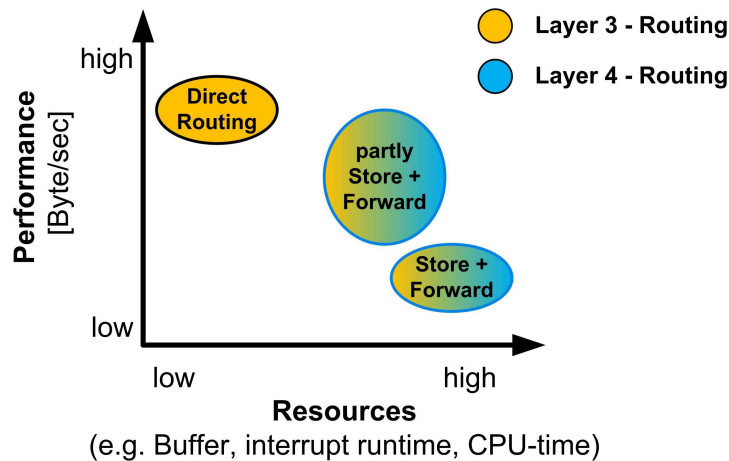


Figure 7.4-1: Routing performance vs. resources

Direct routing provides best performance vs. resources relation because each received PDU on a source bus system could be routed directly to the target bus system. A buffer queue for the layer 3 PDUs is necessary to prevent overwriting in case of jitter on equal bus systems with equal bandwidth or in case of transmission accumulation on equal bus systems with different bandwidth.

Store and forward routing provides less performance vs. resources relation because routing is only possible when all payload data have been received via the source bus system. While reception on the source bus system is ongoing, no data transmission on the target bus is processed. The result is a delay in transmission and therefore less performance for reprogramming. Additional buffer for the complete payload is required to store the SDU until all data have been received (CAN via ISO 15765-2: 4095 byte; FlexRay via ISO10681-2: 65535 byte). Finally, routing on layer 4 increases demand on inter-layer communication and enlarges the CPU runtime for routing processing.

Partly store and forward routing strategy is a good compromise between the other strategies. Routing runtime is equal but, depending on the current situation routing could be configured either on performance requirements or on RAM resources requirements.

If prices for microcontroller RAM decreases in future and the CPU power increases, the *partly store and forward* strategy might be nearly as powerful as direct routing because resource disadvantages (memory size, runtime, etc.) are no longer critical.

7.4.2 Network design

The discussion about network impacts for the reprogramming process results in a discussion about network design aspects. Future automotive networks have to be designed to support the software reprogramming issue, too. As described in chapter 1, network design was influenced by application software's functional communication requirements and cost

aspects in the past. Up to now software reprogramming is an important, but mostly uncritical issue. Currently it is processed via the same network than functional communication to save costs. The costs are still an important aspect but the different contributions to calculate a total cost of ownership are changing: The relation between invest in network infrastructure (gateways, bus systems etc.) and reprogramming costs is moving towards increasing programming costs. Due to that potential commercial disadvantage, an additional parameter has to be taken into account for network design aspects: reprogramming time limit.

Design aspects based on reprogramming time limits

Within several scenarios an upper limitation for the reprogramming time is given. Within a vehicle plant, for example, the assembly line timing defines the upper limit. Software programming has to be finalised within one or, in good cases, in a well defined number of timing cycles. If the reprogramming time is fix, either ECU's upper memory limitation depends on bus system bandwidth or a bus system has to be selected, whose bandwidth is sufficient to program the given memory. Figure 7.4-2 depicts the relation between ECU memory and reprogramming time limitation.

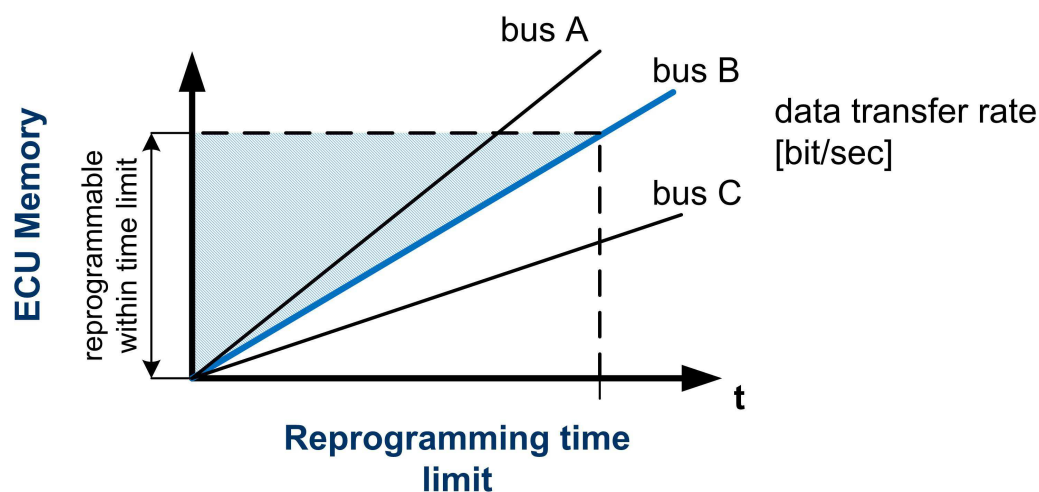


Figure 7.4-2: Design based on timing limitations

According to the following equation for given bus systems the maximum reprogramming time is limited by the ECU's memory.

$$\text{DataTransferRate} = \frac{n_{\text{EcuMemory}}}{t_{\text{Reprogram}}} \quad (7.4-1)$$

The data transfer rate calculation is important especially for new developed ECUs. With focus on reprogramming it has to be distinguished, whether the ECU could be repro-

grammed within time limitations via the regular connected bus system for normal application communication. Alternatively the ECU has to be connected to another bus system with more bandwidth. Of course, in that case all other network design issues (e.g. maximum bus load, cable length, costs, weight etc.) have to be taken into account to find the best economic solution. Based on that decision the gateways on the communication link can be designed, too. Because of the complex optimisation process and the large number of influencing parameters this analysis has to be supported by network analysis tools⁶⁶.

Design aspects for reprogramming in parallel

Within distributed systems it is also possible that more than one ECU have to be reprogrammed⁶⁷. In that case the execution in parallel is a powerful approach to accelerate the overall process and to save time. The network design has to take care that reprogramming parallel is possible and concurrent communication links can be supported. Hence, the data transfer ratio C (refer to formula 7.4-2 below) of the different sub-bus systems within the network is important.

Figure 7.4-3 depicts a network that supports reprogramming in parallel. It is divided into several sub-bus systems coupled by gateways. The developed formula 7.4-2 defines the basic requirement to guarantee sufficient data transfer rates on all sub-bus systems for a concurrent communication traffic.

$$C_{\text{SourceBus}} \geq \sum_{i=1}^{i=n} C_{\text{TargetBus}_i} \quad (7.4-2)$$

⁶⁶ The Symta Vision GmbH develops and sells the tool „SymTA/S“ to analyse diagnostic communication via network. It is a „model-based solution for timing design, performance optimisation and timing verification for real-time systems“ [Sym]. During the „5th Symtavision NewsConference on Timing Analysis“ R.Schmidgall spoke about „Diagnostic Communication – A Challenge For Network Analysis“ [Sch11]. The presentation depicts the different aspects of diagnostic communication (inclusive software reprogramming aspects) and the challenges and complexity if the different communication stack protocols shall be analysed for realtime network analysis (refer to appendix D).

⁶⁷ The introduction of AUTOSAR's layered software architecture allows to divide system functionality from the physical ECU. That means that a system is developed or modeled first in an abstract way. Later on the different function modules are mapped to one or more physical nodes (ECUs) (refer to [AUTOSAR] key features). In case an error occurs within a system that is mapped to several physical ECUs, the functional dependencies might require reprogramming of all ECUs. This scenario is also possible for non-AUTOSAR based systems if functional dependencies exist.

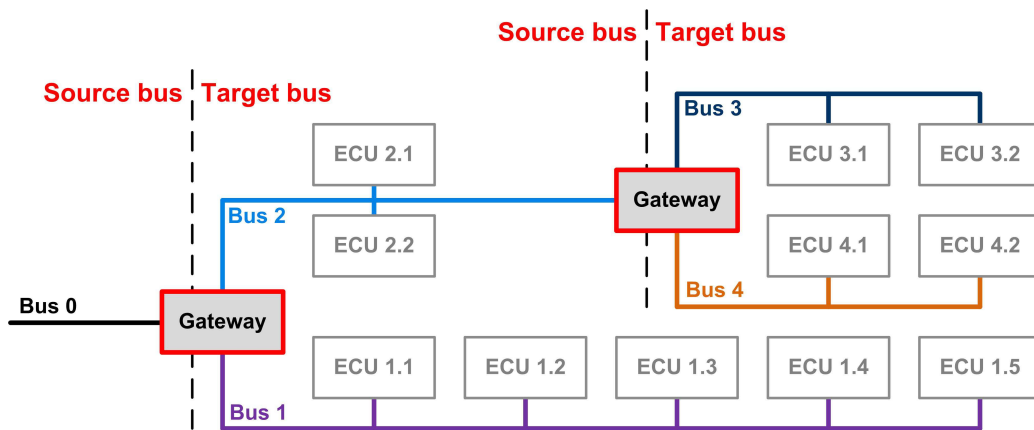


Figure 7.4-3: Source bus and target bus definition for reprogramming in parallel

As discussed in chapter 3 maximum performance for data transfer is reached for ECU reprogramming if the bus load for data transfer is up to 100%. Hence, for reprogramming in parallel this must be possible on each sub-bus system, too. According to formula 7.4-2 the network design shall provide source bus systems that are able to generate 100% busload on all target bus systems. In case of cascaded networks with several gateways, this has to be guaranteed for each source bus - target bus relation.

For the network in figure 7.4-3 the bandwidth for bus 0 shall be according to formula 7.4-2:

$$C_{\text{Bus0_max}} \geq C_{\text{Bus1_max}} + C_{\text{Bus2_max}}$$

$$C_{\text{Bus0_max}} \geq C_{\text{Bus1_max}} + C_{\text{Bus3_max}} + C_{\text{Bus4_max}}$$

Nevertheless, even if the network (bus 0 or bus 2) does not support the maximum data transfer rate according to formula 7.4-2, reprogramming in parallel is possible, however without 100% busload on each bus system. The strategy for reprogramming in parallel is discussed in chapter 8.

7.4.3 Summary

Network design and the routing strategy have a deep impact to the data transfer performance via network and also an impact to the overall ECU software reprogramming performance.

The necessity of different physical bus systems for application software communication results in heterogeneous networks. Due to that the gateways become more complex (routing on layer 3 vs. routing on layer 4) and require more resources (RAM, execution runtime, etc.) to provide the same data transfer rate as for homogeneous networks.

With focus on reprogramming several ECUs in parallel (refer to chapter 8), the available data transfer rate is an important parameter. The possibility to reprogram in parallel is built in during network design phase when the bandwidths of the sub bus systems are defined. The smallest bandwidth on the communication link between the PCU and the ECU will affect the link performance.

For future network design it is necessary to have different, scenario oriented views to the same network: a) the established view on application function's communication and b) a view for reprogramming aspects. Figure 7.4-4 depicts the approach.

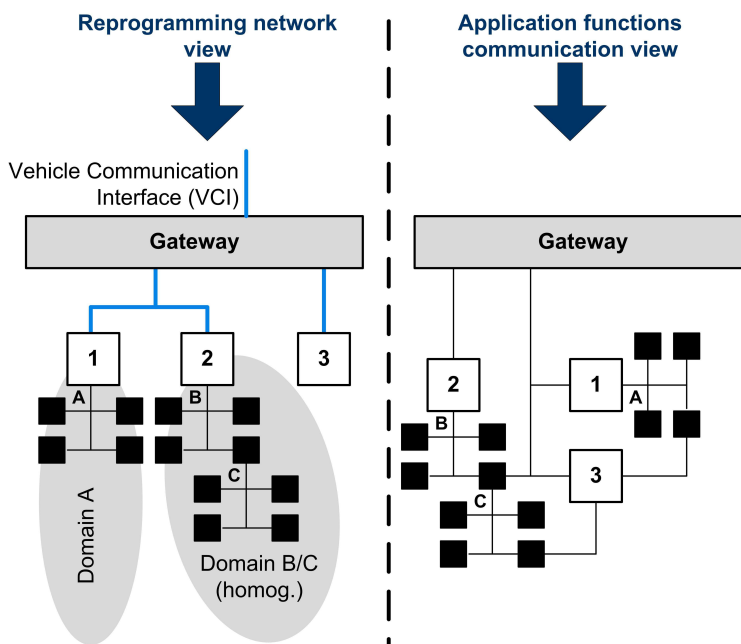


Figure 7.4-4: Different, scenario oriented views to the same network

For software reprogramming purpose some design requirements are necessary to guarantee high performance:

- a) If a cascaded network with sub-domains is necessary (Domain B/C) the network shall be homogeneous.
- b) If the bandwidth can not be equal on all bus systems, the bandwidth shall be at least decreasing from the vehicle communication interface to the sub-domains according to formula 7.4-2.

Finally, there is the question whether a network which is designed for software reprogramming will solve the normal ECU application communication, too?

Of course, this is a new approach for automotive network design, but the given cost pressure requires to think about communication via the same network. Why not move the design priority?

8 Reprogramming in parallel

Content

8.1 Introduction	142
8.2 ECU schedule calculation	143
8.3 Discussion	147
8.4 Conclusion.....	149

As mentioned in chapter 1, several trends are visible within the automotive industry that influence the ECU reprogramming process. Of course, automotive ECUs' application software sizes are increasing continuously, but during the last years this trend has affected only single ECUs with their own application software. As a result the total reprogramming time for those ECUs has been increased. Nevertheless, an additional trend is visible: increasing system complexity based on the distribution of functionality to several ECUs. Jan Danenberg et al. described in [Dan07] that 90% of all innovation will be build in software. The trend is to combine available information of ECUs to new functionalities. Especially for driver assistance systems new innovative systems based on available signals. The AUTOSAR software architecture model supports this trend. In AUTOSAR hardware independent modelling of functions (inclusive all communication signals) is possible. After modelling has been finalised the function could be distributed to several host ECUs. The communication is done by the basic software⁶⁸.

A disadvantage of the new trend of model based software development and distributed system functionality is the risk that in case of an error all ECUs which are hosting a part of the system are affected by reprogramming. In that case software reprogramming of all affected ECUs in parallel is a powerful approach to reduce the overall reprogramming time. Within this chapter the approach of reprogramming ECUs' application software in parallel is discussed.

⁶⁸ "Basic software" (BSW) is a term of the AUTOSAR nomenclature and comprehend all software modules responsible for the basic functionality of an ECU. In the AUTOSAR architecture the basic software is below the Runtime Environment (RTE). Refer to [Aut11].

8.1 Introduction

To reprogram several ECUs' application software, concurrent communication links have to be processed. There are different scenarios to reprogram ECUs in parallel.

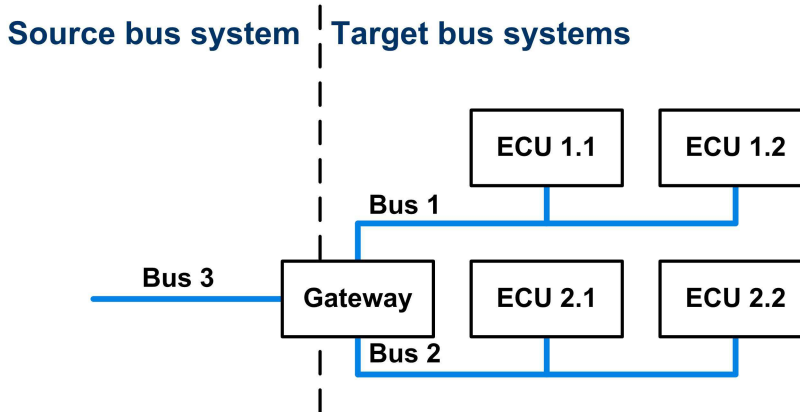


Figure 8.1-1: Network classification

Figure 8.1-1 depicts a network to illustrate the different scenarios as described in table 8.1-1.

Table 8.1-1: Reprogramming scenarios

Scenario	Description	Example
1	Single ECU on a single bus system.	Only ECU 1.1
2	Single ECU on different bus systems.	ECU 1.1 and ECU 2.1 in parallel
3	Multiple ECUs on a single bus system.	ECU 1.1 and ECU 1.2 in parallel
4	Multiple ECUs on different bus systems.	ECU 1.1, 1.2, 2.1, 2.2 in parallel

The scenario 1 (reprogramming of a single ECU) is only for list's completeness. The scenarios 2, 3 and 4 have to be discussed for reprogramming in parallel. Especially scenario 3 and scenario 4 seem to be quite complex, because the bandwidth has to be shared between several ECUs on the same bus system.

Simplification of the scenarios

For the theoretical approach it is assumed that bandwidth of the source bus system is sufficient to process all concurrently active communication links. Nevertheless, the scheduling of the active communication links is quite complex. A simplification is possible if the research results of chapter 3, 4 and 7 are taken into account:

- a) Best data transfer performance is reached if 100% busload is reached. In that case only one communication link allocates the complete bus system's bandwidth to communicate with only one ECU. This is possible by implementing double buffered data reception and optimisation of the communication protocol stack.
- b) The network is designed to support 100% bus load on the slowest communication link sub-bus system for the corresponding communication link. In best case this is the target bus system itself.

In that case scenario 3 (reprogramming of multiple ECUs on a single bus system) could be mapped to scenario 1, a sequential reprogramming process of two ECUs on the same bus system. Scenario 4 (reprogramming of multiple ECUs on different bus systems) can be mapped to scenario 2, where single ECUs on different bus systems are reprogrammed.

8.2 ECU schedule calculation

The overall time to reprogram all ECUs depends on the schedule that defines in which order the different ECUs are processed. To calculate the best schedule some previous steps are necessary.

- 1.) Evaluation of the slowest sub-bus segment (bandwidth limiter) on the communication link for each ECU.
- 2.) Calculation of the expected reprogramming time for each ECU based on the corresponding communication link's slowest sub-bus system's bandwidth.
- 3.) Definition of a priority list depending on the expected reprogramming time. The ECU with the longest reprogramming time has the highest priority.
- 4.) Definition of a processing schedule. The ECUs are arranged based on the priority list and the available bandwidth on the network.

Bandwidth capacity utilisation

To calculate an optimised schedule the available bus system's bandwidth must be evaluated. This is possible, if the time-discrete view to the network is moved to a continuous bandwidth view.

At one discrete point in time only one communication link can be supported by a transmitter. A real concurrent communication is not possible, because at one discrete point in time only one bit of a well defined connection is transmitted. If transmission of one connection link's PDU has been finalised a new PDU, maybe of another communication link, could be processed.

If two bus systems are connected via a gateway (best practice) and both bus systems have an equal bandwidth and 100% bus load is produced, no concurrent communication on another link will be possible.

If two bus systems with different bandwidths are connected via gateway (best practice) and 100% bus load is produced on the slower bus system, some bandwidth is free on the faster bus system. This view is helpful to arrange different ECUs on a bus system. Figure 8.2-1 depicts that example. The time discrete view is moved to a continuous bandwidth view.

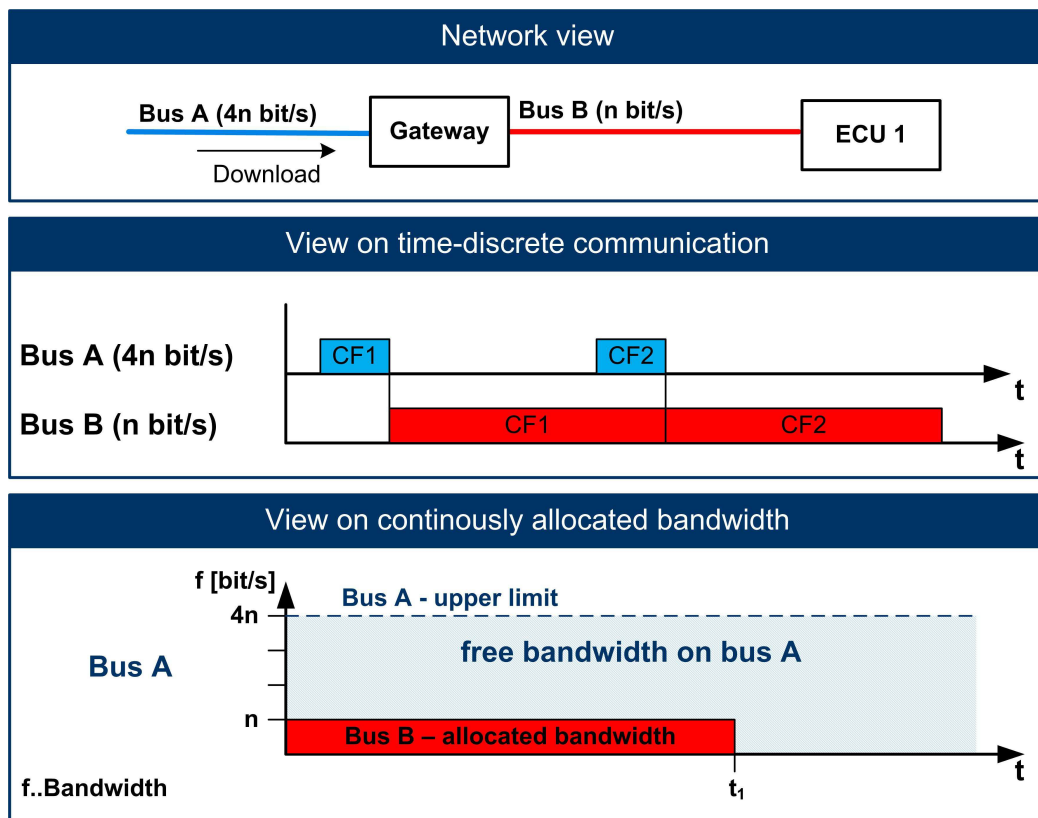


Figure 8.2-1: Bandwidth capacity utilisation

The time t_1 is the required time to transmit all data via that communication link. The time t_1 is calculated as

$$t_x = \frac{S}{f} \quad (8.2-1)$$

S .. Software Size [bit]; f .. bandwidth [bit/s]; t .. data transfer time [s]

Up to t_1 the bandwidth n is allocated for the communication link with ECU1. The remaining bandwidth is $3n$ ($4n - n = 3n$) and can be used for other communication links. Additional communication links can be established as long as the remaining bandwidth n of bus A is more than zero ($n > 0$).

Example with 4 ECUs on three sub bus systems

The following example illustrates the method to arrange ECUs for reprogramming in parallel. Figure 8.2-2 depicts a network with 4 bus systems and 4 ECUs. The source bus system A does not support the necessary bandwidth to process 100% busload on all sub-bus systems (refer to chapter 7). Nevertheless, reprogramming in parallel is possible.

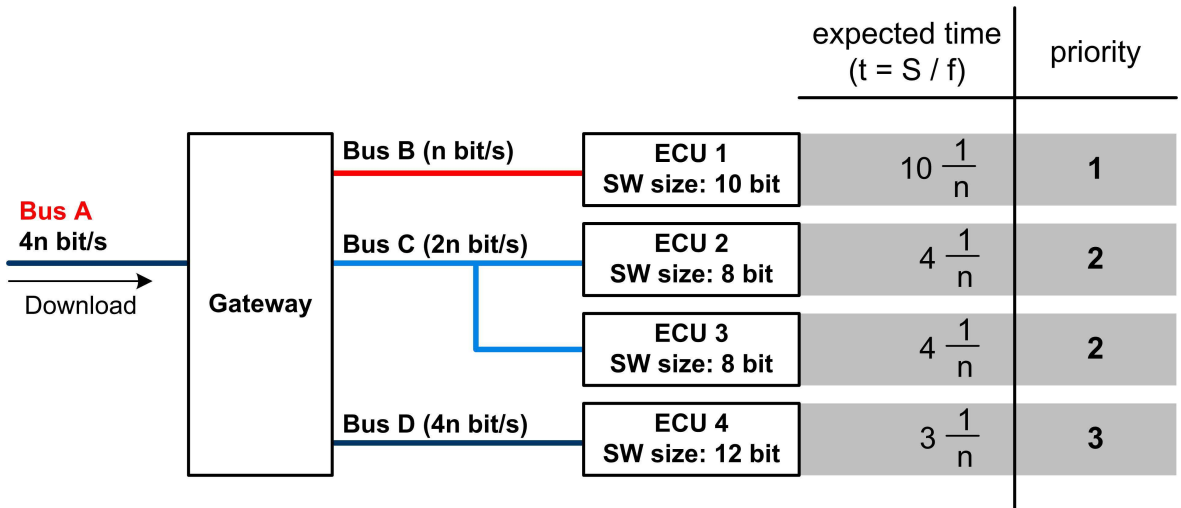


Figure 8.2-2: Priority calculation on a network with 4 ECU

The expected time for the complete data transfer via the communication link can be calculated according to formula 8.2-1 and under consideration of the slowest sub-bus system on the corresponding communication link. The priority is given based on the expected data transfer time.

The schedule can be defined under consideration of the rules below:

- 1) Start with the highest priority (longest expected data transfer time)
- 2) Calculate always if the remaining bandwidth is sufficient for the next ECU.

$$f_k \leq f_{\text{SourceBusSystem}} \quad (8.2-2)$$

$$f_k = \sum_{x=\text{Priority1}}^{\text{Priorityk}} f_{\text{ECU}_x} \quad (8.2-3)$$

If a recalculation of the finalisation time t_x is not necessary based on the remaining bandwidth, then:

- 3) Arrange only one ECU from a sub-bus system at a single point in time. Note that 100% bus load is processed on the limiting sub bus system.

Figure 8.2-3 depicts a possible schedule to reprogram the given network in parallel.

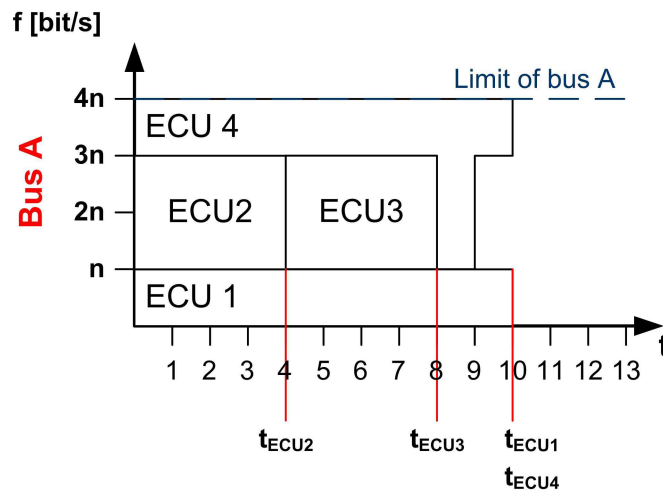


Figure 8.2-3: Schedule calculation on a network with 4 ECU

ECU1 has the highest priority because the expected data transfer time is $t = 10 \cdot 1/n$. ECU2 and ECU3 have equal priorities. Additionally they are both connected to the same bus system. Hence, only ECU2 is scheduled in the first phase. Until $t = (4 \cdot 1/n)$ the remaining bandwidth is only n bit/s ($4n - n - 2n = n$) and therefore ECU4 could not be arranged with full bandwidth. As a result the data transfer time has to be recalculated. The bandwidth limiting bus system is now bus A.

At t_{ECU2} the data transfer to ECU2 has been finalised. Data transfer to ECU3 could be scheduled (priority is higher than to ECU4, no concurrent communication on the limiting bus C). Communication of ECU4 is ongoing with bandwidth n bit/sec.

At t_{ECU3} the data transfer to ECU3 has been finalised. For communication to ECU4 now a bandwidth of $3n$ is available for the remaining software size of 4 bit (8 bit have been transmitted with a bandwidth of n bit/s). Recalculation of the finalisation time results in

$$t_{\text{ECU4}} = t_{\text{ECU3}} + \left[\frac{4 \text{ bit}}{3 \frac{\text{bit}}{\text{s}}} \right] = t_{\text{ECU3}} + 2 = 10\text{s}$$

At t_{ECU1} data transfer to all ECUs had been finalised. Reprogramming all ECUs sequentially requires $21 \cdot a/n$ seconds. Reprogramming in parallel requires only $10 \cdot a/n$ seconds and reduces reprogramming time to 47%.

Bandwidth capacity utilisation for cascaded bus systems

The schedule development method will work also for cascaded bus systems. Figure 8.2-4 depicts the principle.

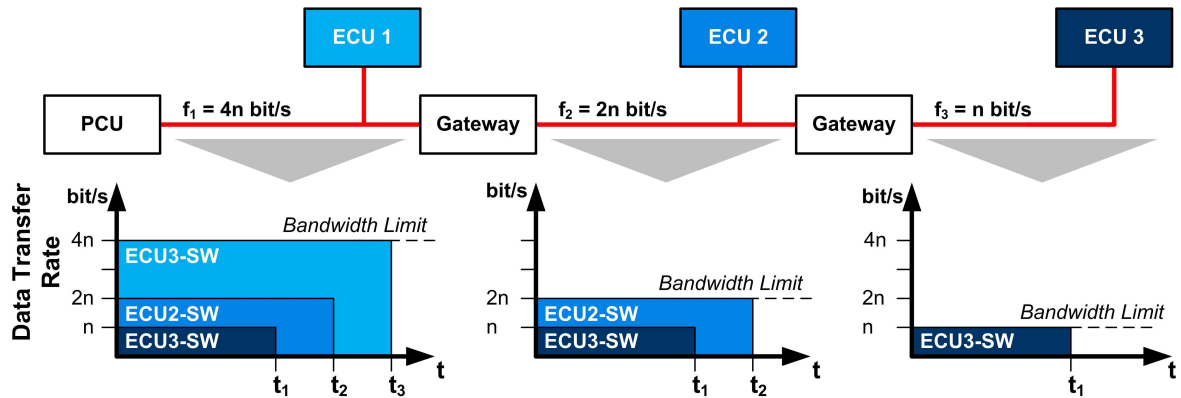


Figure 8.2-4: Schedule calculation for networks with cascaded bus systems

8.3 Discussion

The method to calculate a schedule for ECU reprogramming is based on a simple principle. The slowest sub bus system's bandwidth on the communication link between PCU and ECU is allocated also on all other sub-bus systems. The remaining bandwidth can be used to establish another communication link to another ECU. The aim is to utilise the given bandwidth well.

The method is simple because the assumption that 100% bus load on a bus is possible. This can be reached by implementing double buffered data reception on the receiver ECU.

100% busload not possible

The basic method will work also, if 100% bus load will not be possible by a single ECU. However, the calculation of the real bandwidth for each ECU is quite complex, but if the bandwidth is determined, the normal schedule calculation can be processed. Without double buffered data reception the reprogramming time cannot be compensated (refer to chapter 3). Hence, a forecast for the reprogramming time is necessary. However, the physical reprogramming time can drift by aging of the microcontroller. Unfortunately, such a drift is only detected, if a data transfer to the ECU and therefore a physical reprogramming process is executed. The measurement of the time between the transmission of the

request's⁶⁹ last data transfer frame and the reception of the response⁷⁰ reduced by the network's Transmission runtime represents the physical reprogramming time and the internal processing time. In worst case the given forecast based on the physical programming time is not longer valid and a recalculation is necessary. In that case a recalculation of the schedule might be necessary, too.

How to get the information about sub-bus bandwidths?

The schedule calculation method is based on the knowledge of the network's sub-bus systems bandwidth. Hence, a method is necessary to provide the network topology for the schedule calculation.

The ASAM MCD-2D⁷¹ standard (market name *ODX – Open Diagnostic Data Exchange*) provides abstract information about the topology of the network. In [Zim10-14] W. Zimmermann and R. Schmidgall give an introduction to the ODX sub document „*VEHICLE-INFO-SPEC*“ (*ODX-V*). This document is defined to provide the logical links from an external diagnostic test tool to the corresponding ECU. Unfortunately, the ODX-V is currently not prepared to describe a complete network in detail with all information required to calculate a schedule (e.g. bus' bandwidth, ECU arrangement etc.). S. Karic analysed in his bachelor thesis the ODX-V data model and developed a method to describe all relevant information for reprogramming in parallel [Kar11]. The conclusion of this work is that the ODX-V data model is currently not applicable to support reprogramming in parallel, because some important information (e.g. gateway information etc.) are not part of the data model.

Without ODX-V the topology information has to be stored in another format or document for the schedule calculation.

Reprogramming of gateways

It is possible that the gateway application software shall be reprogrammed. This can occur if routing relations will change, e.g. in case a new ECU (new innovation) is introduced into a network. As described in chapter 1, typically ECUs flashloader software is optimised only for software reprogramming purpose because of less ROM or Flash memory resources. Typically gateways in flashloader mode will route only broadcast

⁶⁹ Diagnostic Request (refer to UDS)

⁷⁰ Diagnostic Response (refer to UDS)

⁷¹ ODX defines a unique, open XML exchange format for diagnostics data.

diagnostic⁷² messages but no normal application communication PDUs of other ECUs. Hence, it is necessary to have information during schedule calculation whether an ECU is a gateway or not. This information could be also a part of ODX-V.

Due to the above described gateway behaviour in flashloader mode, gateways has to be reprogrammed within an own schedule. However, it has to be taken into account that gateways of cascaded networks have to be reprogrammed sequentially.

8.4 Conclusion

Reprogramming of ECUs application software in parallel is a powerful approach to reduce the total reprogramming time. However, some prerequisites are necessary to be able to reprogram in parallel: a) the network shall be designed for that approach and b) topology information shall be available for schedule calculation.

The calculation of the reprogramming schedule has the aim a) to utilise the available bandwidth well and b) to create an arrangement to reprogram all ECUs in the shortest possible time. Gateways shall be reprogrammed separately in sequential order because gateways typically stop routing in flashloader mode.

Best practice for schedule calculation based on ASAM MCD-2 (ODX)

The *VEHICLE-INFO-SPEC* document of the ODX standard is partly not applicable to support the reprogramming process. Hence, some optimisation proposals for ODX-V are provided to the ASAM MCD 2 standardisation working group.

In case the ASAM MCD-2 (ODX) standard model provides the network topology information in future, the offboard activities can be managed completely by the ODX data model. Figure 8.4-1 depicts an overview in principle.

The *ODX-F* document provides all information about ECU's application software (e.g. software size, possible compression algorithm etc.) The *ODX-D* document provides all diagnostic information of the ECU (e.g. supported diagnostic services according to the

⁷² Diagnostic broadcast messages are typically send by the PCU and received by all ECUs to keep the ECUs into diagnostic session. Typically an ECU supports a "normal default session" at least two diagnostic sessions (extended diagnostic session and reprogramming session). Diagnosis is only possible in a non-default session and is initiated by the diagnostic service request "\$10 – Session Control" (refer to UDS). An important broadcast message is the cyclic transmitted diagnostic service request "\$3E – Tester Present". It signals that a diagnostic test tool is connected and prevents ECUs fall back from non-default (diagnostic) session to normal default session. Hence, broadcast messages have to be routed also if the gateway is in boot mode to keep the domain into diagnostic mode.

UDS standard). The *ODX-V* document provides the network information (e.g. network topology, sub-buses' bandwidth, gateway declarations etc.).

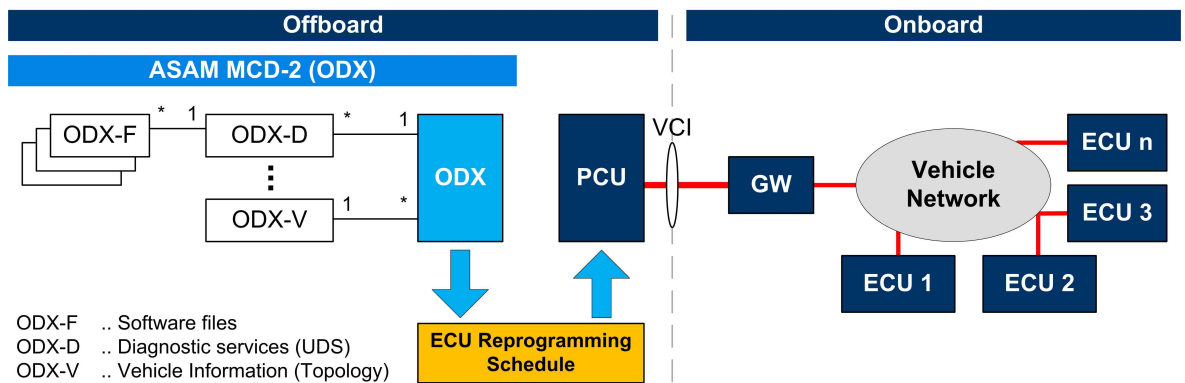


Figure 8.4-1: ODX integration to support reprogramming in parallel

By evaluating the *ODX-F* document and the *ODX-V* document the reprogramming time for each ECU can be calculated. These values are used to calculate the reprogramming schedule and process reprogramming. The sequence for schedule calculation is as listed below:

- 1) Calculate reprogramming time of each ECU.
- 2) Select all ECUs which are marked as a gateway and skip them from the list. Gateways are reprogrammed separately before the other ECUs are processed.
- 3) Calculate the priority depending on the reprogramming time. Highest priority for longest reprogramming time
- 4) Calculate the schedule depending on the priority. The following rules have to be taken into account:
 - a) Bus system's bandwidth limitations have to be considered.
 - b) If 100% bus load is possible (e.g. by double buffered data transfer) only one ECU on the final target bus system shall be scheduled at one point in time.
 - c) Arrange the ECUs that on the first source bus (from PCU to first gateway) in that way that as less as possible bandwidth is free.

Based on this algorithm, ECUs' application software reprogramming in parallel will be a powerful approach to reduce the total reprogramming time significantly.

9 Magnetoresistive RAM

Content

9.1 Introduction	152
9.2 Discussion	153
9.3 Case study to the differential file approach.....	154
9.4 Conclusion.....	156

As discussed in the chapters before the current software reprogramming process is significantly influenced and limited by the currently given memory technology. Of course, the Flash memory technology provides benefits which were never given before for embedded systems by the old ROM mask technology and, of course, without the established reprogramming process for Flash memory the product costs especially within the automotive industry, will be still higher. Nevertheless, the Flash memory technology has some restrictions which constrain and limit an optimized reprogramming process. A real quantum transition will be possible if the established Flash memory technology is replaced by the new proposed MRAM technology (Magnetoresistive Random Access Memory) in microcontrollers. Some disadvantages of flash memory caused by the inherent technology can be eliminated by the employment of possible MRAM technology. With focus on reprogramming time the MRAM technology provides essential advantages. In contrast to the currently established flash memory technologies MRAM semiconductors store the information not by electrical, but by magnetic load elements.

Of course, currently there is no microcontroller available that supports MRAM on chip. However, Freescale Semiconductor provides MRAM as an external memory device [Fre07-1]. Nevertheless, it seems that this memory technology will be the next evolution step for embedded systems' memory and therefore this memory technology is discussed within this thesis.

This chapter is intended to discuss the impact of MRAM technology to the reprogramming process of embedded systems as well as to depict the necessary changes for software development process.

9.1 Introduction

In contrast to currently established memory technologies, MRAM semiconductors store the information not using electrical, but by magnetic load elements. The effect is based on the fact that certain materials change their electrical resistance if they are influenced by magnetic fields. Alfred Hammerl and Halit Bag give an overview of the different magnetoresistive effects [Ham03]. R.C. Sousa et al. reviewed the progress of the MRAM research process and provide a briefly overview to “conventional MRAM operations” like reading or writing a bit [Sou05].

Magnetic Tunnel Junction (MTJ) effect

In an MRAM cell the information zero (0) and one (1) are represented by the orientation of magnetic fields and is based on the Magnetic Tunnel Junction (MTJ) effect. A MTJ semiconductor has a three-layer structure. It consists of two magnetic layers and an insulation layer. One of the magnetic layers is a fixed ferromagnetic layer and has a fixed orientation (fixed magnetic layer). The other magnetic layer can change its magnetic polarization (floating magnetic layer). It is aligned either in the same orientation as the fixed layer (parallel magnetic orientation) or in the opposite (opposing magnetic orientation). Although not shown in figure 9.1-1, a bit line and digit line are located above and below the MTJ. The electrical resistance of the memory cell changes depending to the magnetic orientation of the floating magnetic layer. According to the electrical resistance a high or low current could occur. A current switch converts the binary information low current and high current to voltage levels (low current = 0_{bin} ; high current = 1_{bin}).

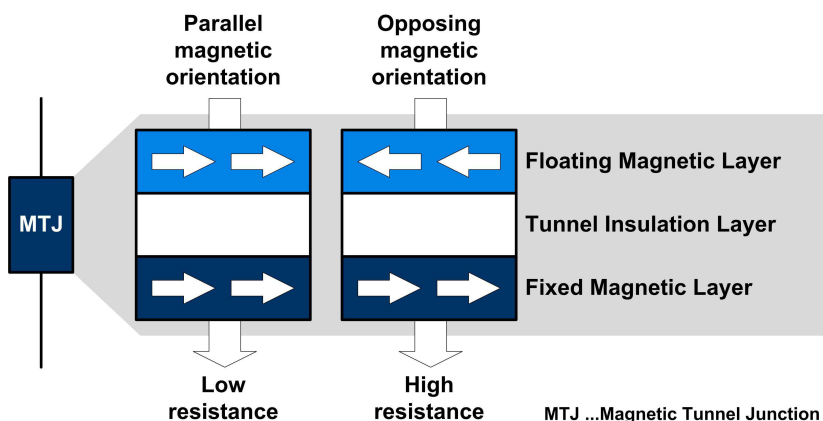


Figure 9.1-1: Bit information storage based on the MTJ effect

The MRAM technology does not need any electrical current in order to hold the stored information. Once the magnetic adjustment is made the variable magnetic layer remains static, i.e. no further current is required.

Comparison of MRAM to other memory technologies

MRAM adopts the advantages of several memory technologies available today. Similar to flash memory or EEPROM (Electrical Erasable and Programmable Read Only Memory), a non-volatile data retention takes place, i.e. program code and data are sustained without power supply. MRAM reduces the power consumption because the refresh pulses as required for DRAM are no longer necessary. The data access is very fast (cf. SRAM) and the cells are small which results in a high device integration level. Table 9.1-1 depicts an overview to the typical memory parameters and compares MRAM and other memory technologies [Fre07].

Table 9.1-1: Comparison of expected MRAM features with other memory technologies [Fre07]

	MRAM	SRAM	DRAM	Flash	FeRAM
Read Speed	fast	fastest	medium	fast	fast
Write Speed	fast	fastest	medium	low	medium
Non-Volatile	yes	no	no	yes	yes
Low Voltage	yes	yes	limited	limited	limited
Complexity	medium	low	medium	medium	medium

The advantages of MRAM based systems are quite evident. The main advantages of MRAM compared to Flash memory technology, with respect to reprogramming activities, is the byte-wise access and the possibility to overwrite data without an initial memory erase phase.

9.2 Discussion

Some disadvantages of Flash memory's reprogramming process caused by the inherent technology can be eliminated by the employment of possible MRAM technologies.

Erase process skipping

In section 2.4 the typical reprogramming process of a flash memory based embedded system was described. Normally Flash memory technology does not allow the overwriting of programmed memory cells without prior erasing memory partitions/blocks. It is currently not possible to erase a single memory cell.

MRAM technology allows overwriting of individual programmed memory cells without prior erasing of the cell. Therefore the erase step within the reprogramming process is no longer required. Table 9.2-1 shows the potential to save processing time based on the erase time of two different, currently state of the art microcontrollers' Flash memory.

Table 9.2-1: Example of microcontroller's erase time for Flash memory

Infineon TriCore TC 1797 [TC1197]	min.	typ.	max	unit
Program Flash Erase Time per 256 kByte Sector	-	-	5	s
Freescal MC9S12XEP [MC9S12X]				
Freescal MC9S12XEP [MC9S12X]	min.	typ.	max	
Program Flash Erase Time per 1024 Byte		20	21	ms
Normalized to 256 kByte		5.1	5.4	s

Based on this data given by the manufacturer's data sheets [TC1197, MC9S12X] the predicted total erase time for a 2 MByte on-chip flash memory is up to 40 seconds. This time can be saved potentially in case of skipping the erase process because of using MRAM. With focus on the total amount of automotive embedded systems as discussed in chapter 1 these potential might be considerably higher.

Byte-wise read/write access

In a Flash memory complete physical memory sectors must be erased and reprogrammed. Erasing the complete physical sector is necessary no matter if a complete memory section or only a few bytes have changed. Thus, the data for reprogramming the complete physical sector always has to be transferred and programmed. MRAM technology allows read/write access basically for each single byte (alignment has to be taken into account). This byte-wise read/write access allows the usage of the method of differential file update as discussed previously in section 5.4 but without the Flash memory's disadvantage of storing the non-changed bytes into RAM mirror. Only the real differences of the old and new compiler/linker output file have to be transferred and reprogrammed. Of course, the overall effort of this method is high but the benefit is enormous. The data transfer time could be reduced significantly and this will finally solve the initial problem.

9.3 Case study to the differential file approach

In section 5.4 the approach of software reprogramming based on differential file data transfer was discussed in principle. The assumption in section 5.4 was that software bug

fixing is the most important reason for software reprogramming and 80% of bug fixings result in less than 1 kByte OP-code changes and only 20% in more than 1 kByte. As a result of this assumption only a few bytes within a memory sector/partition need to be changed compared to the total software size.

The case study below continues the case study of section 5.4 and compares the differential file approach for Flash memory technology systems as well as for MRAM technology based systems.

According to chapter 2 and as mentioned above Flash memory requires that the corresponding memory part is previously erased before it can be programmed. If the RAM resources are not given to mirror the corresponding memory part, this memory part will be erased and the data for that part have to be transferred completely. For MRAM technology only the differences have to be transferred.

The microcontroller parameters (Flash memory section size etc.) are based on Infineon's TriCore TC1197 microcontroller. According to table 9.1-1 it is assumed that the write speed to MRAM is equal to existing Flash memories and therefore we use the write speed value of the TC1767, too. We assume that the size of the modified OP code is 1 kByte and all modifications are constrained within 4 memory pages⁷³.

The case study based on the assumptions and parameters as listed below:

Modified OP code size:	1 kByte
Flash memory size:	16 kByte, 128 kByte, 256 kByte
Erase performance:	51.2 kByte/s (refer to table 9.1-1)
Program performance:	50 kByte/s ⁷⁴ [TC1197, TC1197-2]
Payload:	8 Byte (pure CAN protocol without any transport protocol)
FrameLength:	123 bit (refer to table 4.1-1)
CAN bit rate:	500 kbit/s

The memory erase time t_{Erase} is calculated as

$$t_{\text{Erase}} = \frac{\text{MemorySize}}{f_{\text{Erase}}} \quad (9.3-1)$$

The data transfer time t_{Transfer} is calculated by formula 9.3-2:

⁷³ Page = 256 Byte Refer to [TC1797-2]

⁷⁴ 256 byte per page programmed in 5 ms \rightarrow (256 byte : 0.005s) : 1024 = 50 kByte/s

$$t_{\text{transfer}} = \frac{\text{DataVolume}}{\text{Payload}} \cdot \text{FrameLength} \cdot \frac{1}{\text{bitrate}} \quad (9.3-2)$$

Table 9.3-1 depicts the results of that theoretical case study:

Table 9.3-1: Differential file approach comparison to Flash and MRAM technology

Memory Type	Sector Size [TC1197]	Data volume to transfer	Erase time	Transfer time	Program. time	Total time
Flash	256 kByte	256 kByte	5.000 s	8.061 s	5.120 s	18.181 s
Flash	128 kByte	128 kByte	2.500 s	4.030 s	2.560 s	9.090 s
Flash	16 kByte	16 kByte	0.313 s	0.504 s	0.320 s	1.136 s
MRAM	256 kByte	1 kByte	0.000 s	0.031 s	0.020 s	0.051 s
MRAM	128 kByte	1 kByte	0.000 s	0.031 s	0.020 s	0.051 s
MRAM	16 kByte	1 kByte	0.000 s	0.031 s	0.020 s	0.051 s

The total time depicts a significant benefit for MRAM technology based software reprogramming process. The data transfer time as well as the reprogramming time is reduced. An erase process is not required and therefore an erase time will not occur.

Nevertheless, to get the results as depicted ahead, a corresponding software development process is a previously required. If the software is not designed and prepared to support differential file updates (refer to chapter 4) the benefits will not be realized.

9.4 Conclusion

The MRAM technology's byte-wise access allows software updates by transferring and overwriting only differences between the old and new software. Due to the reduced amount of data to transfer, the data transfer time and the physical programming time significant time savings can be made. Thus the potential cost savings of the new technologies could solve the rapidly approaching technological limitation of flash memories in modern complex embedded vehicle systems.

As introduced ahead, MRAM seems to be the next technology step for embedded non-volatile memory. With respect to software reprogramming the typical software architecture and software structure has to be modified to fulfil the initial requirements to use the differential file update approach. Only if software is designed to produce smallest possible OP-code differences between an actual and a preview file the full power of the differential file update method can be realized.

Appendix A provides a concentrated view to MRAM technology with focus to software reprogramming aspects which are discussed in several chapters of this work.

10 Case study – Software reprogramming

Content

10.1 Software reprogramming via CAN	157
10.1.1 ISO15765-2 (CAN-TP) model evaluation	158
10.1.2 ISO14229 (UDS) on CAN model evaluation.....	162
10.1.3 CAN bus baud rate optimisation.....	166
10.1.4 ISO 15765-2 (CAN TP) Flow Control parameter Block size.....	168
10.1.5 ISO 15765-2 (CAN TP) FlowControl parameter STmin	170
10.2 Application Protocol ISO 14229 (UDS) Optimisation	172
10.3 Gateway optimisation	178
10.3.1 Buffer for the partly store and forward routing strategy.....	179
10.3.2 Increasing gateways clock frequency.....	181
10.3.3 Summary	182
10.4 Software reprogramming via FlexRay	182
10.4.1 Vehicle access by CAN bus system.....	182

This chapter presents a case study to verify the quantitative models presented to evaluate software reprogramming purpose in the earlier chapters. The study provides answers to the question of performance increase based on protocol optimisations as well as the effects of data size reduction (compression).

10.1 Software reprogramming via CAN

This section is intended to verify the theoretical discussions of chapter 3 and chapter 4 by a real implementation. The flashloader implements a CAN communication stack according to the protocols ISO 11898, ISO 15765-2 and ISO 14229.

Experimental Setup

The approaches of chapter 3 and 4 to accelerate data transfer have been implemented within a prototype project for a V850 CargateM⁷⁵ microcontroller. Figure 10.1-1 depicts the evaluation test system layout.

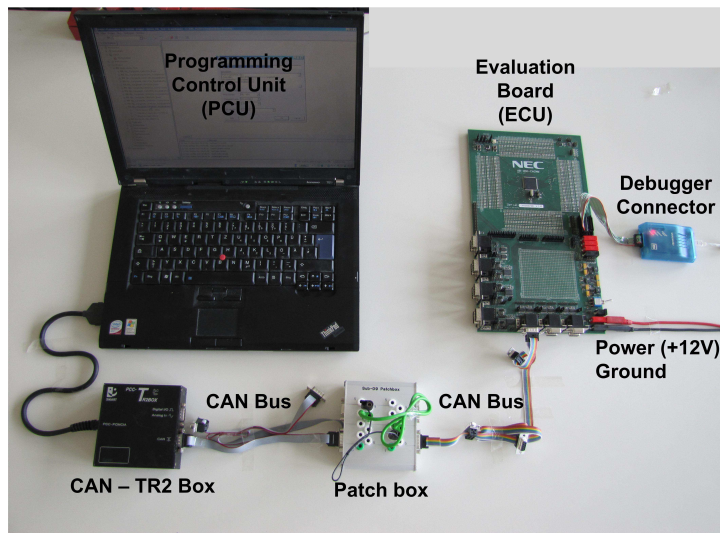


Figure 10.1-1: Test environment

PCU:	FlashCedere® V1.20 and PCCOM: V01.61
CAN - PCC-TR2Box:	S/N 0700/027-1
Evaluation Board:	NEC AB050 CAG4M ; SN CA 0050071D V1.00
Microcontroller:	V850 UPD 70F 3461 6J(A1) (“CargateM”)

The PCU (FlashCedere^{®76}) controls the reprogramming process by sending the sequence of diagnostic services as described in section 2.5.4. The CAN-TR2 Box provides PCU’s CAN interface. The connector’s pin layout of the CAN-TR2-Box and the evaluation board are different. Hence, the CAN signals (CAN-High and CAN-Low) are mapped to the corresponding pins by the Patch-Box. The debugger connector allows flashloader reprogramming for the different test scenarios.

10.1.1 ISO15765-2 (CAN-TP) model evaluation

Study’s aim

The aim of this study is the validation of formula 4.1-17. This formula represents the mathematical model to calculate the data transfer time via CAN transport layer protocol according to ISO 15765-2.

⁷⁵ Microcontroller: Renesas V850 D70F3461GJ(A1)

⁷⁶ SMART Electronic Development GmbH, Germany [Smart]

Experimental Setup

In section 4.1-2 communication via the transport layer protocol according to ISO 15765-2 was discussed. It was stated that segmented data transfer provides benefits compared to unsegmented data transfer with focus on data transfer rate. Figure 10.1-2 depicts the PDU sequence for a segmented transmission of 4,082 byte from a sender node to a receiver node.

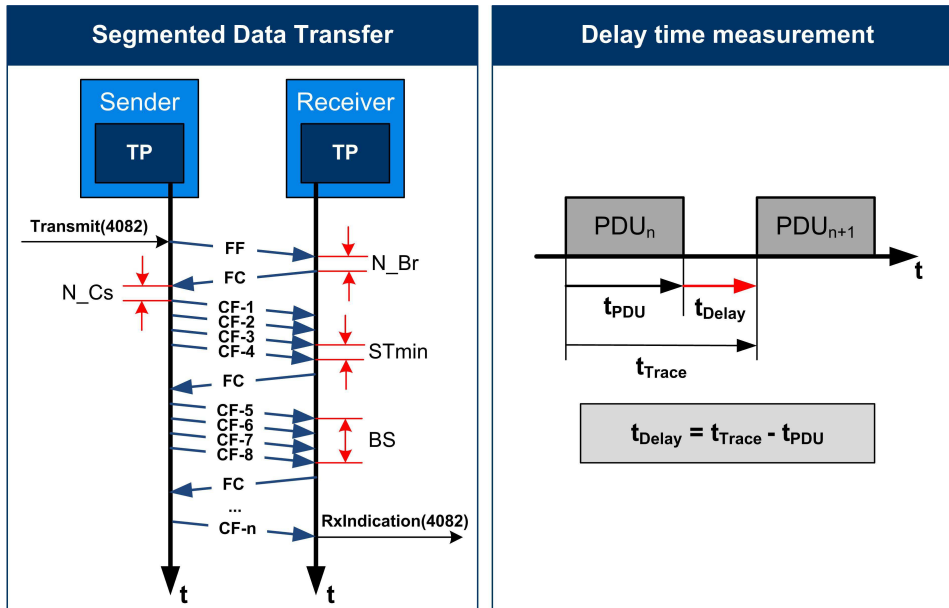


Figure 10.1-2: Segmented data transfer according to ISO 15765-2

The model depends on the configuration parameters as listed in table 10.1-1 and the system runtime parameters as listed below:

- a) Sender node's processing time N_Cs
- b) Receiver node's processing time N_Br

ISO 15765-2 – N_Br and N_Cs parameter

An additional delay is the processing time of ISO 15765-2 transport protocol's flow control handling. As discussed in section 4.1.2.4 the ECU (N_Br time) as well as the PCU (N_Cs time) requires time to process a *FlowControl* PDU. This delay is independent from the bus system's baud rate. The constant value reduces the data transfer rate. This is also an indication for the theoretical discussion about reduction of *FlowControl* PDUs to reduce processing delays by increasing the block size parameter in the transport layer protocol.

To evaluate the model the system related processing times N_Cs and N_Br have to be measured. These processing times are independent of the underlying bus system.

Table 10.1-1: PCU and ECU communication configuration parameters

Parameter	Value
CAN Baud rate	125 ; 250 ; 500 ; 1,000 kBit/s
CAN Address Mode	11 bit – normal addressed
Block size (BS)	32
STmin	0 ms

10.1.1.1 Measurement results

Table 10.1-2 shows the measurement results for the test system.

Table 10.1-2: Measurement results of PCU and ECU processing parameters

Parameters	CAN baud rate				unit
	125	250	500	1,000	kBit/s
N_Cs	2.9	2.9	2.7	2.6	ms
N_Br	0.003	0.003	0.003	0.003	ms
STmin	0.014	0.024	0.030	0.038	ms
n_{PDU_s}	603	603	603	603	
$n_{CF_PDU_s}$	583	583	583	583	
Transfer Time	665.3	363.7	215.7	144.5	ms
Transferred data $d_{SDU_ISO15765-2}$	4,082	4,082	4,082	4,082	Byte
Performance	5.99	10.96	18.48	27.59	kByte/s
	47.931	87.686	147.827	220.722	kBit/s

The measurement shows a stable parameter set for N_Cs (2.77ms) and N_Br (0.003ms). The transport layer is configured to a STmin = 0. Nevertheless, the ECU requires processing time between two *ConsecutiveFrame (CF)* PDUs.

10.1.1.2 Evaluation

In chapter 4 formula 4.1-17 was developed to calculate the net data rate for data transfer via CAN.

$$f_{Data_net} = \frac{d_{SDU_ISO15765-2}}{(n_{PDU_s} \cdot d_{PDU_ISO11898} \cdot t_{bit}) + \left(n_{CF_PDU} \cdot \left[\frac{n_{CF_PDU}}{BS} - 1 \right] - 1 \right) \cdot t_{STmin} + \left[\frac{n_{CF_PDU}}{BS} \right] (t_{N_Br} + t_{N_Cs})}$$

The integration of all configured of measured parameters results in the formula below:

$$f_{Data_net} = \frac{d_{SDU_ISO15765-2}}{(603 \cdot d_{PDU_ISO11898} \cdot t_{bit}) + 566 \cdot t_{STmin} + 19 (2.773 \cdot 10^{-3} s)}$$

Table 10.1-3: Communication model's calculation of data transfer rate

Parameters	CAN baud rate				unit
	125	250	500	1,000	kBit/s
$d_{PDU_ISO11898}$	123	123	123	123	bit
STmin	0.014	0.024	0.030	0.038	ms
Transferred data $d_{SDU_ISO15765-2}$	4,082	4,082	4,082	4,082	Byte
Performance	6.09	10.98	18.30	26.87	kByte/s
	48.758	87.877	146.425	214.978	kBit/s

Table 10.1-3 depicts the calculated data transfer rate results of the communication model based formula 4.1-17. The error between the measured and calculated data transfer rates is in a range of -1.7% to + 2.7%.

The model depicts that doubling pure CAN bit rate will not result in doubling communication performance. The reason is the relation between the PDU transfer and the nearly constant processing delays:

$$r = \frac{603 \cdot d_{PDU_ISO11898} \cdot t_{bit}}{(603 \cdot d_{PDU_ISO11898} \cdot t_{bit}) + 566 \cdot t_{STmin} + 19 (2.773 \cdot 10^{-3} s)}$$

Table 10.1-4: Communication model's calculation of data transfer rate

Parameters	CAN baud rate				unit
	125	250	500	1,000	kBit/s
Relation [$t_{Transfer} : (t_{Transfer} + t_{Delay})$]	0.91	0.82	0.68	0.50	
Theoretically max value (refer to table 4.1-4)	6.7	13.3	26.7	53.3	kByte/s
Corrected value	6.08	10.87	18.19	26.65	kByte/s
Measured value (refer to table 10.1-2)	5.99	10.96	18.48	27.59	kByte/s

Table 10.1-4 depicts the calculated relation. Due to that effect the theoretically possible maximum data transfer rate will not be reached. Table 10.1-4 depicts the theoretically possible values (refer to Table 4.1.-4 in chapter 4). According to the calculated relation, the maximum data transfer rate via CAN (1,000 kBit/s) is only 50% of the theoretically possible data rate. The comparison between the corrected but theoretically calculated values and the real measurement values proofs that theory.

10.1.1.3 Conclusion

Formula 4.1-17 works. The additional delays result in a reduction of the maximum data transfer rate for a segmented data transfer.

10.1.2 ISO14229 (UDS) on CAN model evaluation

Study's aim

The aim of this study is the validation of formula 4.1-21. This formula represents the mathematical model to calculate the download time via CAN transport layer protocol according to ISO 15765-2.

Experimental Setup

In section 4.1-3 the complete download of a reprogramming sequence based on ISO 14229 - UDS via the transport layer protocol according to ISO 15765-2 on CAN is discussed. Figure 10.1-3 depicts the PDU sequence of several segmented transmission of 4,082 byte from a PCU to an ECU (Request) and the responses from the ECU to the PCU.

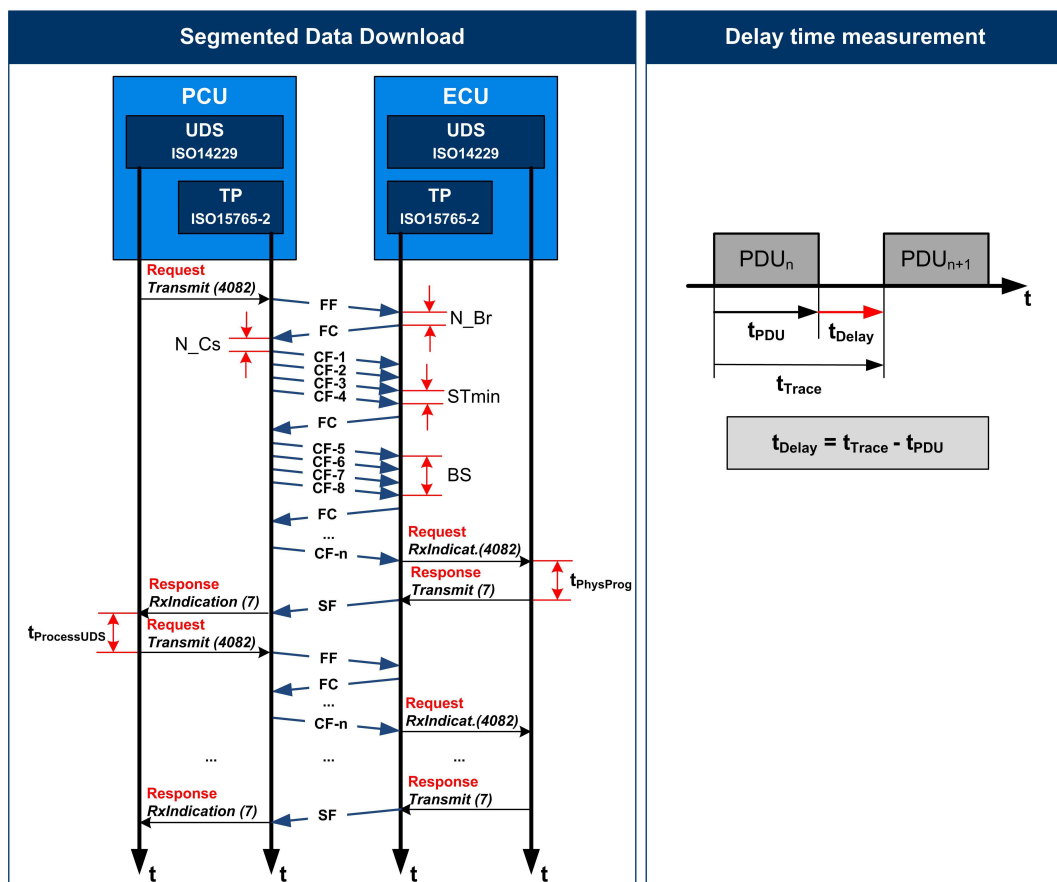


Figure 10.1-3: Segmented data transfer according to ISO 15765-2

The model depends on the configuration parameters as listed in table 10.1-5 and the system runtime parameters as listed below:

- PCU's processing time N_Cs (refer to section 4.1.2.4)
- ECU flashloader's processing time N_Br (refer to section 4.1.2.4)

- c) The ECU flashloader's physical programming time t_{PhysProg} ()
- d) The PCU's processing time $t_{\text{processUDS}}$ ()

To evaluate the model the system related processing times N_Cs , N_Br , t_{PhysProg} and $t_{\text{processUDS}}$ have to be measured. These processing times are independent of the underlying bus system.

Table 10.1-5: PCU and ECU communication configuration parameters

Parameter	Value	Comment
CAN Baud rate	125 ; 250 ; 500 ; 1,000 kBit/s	configured
CAN Address Mode	11 bit – normal addressed	configured
Block size (BS)	32	configured
STmin	0 ms	configured
$d_{\text{SDU_ISO15765-2}}$	4,082 Byte	configured
n_{PDUs}	603	refer to table 10.1-2
$n_{\text{CF_PDUs}}$	583	refer to table 10.1-2

10.1.2.1 Measurement results

Table 10.1-6 shows the measurement results for the test system.

Table 10.1-6: Measurement results of PCU and ECU processing parameters

Line	Parameters	CAN baud rate				unit
		125	250	500	1,000	kBit/s
1	N_Cs	2.9	2.9	2.7	2.6	ms
2	N_Br	0.003	0.003	0.003	0.003	ms
3	STmin	0.014	0.024	0.030	0.038	ms
4	t_{PhysProg}	21.41	22.13	22.34	22.61	ms
5	$t_{\text{processUDS}}$	5.36	5.27	5.94	5.55	ms
6	Download Time	77.7	44.1	27.5	19.3	s
7	Transferred data $d_{\text{SDU_ISO14229}}$	458,720	458,720	458,720	458,720	Byte
8	Performance	5.77	10.17	16.27	23.23	kByte/s
		46.146	81.354	130.145	185.862	kBit/s

The measurement shows a nearly stable parameter set for N_Cs (2.77ms) and N_Br (0.003ms). The transport layer is configured to a $ST_{\text{min}} = 0$. Nevertheless, the ECU requires processing time between two *ConsecutiveFrame (CF)* PDUs. The programming time t_{PhysProg} of this microcontroller is approximately 22.12 ms.

The PCU requires a processing time $t_{\text{processUDS}}$ of 5.53 ms between a received response from the ECU and the transmission of the next request.

10.1.2.2 Evaluation

In chapter 4 the formula 4.1-21 was developed to calculate the net data rate for data transfer via CAN on ISO 15765-2 based on UDS (ISO 14229).

$$t_{DL} = \left\lfloor \frac{d_{PDU_ISO14229}}{d_{SDU_Iso15765-2}} \right\rfloor \cdot \left(\begin{array}{l} \left(n_{PDU(x)} \cdot d_{PDU_ISO11898} \cdot t_{bit} \right) + \\ \left(n_{CF_PDU(x)} - \left\lfloor \frac{n_{CF_PDU(x)}}{BS} - 1 \right\rfloor - 1 \right) \cdot t_{STmin} + \\ \left\lfloor \frac{n_{CF_PDU(x)}}{BS} \right\rfloor (t_{N_Br} + t_{N_Cs}) + \\ t_{PhysProg(x)} + \\ (1 \cdot d_{PDU_ISO11898} \cdot t_{bit}) + \\ t_{PCU_Process} \end{array} \right) + \left(\begin{array}{l} \left(n_{PDU(n)} \cdot d_{PDU_ISO11898} \cdot t_{bit} \right) + \\ \left(n_{CF_PDU(n)} - \left\lfloor \frac{n_{CF_PDU(n)}}{BS} - 1 \right\rfloor - 1 \right) \cdot t_{STmin} + \\ \left\lfloor \frac{n_{CF_PDU(n)}}{BS} \right\rfloor (t_{N_Br} + t_{N_Cs}) + \\ t_{PhysProg(n)} + \\ (1 \cdot d_{PDU_ISO11898} \cdot t_{bit}) \end{array} \right)$$

The number of repetitions n is calculated by:

$$n = \left\lfloor \frac{d_{PDU_ISO14229}}{d_{SDU_Iso15765-2}} \right\rfloor = \left\lfloor \frac{458,720}{4,082} \right\rfloor = 112 \quad (70_{hex})$$

Hence, 112 UDS requests are transmitted with a SDU size of 4,082 bytes. The last UDS request has a SDU size of 1,760 byte and is calculated by formula 4.1-20:

$$\begin{aligned} d_{SDU(n)} &= d_{PDU_ISO14229} \bmod d_{SDU_Iso15765-2} \\ d_{SDU(n)} &= 458,720 \bmod 4,082 = 1,760 \end{aligned}$$

To transmit this data size, 260 PDUs ($n_{PDU(n)}$) are necessary (1 *First Frame* PDU, 251 *Consecutive Frame* PDUs ($n_{CF_PDU(n)}$) and 8 *Flow Control* PDUs).

The integration of all configured of measured parameters results in the formula below:

$$t_{DL} = 112 \cdot \left(\begin{array}{l} (603 \cdot d_{PDU_ISO11898} \cdot t_{bit}) \\ + 566 \cdot t_{STmin} \\ + 19 \cdot (2.773 \cdot 10^{-3} s) \\ + 22.12 \cdot 10^{-3} s \\ + (1 \cdot d_{PDU_ISO11898} \cdot t_{bit}) \\ + 5.53 \cdot 10^{-3} s \end{array} \right) + \left(\begin{array}{l} (260 \cdot d_{PDU_ISO11898} \cdot t_{bit}) \\ + 245 \cdot t_{STmin} \\ + 8 \cdot (2.773 \cdot 10^{-3} s) \\ + 22.12 \cdot 10^{-3} s \\ + (1 \cdot d_{PDU_ISO11898} \cdot t_{bit}) \end{array} \right)$$

The data transfer rate is calculated by:

$$f_{Data_net} = \frac{d_{SDU_ISO14229}}{112 \cdot \left(\begin{array}{l} (603 \cdot d_{PDU_ISO11898} \cdot t_{bit}) \\ + 566 \cdot t_{STmin} \\ + 19 \cdot (2.773 \cdot 10^{-3} s) \\ + 22.12 \cdot 10^{-3} s \\ + (1 \cdot d_{PDU_ISO11898} \cdot t_{bit}) \\ + 5.53 \cdot 10^{-3} s \end{array} \right) + \left(\begin{array}{l} (260 \cdot d_{PDU_ISO11898} \cdot t_{bit}) \\ + 245 \cdot t_{STmin} \\ + 8 \cdot (2.773 \cdot 10^{-3} s) \\ + 22.12 \cdot 10^{-3} s \\ + (1 \cdot d_{PDU_ISO11898} \cdot t_{bit}) \end{array} \right)}$$

Table 10.1-7: Communication model's calculation of data transfer rate

Parameters	CAN baud rate				unit
	125	250	500	1,000	kBit/s
$d_{PDU_ISO11898}$	123	123	123	123	bit
Transferred data $d_{SDU_ISO14229}$	458,720	458,720	458,720	458,720	Byte
Performance	5.84	10.19	16.23	22.64	kByte/s
	46.699	81.543	129.811	181.134	kBit/s

Table 10.1-7 depicts the calculated data transfer rate results of the communication model based on formula 4.1-21. The error between the measured and calculated data transfer rates is in a range of -1.2% to + 2.6%.

10.1.2.3 Conclusion

Formula 4.1-21 works. Compared to the study in section 10.1.1, the additional delays for the physical data reprogramming, the response transmission and the PCU processing time results in a reduction of the maximum data transfer rate for a segmented data transfer.

10.1.3 CAN bus baud rate optimisation

Study's aim

The aim of this study is the validation of the approach to increase the software reprogramming performance for an embedded system by increasing physical layer's baud rate. The theoretical background was discussed in section 4.1.

Experimental Setup

The system was configured as listed in table 10.1-8.

Table 10.1-8: PCU and ECU communication configuration parameters

Parameter	Value	Comment
CAN Baud rate	125 ; 250 ; 500 ; 1,000 kBit/s	configured
CAN Address Mode	11 bit – normal addressed	configured
Block size (BS)	32	configured
STmin	0 ms	configured
d _{SDU_ISO14299}	458,720 Byte	configured

10.1.3.1 Measurement results

Figure 10.1-4 depicts the measurement results for the software reprogramming data transfer between the PCU and the ECU. Table 10.1-9 depicts the details.

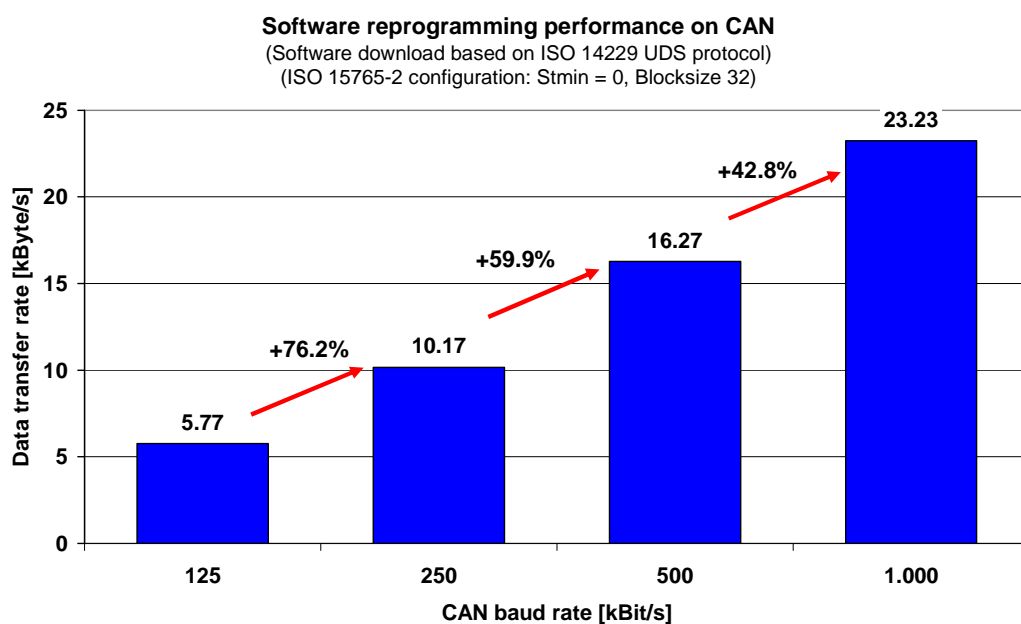


Figure 10.1-4: Software reprogramming performance on CAN

Table 10.1-9: Software reprogramming performance on CAN

Parameters	CAN baud rate				unit
	125	250	500	1,000	kBit/s
Transferred data $d_{SDU_ISO14229}$	458,720	458,720	458,720	458,720	Byte
Download Time	77.7	44.1	27.5	19.3	s
Performance	5.77	10.17	16.27	23.23	kByte/s
	46.15	81.35	130.15	185.86	kBit/s
	346.09	610.16	976.09	1,393.97	kByte/min

10.1.3.2 Evaluation

The critical evaluation of the measurements provides 3 main results:

- 1) Increasing bit rate results in increasing programming performance.
- 2) Doubling bit rate does not result in doubling programming performance.
- 3) The theoretically calculated maximum data transfer rates are not achieved.

As discussed in the previous studies the system's processing delays have an impact to the overall performance. The delays are constant and usually independent of the bus systems bandwidth. In relation to the pure data transfer time the delays' impact is increasing the shorter the pure data transfer time is. In the study this effect is visible when doubling baud rate (e.g. 500 kBit/s to 1,000 kBit/s) results only in 42.8% performance increase.

10.1.3.3 Conclusion

System Design Requirements

The system's delay times have to be analysed before a decision to network optimisation by increasing CAN baud rate is done. As depicted in this and the previous studies within this thesis, a bandwidth optimisation will result in a higher data transfer rate but not automatically in a satisfactory system reprogramming performance. Without knowledge of the system delays a bandwidth optimisation is not advisable.

10.1.4 ISO 15765-2 (CAN TP) Flow Control parameter Block size

Study's aim

In section 4.1.2.1 the impact of the ISO 15765-2 *Flow Control* PDU's parameter *BlockSize* was analysed for a data transfer of only 4,095 bytes. The aim of this study is the validation of the research results even if the data transfer size is increased to 45,870 bytes.

Background

The flow control parameter *Block Size (BS)* defines the number of *Consecutive Frame (CF)* PDUs that can be received by a receiver node in one block within a segmented data transmission. After reception of that block a *Flow Control (FC)* PDU has to be sent by the initial receiver node to signal the current flow state and to continue data transfer.

Experimental Setup

The system was configured as listed in table 10.1-10.

Table 10.1-10: PCU and ECU communication configuration parameters

Parameter	Value	Comment
CAN Baud rate	125 ; 250 ; 500 ; 1,000 kBit/s	configured
CAN Address Mode	11 bit – normal addressed	configured
Block size (BS)	{0, 1, 8, 16, 32}	configured
STmin	0 ms	configured

The *BlockSize (BS)* parameter of the flashloader's *FlowControl* PDU varied according to the values as listed above.

10.1.4.1 Measurement results

Table 10.1-11 depicts the measurement results (average of 5 independent measurements) for each flashloader *BlockSize* configuration. Figure 10.1-5 depicts the corresponding graphical evaluation.

Table 10.1-11: Flow Control parameter “BS” measurement results

BS	Flashsize	Flashtime	Data transfer rate		Relation
0 CF-Frames	458,720 Byte	23.16 s	19.34 kByte/s	1,161 kByte/min	100.00 %
1 CF-Frames	458,720 Byte	192.83 s	2.32 kByte/s	139 kByte/min	12.01 %
8 CF-Frames	458,720 Byte	48.51 s	9.24 kByte/s	554 kByte/min	47.74 %
16 CF-Frames	458,720 Byte	35.03 s	12.79 kByte/s	767 kByte/min	66.12 %
32 CF-Frames	458,720 Byte	28.43 s	15.75 kByte/s	945 kByte/min	81.45 %

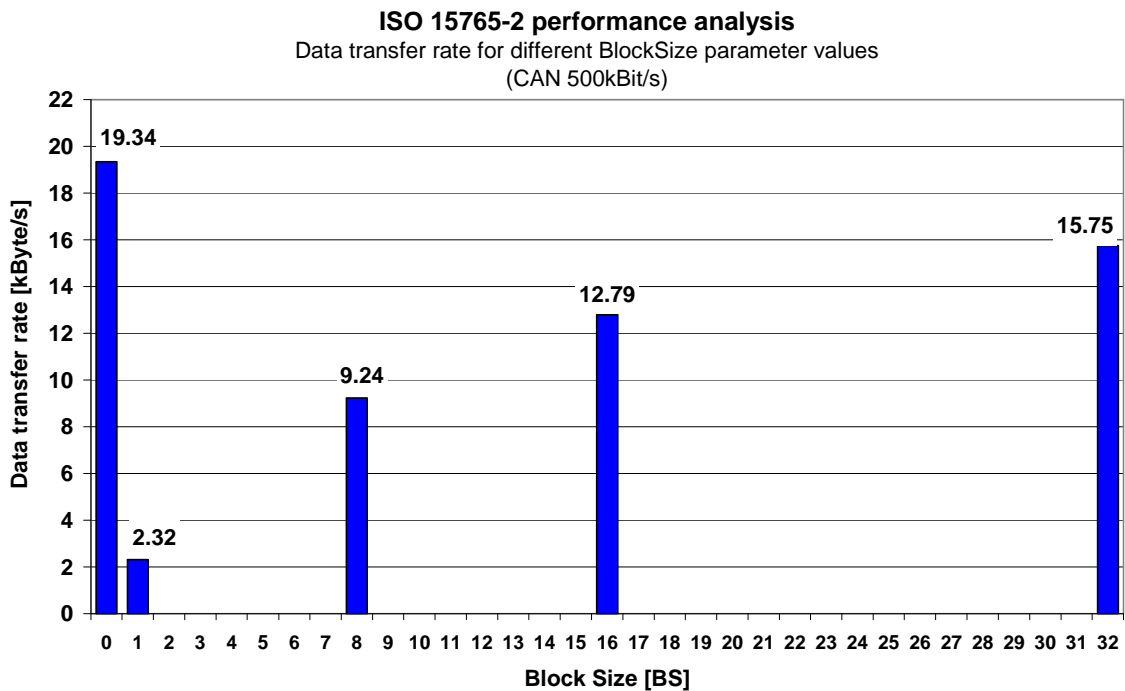


Figure 10.1-5: Impact of Flow Control parameter BS

10.1.4.2 Evaluation

The measurement result has an equal tendency and figure 10.1-5 has an equal characteristic as figure 4.1-4. As discussed in section 4.1.2.1 the parameter *BlockSize* (*BS*) has a significant impact on the data transfer rate.

A block size equal to 0 provides best results because after the first *FlowControl* PDU no additional *FlowControl* PDUs are required. Hence, neither additional PDU runtimes and processing times within the ECU nor additional PCU processing times occur.

A *Blocksize* equal to 1 provides worse results because after each received data PDU the ECU has to send a flow control PDU. This results in a maximum possible additional PDUs processing time in the PCU and ECU and additional PDU runtimes for *FlowControl* PDUs.

Compared to the theoretically calculated values based on an ideal system, without any system delays, in table 4.1-4 (note: transmission of only 4,095 byte) the measured performance relation between the configuration of $BS = 0$ and other *BS* configurations is higher. This is because of the additional system delays N_{Cs} , N_{Br} , $t_{PhysProg}$ and $t_{Process_UDS}$ which reduce the data transfer rate even if only a few *FlowControl* PDUs are necessary.

10.1.4.3 Conclusion

The trend of the curve in figure 4.1-4 (section 4.1.2.1) is correct. The impact of the configured parameter *BlockSize* is given.

System Design Requirements

With focus on data transfer rate the ISO 15765-2 *FlowControl* parameter *BlockSize* shall be configured equal to zero. On the other hand, this requires buffer to receive the complete SDU without further *FlowControl* communication to the sender node. If not enough buffer is available, the *BlockSize* has to be configured to a value that supports the maximum possible buffer size. Nevertheless, each reduction of the *Blocksize* parameter configuration (when unequal to zero) will reduce the data transfer rate.

10.1.5 ISO 15765-2 (CAN TP) FlowControl parameter *STmin*

Study's aim

The aim of this study is the validation of the research results of section 4.1.2.2 where the impact of the ISO 15765-2 *Flow Control* PDU's parameter of the minimum separation time *STmin* was theoretically analysed for a data transfer on 4,095 bytes. The study shall depict that the system behaviour is equal by trend, even if the data transfer size is increased to 45,870 bytes.

Experimental Setup

The system was configured as listed in table 10.1-12.

Table 10.1-12: PCU and ECU communication configuration parameters

Parameter	Value	Comment
CAN Baud rate	125 ; 250 ; 500 ; 1,000 kBit/s	Configured
CAN Address Mode	11 bit – normal addressed	Configured
Block size (BS)	32	Configured
<i>STmin</i>	{0, 1, 5} ms	Configured

The *STmin* parameter of the flashloader *FlowControl* PDU varied according to the values as listed above.

10.1.5.1 Measurement results

Table 10.1-13 depicts the measurement results (average of 5 independent measurements) for each flashloader *BlockSize* configuration. Figure 10.1-6 depicts the corresponding graphical evaluation.

Table 10.1-13: Flow Control parameter “STmin” measurement results

	STmin [ms]			unit
	0	1	5	
Transferred data size	408,720	408,720	408,720	Byte
Total programming time	28.43	73.58	342.61	s
Data transfer rate	15.8	6.1	1.3	kByte/s
	945.3	365.3	78.5	kByte/min

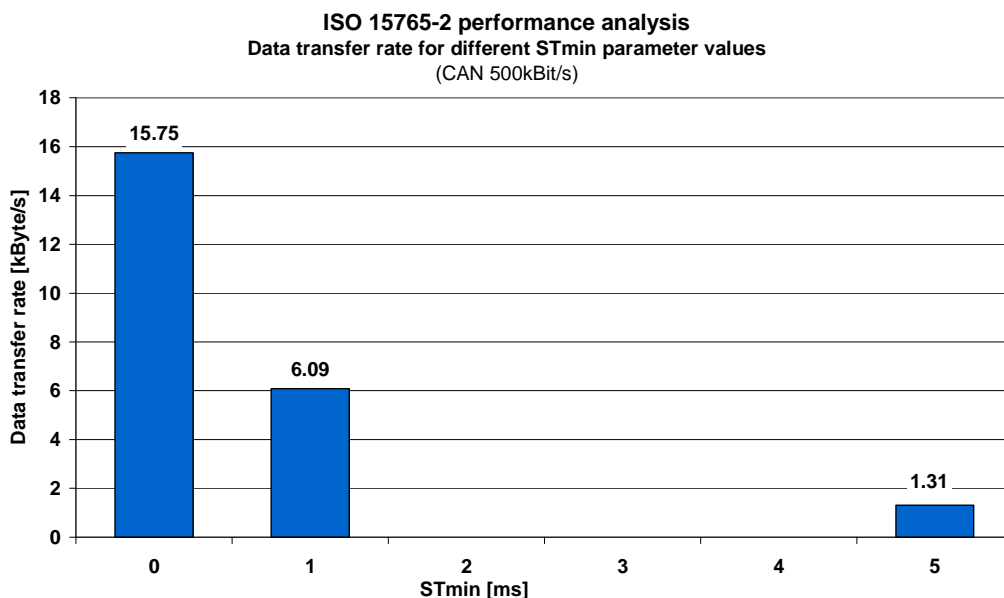


Figure 10.1-6: Impact of Flow Control parameter STmin

10.1.5.2 Evaluation

The measurement result has an equal tendency and figure 10.1-6 has an equal characteristic as figure 4.1-6. As discussed in section 4.1.2.2 the parameter *STmin* has a significant impact to the data transfer rate.

A separation time *STmin* equal to zero provides best results. However, the measured maximum data transfer rate value is not as high as in the theoretical discussion in section 4.1.2.2. This is because of the additional system delays N_{Cs} , N_{Br} , $t_{PhysProg}$ and $t_{Process_UDS}$ which reduce the data transfer rate even if no separation time by protocol is configured (refer to section 10.1.3).

10.1.5.3 Conclusion

The trend of the curve in figure 4.1-6 (section 4.1.2.2) is correct. The impact of the configured parameter *STmin* is given. The conclusion is that each separation time during reprogramming communication shall be avoided. Even the smallest gap between the PDUs provides a large delay time if this time is summarised during a long data transfer period (depends on the size of total transferred data). This is common to all communication systems event triggered, as well as time triggered systems. However, for time triggered systems (e.g. FlexRay) the delay is mainly given by the global schedule.

System Design Requirements

With focus on data transfer rate the ISO 15765-2 *FlowControl* parameter *STmin* shall be configured equal to zero. This is the only possibility to ensure that only the system specific delays are involved for data transfer and protocol handling. On embedded system's side it has to be ensured, that a CAN controller is able to process a received PDU within the available time period before the next consecutive PDU will receive. A hardware interrupt based data reception concept as discussed in section 6.2 will be a possible solution.

10.2 Application Protocol ISO 14229 (UDS) Optimisation

As theoretically analysed in chapter 3 and chapter 4 there are several approaches to accelerate the data transfer between a PCU and an ECU. This test series has the focus on the three approaches based on application layer implementation:

- a) Double buffered data transfer
- b) Data compression
- c) Combination of double buffering and data compression

Study's aim

The aim of this study is the evaluation of the theoretically discussed performance increase for the different approaches. The study shall also proof that only the combination of both approaches will provide best results.

Experimental Setup

The system was configured as listed in table 10.2-1.

Table 10.2-1: Flow Control parameter “STmin” measurement results

Parameter	Value	Comment
CAN Configuration		
CAN Baud rate	125 ; 250 ; 500 ; 1,000 kBit/s	configured
CAN Address Mode	11 bit – normal addressed	configured
ISO 15765-2 Configuration		
Block size (BS)	32	configured
STmin	0 ms	configured
SDU size	4,082 Byte (\$FF2)	configured
Compression algorithm		
Uncompressed data size	458,720 Byte (447 kByte)	
Compressed data size	328,730 Byte (321 kByte)	
Compression ratio	71.66% (-28.34%)	

10.2.1.1 Measurement results

Measurement results overview

Table 10.2-2 depicts the measurement results of different data transfer acceleration scenarios.

Table 10.2-2: Measurement results of different data transfer acceleration scenarios

Scenario	Data rate			
	CAN 125 kBit/s	CAN 250 kBit/s	CAN 500 kBit/s	CAN 1000 kBit/s
Basic	5.70 kByte/s	10.09 kByte/s	15.65 kByte/s	17.80 kByte/s
	100 %	100 %	100 %	100 %
DoubleBuffer	5.99 kByte/s	10.37 kByte/s	16.25 kByte/s	18.46 kByte/s
	105 %	103 %	104 %	104 %
Compression	7.41 kByte/s	12.58 kByte/s	18.26 kByte/s	20.61 kByte/s
	130 %	125 %	117 %	116 %
DoubleBuffer+Compression	8.35 kByte/s	14.32 kByte/s	21.64 kByte/s	24.79 kByte/s
	146 %	142 %	133 %	134 %

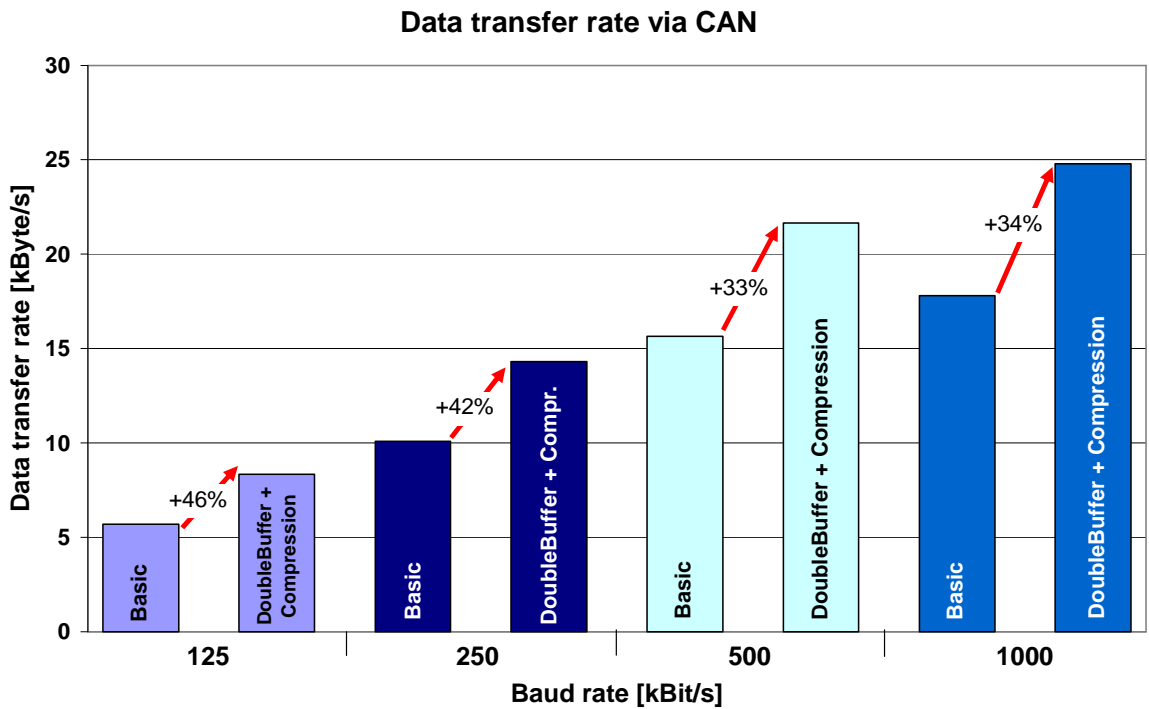


Figure 10.2-1: Measurement results – best case relation for all scenarios

Contributions of the different parameters

Figure 10.2-2 depicts the data transfer rate for the different scenarios in correlation to the contributions of the different influencing parameters for all measured scenarios.

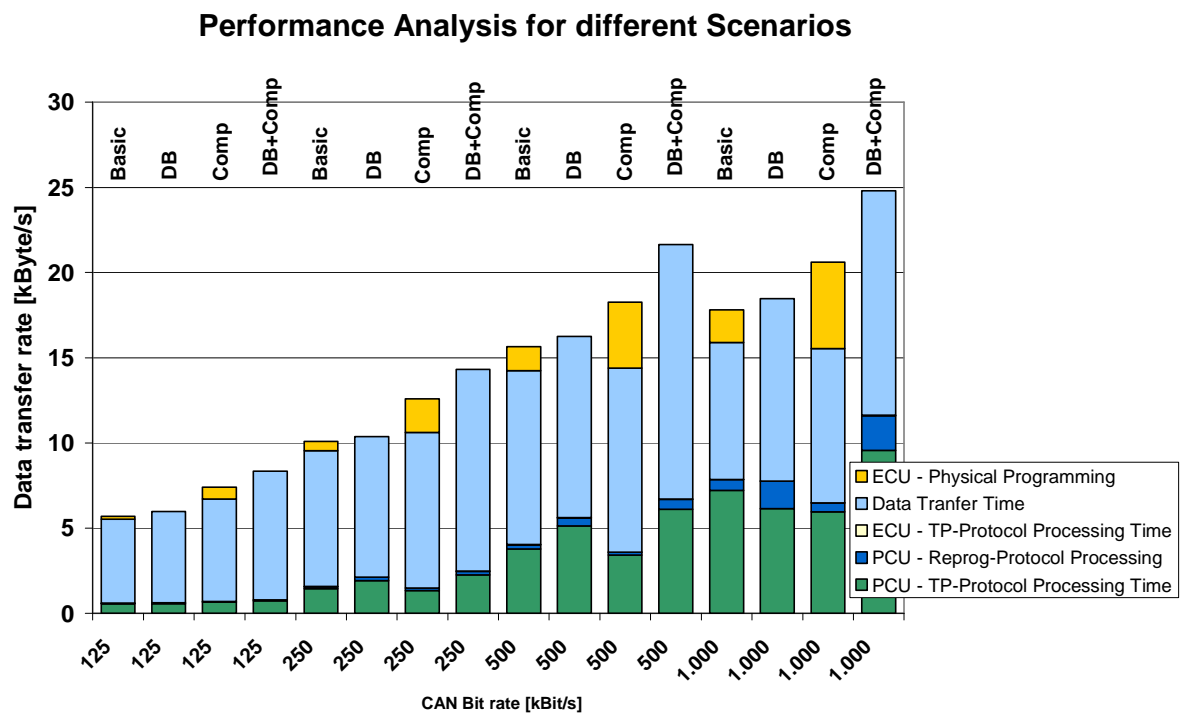


Figure 10.2-2: Contribution of the different parameters

Table 10.2-3 depicts the measured parameter values.

Table 10.2-3: Measurement results – detailed analysis

Line	Scenario	Parameters	CAN baud rate				unit	
			125	250	500	1,000		
		Transferred data $d_{SDU_ISO14229}$	458,720	458,720	458,720	458,720	Byte	
1	Basic	Download Time	78.6	44.4	28.6	25.2	s	
2		Performance		5.70	10.09	15.65	17.80	kByte/s
3				45.60	80.74	125.18	142.38	kBit/s
4				342.00	605.58	938.88	1,067.87	kByte/min
5		Number of SDU _(4080Byte)	113	113	113	113		
6		$t_{Download(4080Byte)}$	704.55	412.04	247.93	212.38	ms	
7		N_Br	0.03	0.02	0.01	0.01	ms	
8		N_Cs	3.62	3.12	3.16	4.53	ms	
9		t_{Prog}	21.80	22.25	22.36	22.79	ms	
10		t_{PCU}	4.18	4.70	3.82	7.61	ms	
11	DoubleBuffer	Download Time	74.8	43.2	27.6	24.3	s	
12		Performance		5.99	10.37	16.25	18.46	kByte/s
13				47.89	82.99	129.99	147.72	kBit/s
14				359.20	622.44	974.98	1,107.87	kByte/min
15		Number of SDU _(4080Byte)	113	113	113	113		
16		$t_{Download(4080Byte)}$	688.84	393.70	242.24	197.33	ms	
17		N_Br	0.03	0.02	0.01	0.01	ms	
18		N_Cs	3.42	3.81	4.03	3.46	ms	
19		t_{Prog}	0.03	0.02	0.02	0.02	ms	
20		t_{PCU}	7.71	8.03	6.89	17.15	ms	
21	Compression	Download Time	60.5	35.6	24.5	21.7	s	
22		Performance		7.41	12.58	18.26	20.61	kByte/s
23				59.28	100.62	146.05	164.88	kBit/s
24				444.60	754.63	1,095.36	1,236.59	kByte/min
25		Number of SDU _(4080Byte)	81	81	81	81		
26		$t_{Download(4080Byte)}$	777.99	470.47	345.24	298.21	ms	
27		N_Br	0.03	0.02	0.01	0.01	ms	
28		N_Cs	3.68	2.64	3.41	4.54	ms	
29		t_{Prog}	72.91	73.12	72.91	73.22	ms	
30		t_{PCU}	2.83	5.11	2.95	7.41	ms	
31	DoubleBuffer + Compression	Download Time	53.7	31.3	20.7	18.1	s	
32		Performance		8.35	14.32	21.64	24.79	kByte/s
33				66.79	114.52	173.13	198.35	kBit/s
34				500.91	858.93	1,298.49	1,487.60	kByte/min
35		Number of SDU _(4080Byte)	81	81	81	81		
36		$t_{Download(4080Byte)}$	649.68	366.32	242.62	198.99	ms	
37		N_Br	0.04	0.02	0.02	0.01	ms	
38		N_Cs	3.04	3.04	3.60	4.04	ms	
39		t_{Prog}	0.03	0.02	0.02	0.02	ms	
40		t_{PCU}	4.20	5.35	6.54	16.32	ms	

10.2.1.2 Evaluation

Figure 10.2-3 depicts a principle, abstract and not in a time content view of the different test scenario results.

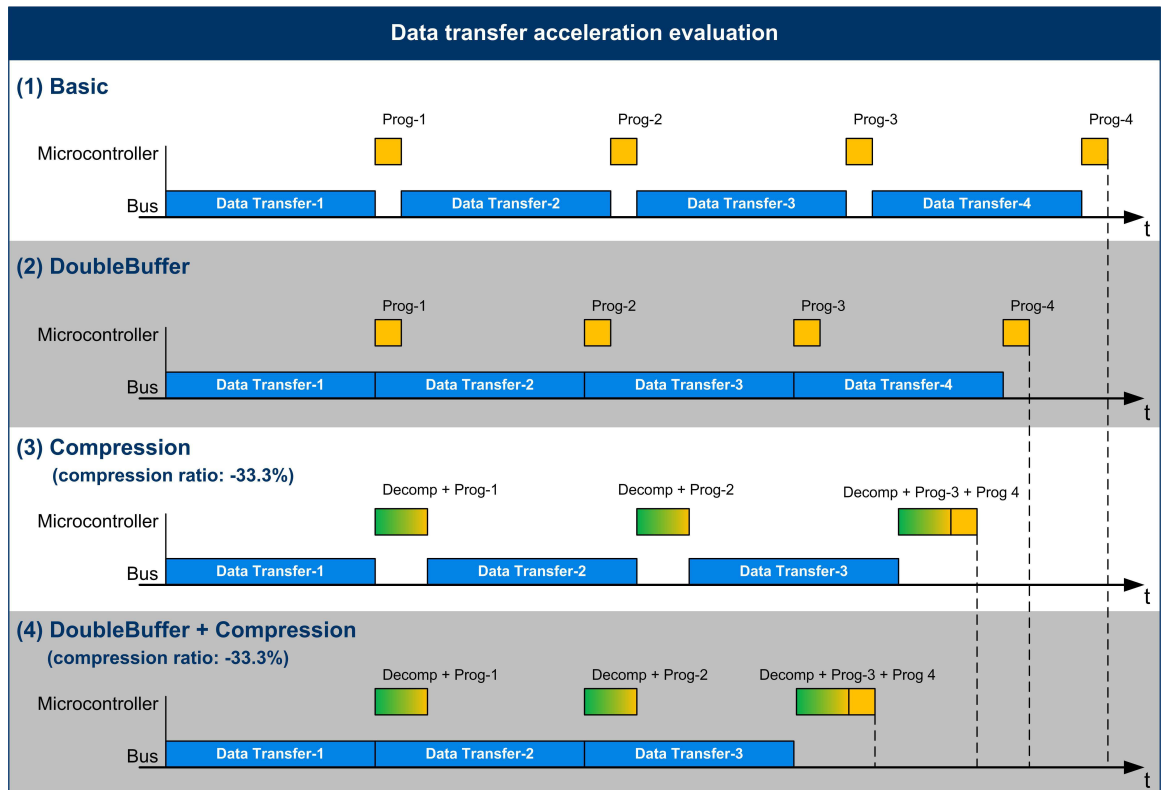


Figure 10.2-3: Test result timing evaluation

Double buffered data transfer (only)

Because of microcontroller's high performance physical programming ($\approx 180\text{kByte/s}$) the double buffered data transfer as the only optimisation method provides only a small benefit. The physical programming time for 4,080 Byte is only 22.3 ms. Compared to the corresponding data transfer time of $\text{CAN}_{125\text{kBit/s}}$ and $\text{CAN}_{250\text{kBit/s}}$, the relation between t_{Transfer} and T_{Prog} is too small (refer to figure 3.2-4). For $\text{CAN}_{500\text{kBit/s}}$ and $\text{CAN}_{1000\text{kBit/s}}$ a higher increase of the data transfer rate is expected because of a better relation of t_{Transfer} and t_{Prog} . But the detailed analysis results of table 10.2-3 and figure 10.2-2 depict also that the PCU's processing time to handle the reprogramming protocol UDS is increased. Hence, the benefit of processing data reception and physical programming in parallel will be reduced.

Data compression (only)

As described in chapter 5 the data compression approach requires additional time to decompress the data before physical programming. This time can be calculated by the values of scenario "Basic" (data transfer without compression) and scenario "Compression" (data transfer with compression):

$$t_{\text{Decompression}} = t_{\text{PhysProg_WithoutComp}} - t_{\text{PhysProg_WithComp}}$$

Table 10.2-4: Decompression routine runtime

Parameter	CAN bit Rate				Unit
	125	250	500	1000	
$t_{\text{Prog_uncompressed}}$ (Scenario "Basic")	21.41	22.13	22.34	22.61	ms
$t_{\text{Prog_compressed}}$ (Scenario "Compression")	73.01	73.05	72.99	73.24	ms
$t_{\text{Decompression}}$	51.60	50.92	50.65	50.63	ms

The data compression to 71% reduce the number of 4,080 byte blocks from 113 to 81 (refer to table 10.2-3 – line 5, 15, 25, 35). But the additional delay time for decompression (visible in figure 10.2-2) of approx. 50.95 ms reduce the benefit of compressed data transfer. Nevertheless, for low speed bus systems the PDU reduction to transmit all data and therefore the reduction of transmission time is significantly higher than the additional decompression time. For high speed bus systems the compression effect will be smaller because the relation of data transfer time reduction and additional compression time is smaller.

Combination of double buffered data transfer and data compression

The combination of doubled buffered data transfer and compressed data transfer provides best performance because the disadvantages of both methods are compensated. The additional time for decompression and the time for physical reprogramming are not visible because of the parallel processed data reception. Only the pure data transfer time is visible. The data compression results in a less number of PDUs necessary to transmit all data. The benefit is the higher the slower the bus system is because each saved PDU saves runtime and therefore reduce total data transfer time.

The increasing PCU time is noticeable. A guess is that the ECU's immediate response and the necessity to compress the data on PCU side require additional time which delays the data transfer. But those effects in the PCU implementation are not part of this work.

10.2.1.3 Conclusion

System Design Requirements

The theoretically discussed approaches for data transfer acceleration works in principle. The case study demonstrates this in detail. The impact of the relation of t_{Transfer} and t_{Prog} is given as discussed in chapter 3.

On the other hand the impact of the PCU is given. The higher the bandwidth the higher the performance reduction that is possible by the PCU because of additional delays during the protocol handling.

Double buffered data transfer and therefore the possibility to do activities in parallel is the base for reprogramming acceleration. At least reprogramming and several other things like data de-compression etc. could be done during ongoing data reception. Next steps should be to do additional things in parallel. Signature calculation or CRC calculation, for example, can be done in parallel, too. But this was not in focus of this work.

10.3 Gateway optimisation

This study is intended to verify the theoretically discussed impacts of a gateway (refer to section 7.3) by a real implementation.

Experimental Setup

The gateway based on the AUTOSAR layered software architecture and implements a CAN communication stack according to the protocols ISO 11898 and ISO 15765-2. The gateway was implemented within a prototype project for a V850 Fx3 and for a V850 Fx4 microcontroller. The implementation was evaluated with the PCU Monaco® developed by the Softing AG⁷⁷. The PCU communicates on CAN₁. The gateway is processing the transfer from CAN₁ to CAN₂ and vice versa. The tracing tool documents the bus communication traffic on both bus systems.

Figure 10.3-1 depicts the evaluation test system layout.

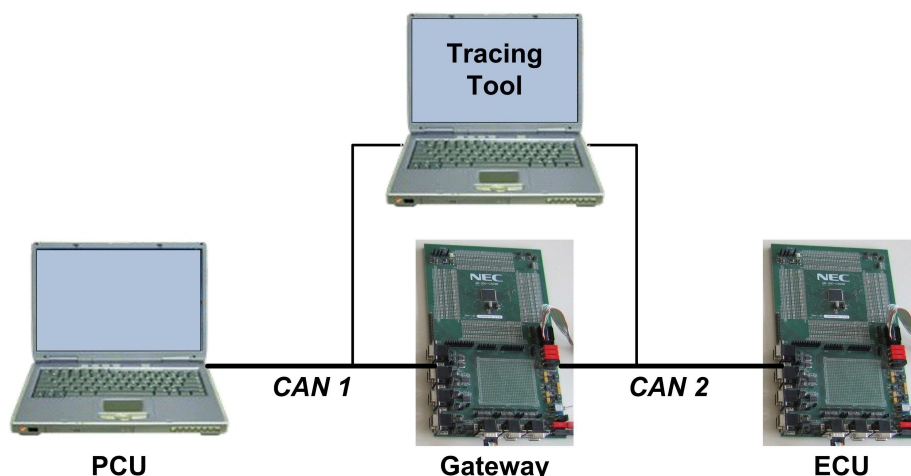


Figure 10.3-1: Gateway test system overview

⁷⁷ Softing AG, Germany [Softing]

PCU:	DTS Monaco (Softing)
Gateway Microcontroller:	V850 Fx3 and V850 Fx4
ECU Microcontroller:	D70F 3461 6J(A1) (“CargateM”)
Tracing tool:	CANoe (Vector)

10.3.1 Buffer for the partly store and forward routing strategy

Study’s aim

The aim of this study is to evaluate the impact of the number of buffers for gateway’s routing performance if a *partly store and forward routing strategy* is used (refer to section 7.3).

10.3.1.1 Measurement results

Figure 10.3-2 depicts the measurement results of the data transfer rate for different buffer configurations.

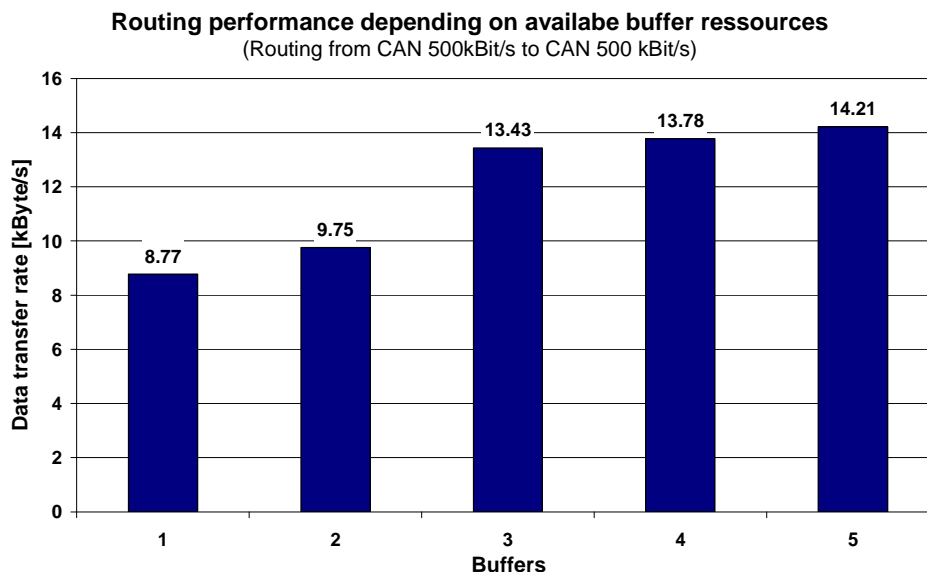


Figure 10.3-2: Data transfer rate for different buffer scenarios

10.3.1.2 Evaluation

As discussed in section 7.3.2 the *partly store and forward routing strategy* is a good compromise between buffer demand (resources) and routing performance (data transfer rate). In figure 7.3-3 the necessity of more than one buffer is illustrated to accelerate the gateway’s routing performance and the corresponding data transfer rate.

The study shows that if only one buffer is configured, this buffer is alternatively the target buffer during data reception on CAN₁ or source buffer during data transmission on CAN₂. If no buffer is available, the data transfer on the corresponding side is delayed and the total data transfer rate is decreased (8.77 kByte/s). In that case the

Table 10.3-1: Trace – data transfer for gateway with 2 buffer resources

+-----+ Transfer on CAN 1 +-----+				+-----+ Transfer on CAN 2 +-----+			
Time	Bus	ID	Data	ID	Data		
[..]							

Download to Flash							

31.076538	1	640	1F F2 36 01 04 06 00 01				
31.076827	2			640	1F F2 36 01 04 06 00 01		
31.076840	1	5C0	30 20 00 00 6E 31 01 00				
31.077087	2			5C0	30 20 00 FF FF FF FF FF		
31.077836	1	640	21 02 03 04 05 06 07 08				
[..]							
31.087283	1	640	20 00 80 07 00 00 80 07				
31.087557	1	5C0	30 20 00 00 6E 31 01 00				
[..]							
31.092758	1	640	2E 07 00 00 80 07 00 00				
31.092963	2			640	21 02 03 04 05 06 07 08		
31.093050	1	640	2F 80 07 00 00 80 07 00				
31.093305	2			640	22 09 0A A0 50 09 22 00		
[..]							
31.098182	1	640	20 00 80 07 00 00 80 07				
31.098299	2			640	20 00 80 07 00 00 80 07		
31.098452	1	5C0	31 20 00 FF 25 A0 21 80				
31.098651	2			640	21 00 00 80 07 00 00 80		
31.099003	2			640	22 07 00 00 80 07 00 00		
[..]							
31.104210	2			5C0	30 20 00 FF FF FF FF FF		
[..]							
31.112257	2			640	27 80 07 00 00 80 07 00		
31.112611	2			640	28 00 80 07 00 00 80 07		
31.112848	1	5C0	30 20 00 04 05 06 07 08				
31.112965	2			640	29 00 00 80 07 00 00 80		
31.113319	2			640	2A 07 00 00 80 07 00 00		
31.113673	2			640	2B 80 07 00 00 80 07 00		
31.113838	1	640	21 00 00 80 07 00 00 80				
31.114027	2			640	2C 00 80 07 00 00 80 07		
31.114128	1	640	22 07 00 00 80 07 00 00				
[..]							
31.438352	2			640	27 00 00 00 00 00 00 00		
31.438618	2			5C0	03 7F 36 78 FF FF FF FF		
31.438895	1	5C0	03 7F 36 78 56 01 00 02				
31.460015	2			5C0	02 76 01 78 FF FF FF FF		
31.460293	1	5C0	02 76 01 78 56 01 00 02				
[..]							

A second buffer increases the routing performance slightly. A significant optimisation of routing performance is visible in case of 3 available buffer resources. As discussed in section 7.3.2 and illustrated in figure 7.3-4, this effect bases on the bandwidth relation of CAN₁ and CAN₂. ($r_{\text{Bandwidth}} \approx 1$). The jitter of both bus systems (data transfer, processing

time etc.) is responsible that CAN₁ will get no free buffer when necessary. In that case the communication on CAN₁ must be delayed by the gateway until the communication fragment on CAN₂ has been finalized and a free buffer is available. Table 10.3-1 depicts a trace of gateways routing process with only 2 available buffers.

The communication is delayed on CAN₁ by a transport layer's *FlowControl* PDU with *FlowState* = *Wait* (\$31) at time stamp 31.098452. After all data of the buffer are transmitted on CAN₂ (visible by a *FlowControl* PDU with *FlowState* = *ClearToSend* (\$30) at time stamp 31.104210) and the buffer is free again a *FlowControl* PDU on CAN₁ is sent with *FlowState* = *ClearToSend* (\$30) at time stamp 31.112848.

In between this time of 5.396 ms no data transfer on CAN₁ is possible. Hence, the total data transfer rate is decreased to 11.01 kByte/s (4,080 Byte / 0.361814s). The software reprogramming performance is decreased, too.

10.3.2 Increasing gateways clock frequency

High performance data routing within a gateway requires, that the routing process is handled in the interrupt modus. That means that a received PDU is immediately processed (e.g. transport layer protocol analysis, payload separation, buffer storage etc.). If interrupt runtimes are too long, the system is not able to handle data reception and data transmission in parallel and typically the data transmission task will be skipped.

An approach to optimise interrupt runtimes or interrupt latencies is to increase the systems clock frequency. Of course, this is only possible within small boundaries. Therefore the gateway was implemented on a V850 Fx4 microcontroller with a clock frequency of 160 MHz.

Table 10.3-2: Routing performance on different microcontrollers

CAN 500kBit/s to CAN 500 kBit/s			
	Microcontroller		unit
	V850 Fx3	V850 Fx4	
Clock frequency	120	160	MHz
Gateway buffers	3	3	buffer
Data Size	458,752	458,752	Byte
Total Reprogramming time	33.36	26.37	s
Routing Performance	13.43	16.99	kByte/s
	805.82	1,019.20	kByte/min

10.3.3 Summary

The theoretically discussed aspects for gateways to couple different bus systems in chapter 7 are valid. The *partly store and forward* routing strategy provides good data transfer rate results, but the effort for implementation is high.

To provide a high data transfer rate more than one buffer resource has to be implemented. The number of buffers depends on the bandwidth relation of the bus systems. If the relation between source and target bus system is nearly 1 the jitter has to be taken into account and additional buffers are required.

An increasing microcontroller's clock frequency will result in a faster interrupt handling and therefore in a faster routing. On the other hand the increasing of clock frequencies provides other disadvantages: system's temperature is increasing by higher clock frequency and other cooling mechanisms (cooling elements) are necessary. Also EMC might be a problem if the clock frequency is increasing. Both topics are critical, especially within the automotive industry.

10.4 Software reprogramming via FlexRay

This sub-chapter is intended to verify the theoretical discussions of chapter 4 a real implementation.

10.4.1 Vehicle access by CAN bus system

The current vehicle networks implement FlexRay only as an in-vehicle bus system. That means that FlexRay is not directly accessible by a PCU. Based on the legislative OBD-II (onboard diagnostics for emission related ECUs based on ISO 15765-4) requirements, that requires CAN as the vehicle access bus system by law for OBD-II communication, CAN is also used for enhanced diagnostic communication. Due to that, software reprogramming as a part of enhanced diagnosis, is processed via CAN.

The case study based on a real vehicle network configuration prove that communication for software reprogramming via FlexRay is currently limited by the vehicle access CAN bus system.

Experimental Setup

The approaches of chapter 4 to accelerate data transfer via FlexRay have been implemented within a prototype project for a V850 Fx4⁷⁸ microcontroller. The flashloader

⁷⁸ Microcontroller: Renesas V850-D70F3461GJ(A1)

implements a FlexRay communication stack according to the protocols FlexRay 2.1, ISO 10681-2 and ISO 14229. The implementation was evaluated with the PCU FlashCedere® developed by the SMART GmbH⁷⁹. The PCU communicates on CAN. The gateway is processing the transfer from CAN to FlexRay and vice versa. The tracing tool documents the bus communication traffic on both bus systems. Figure 10.4-1 depicts the evaluation test system layout.

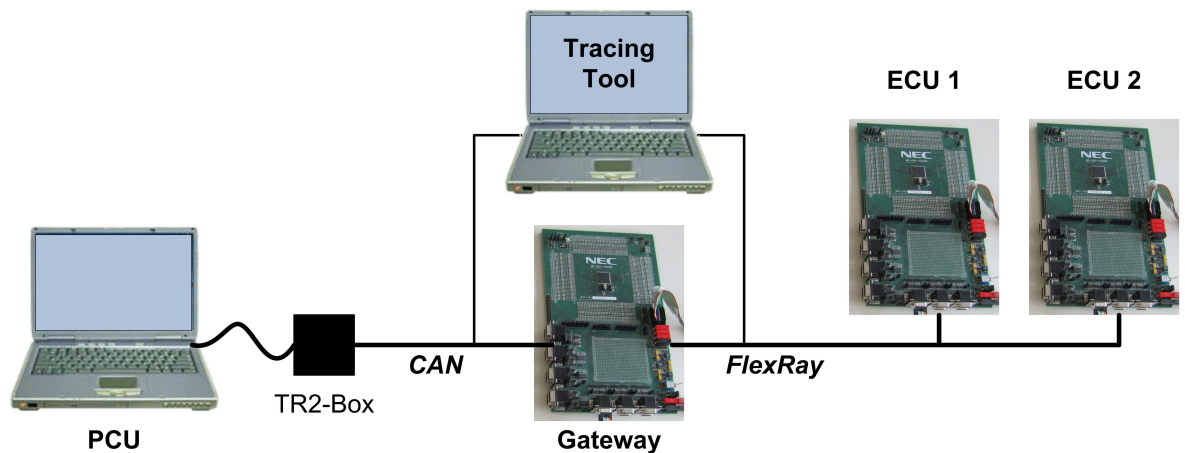


Figure 10.4-1: FlexRay test system overview

(PCU): FlashCedere® V1.20 (8399) and PCCOM: V01.61
 CAN - PCC-TR2Box: S/N 0700/027-1
 Gateway Microcontroller: V850 Fx3
 ECU Microcontroller: V850 Fx4

With focus on software reprogramming within a vehicle the typical communication link within implemented: The PCU communicates on CAN. A gateway is processing the data from CAN to FlexRay and vice versa. The tracing tool documents the bus communication traffic on both bus systems.

CAN bus system

The CAN bus system was configured as listed below:

Bit rate: 500 kBit/s
 ISO15765-2 FlowControl.STmin: 0 ms
 ISO15765-2 FlowControl.Blocksize: 32 PDUs

⁷⁹ SMART Elektronik Development GmbH, Germany [Smart]

FlexRay schedule

The FlexRay communication schedule was configured as listed below:

Base cycle time:	5 ms
Number of gateway PDUs per cycle:	8
Gateway PDU cycle repetition	1
Gateway PDU payload length:	42 Byte
Number of ECU PDUs per cycle:	1
ECU slot PDU repetition:	1 or 4
ECU PDU payload length:	42 Byte

10.4.1.1 Protocol Restrictions

As discussed in section 3.2.2 the data transfer rate on FlexRay depends on several configuration parameters.

FlexRay schedule and FlexRay PDU's payload

A main influencing factor for the data transfer performance is the payload that is transmitted within one communication cycle. In the study 42 byte payload for each FlexRay PDU are configured and the schedule allows transmission of 8 PDUs per cycle. Due to ISO 10681-2 protocol (refer to section 4.2.2) the possible payload for data transfer must be reduced by 8 byte for the *Start Frame's* PCI and 6 byte for the *Consecutive Frame's* PCI. Hence, if only *Consecutive Frame* PDUs are transmitted and all PDUs per cycle are in use, a payload of 288 byte per cycle is possible.

ISO 10681-2 configuration

The FlexRay communication layer protocol ISO 10681-2 defines a data flow controlling (hand shake) between sender and receiver via *Flow Control* PDUs (refer to figure 4.2-10). With focus on data transfer rate optimisation this *Flow Control* PDUs delay the data transmission. Because of FlexRay protocol's exclusive slot allocation for a sender node (refer to section 4.2.1) the cycle repetition of this slot has an impact to the data transfer rate. In case, the cycle repetition is configured to 4, in worst case the sender of the *Flow Control* PDU is allowed to transmit the PDU after 4 communication cycles. During this time no data transfer on that communication link is allowed. Hence, the communication layer must be configured that no additional *Flow Control* PDUs after the initial one are required. This is possible if on sender side the transmission mode of *ConsecutiveFrame_EOB* PDUs is disabled and on sender side enough buffer for the data reception is configured (this results in a Flow Control Bandwidth Control

parameter equal to zero and means no bandwidth control is necessary – refer to table 4.2.5 and [ISO 10681-2_2]).

10.4.1.2 Test results

Figure 10.4-2 depicts measured and calculated data transfer rates for the different network configurations.

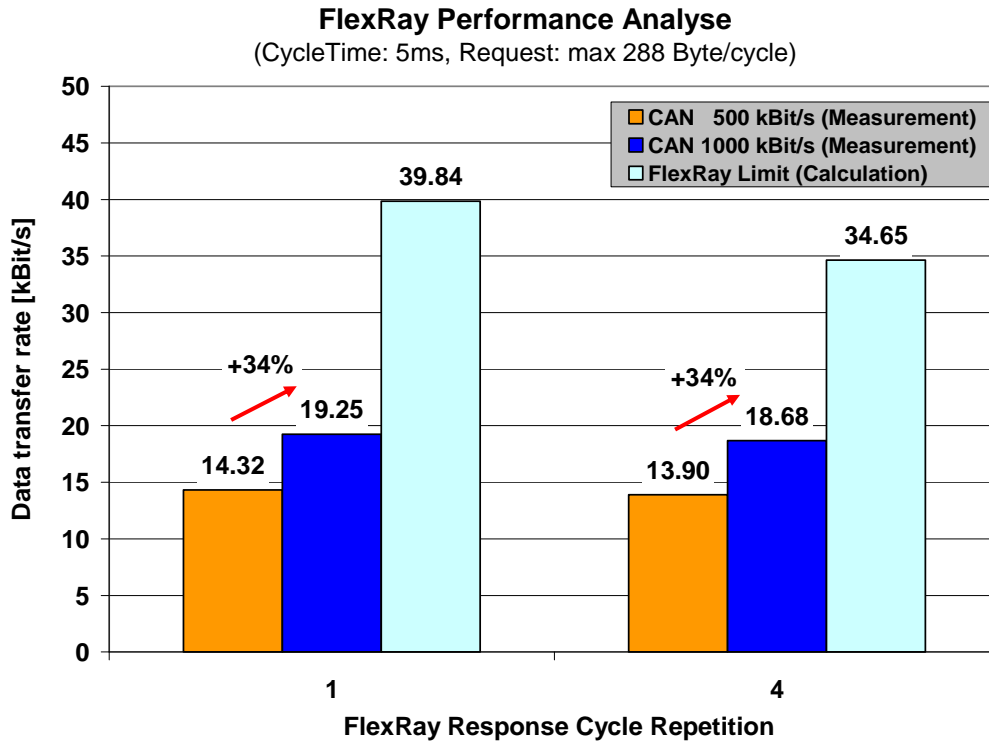


Figure 10.4-2: FlexRay data transfer rate

The data transfer values for CAN_{500kBit/s} and CAN_{1000kBit/s} as the source bus system are measured. To depict the potential of that FlexRay configuration, the data transfer rate for an assumed high speed source bus system (e.g. Ethernet) is depicted, too. Table 10.4-1 depicts the measurement and calculation values.

Table 10.4-1: Data transfer performance based on CAN as vehicle interface bus system

FlexRay	Parameter	CAN		FlexRay (Config-Limit)	unit	
		500	1,000			
Response Cycle Repetition = 1	Transferred Data Size	3,080,192	3,080,192	4,080	Byte	
	Download Time	210.10	156.22	0.10	s	
	Performance		14.32	19.25	39.84	kByte/s
			859.03	1,155.27	2,390.63	kByte/min
Response Cycle Repetition = 4	Transferred Data Size	3,080,192	3,080,192	4,080	Byte	
	Download Time	216.35	161.04	0.12	s	
	Performance		13.90	18.68	34.65	kByte/s
			834.22	1,120.69	2,078.80	kByte/min

10.4.1.3 Evaluation

Response cycle repetition impact

The study shows the impact of the response slot's cycle repetition. With an increasing value the data transfer rate is decreasing. For this system configuration, where only one *FlowControl* PDU and the final *Response* PDU are sent, the effect is small (only 3% between CR=1 and CR=4). However, if the cycle repetition is 8 or 16 the data transfer rate decrease will be significant. By this system configuration 17 cycles are necessary to transmit 4,080 byte. If the cycle repetition is 16, in worst case the performance will decrease to nearly 50%.

Due to the measurement results, a cycle repetition of 2 or 4 in combination with the configuration that no additional *FlowControl* PDUs are required is a good compromise between data transfer rate and allocated slot resources.

Bandwidth control configuration impact

Bandwidth control allows limitation of the maximum number of PDUs per cycle that can be received by a receiver node. If bandwidth control is enabled the ECU will not receive the maximum number of possible bytes per communication cycle and therefore more cycles are required to transmit all data. This results in a performance decrease. Hence, if an ECU provides not sufficient buffer to receive the maximum number of payload per cycle, the maximum data transfer rate can not reached.

Source bus system's and gateway's impact

The step from CAN bus system with 500 kBit/s to a CAN bus system with 1,000 kBit/s provides a benefit up to 34%. As discussed initially this FlexRay configuration provides a data transfer rate for the request direction (PCU to ECU) of 288⁸⁰ byte per cycle (57.6 kByte/s). The CAN bus system can not support this data transfer rate and is therefore the limiting sub-link in that network. The potential of the FlexRay bus system is visible in figure 10.4-2. The calculation based on the assumption that the ECU requires 20 ms for physical programming. Due to that the download performance is given for the corresponding cycle repetitions.

10.4.1.4 Summary

The study depicts that the currently given performance limitation is based on the vehicle interface bus system CAN. The FlexRay system configuration is able to support higher

⁸⁰ 36 Byte Payload/ PDU_{CF} * 8 PDU / Cycle = 288 Byte / Cycle → 288 Byte / 0.005s = 57.6 kByte/s

data transfer rates. This is illustrated in figure 10.4-2 by the calculated data transfer values. The study depicts also, that the response cycle repetition configuration in the global communication schedule has only a small impact, if the ISO 10681-2 is configured that no additional *FlowControl* PDUs than the initial one are required (disable *ConsecutiveFrame_EOB* mode – refer to section 4.2.2).

In section 4.2 the approach of schedule reconfiguration with the aim to increase data transfer rate was discussed. With a view to the measurement results of this study, schedule reconfiguration is only a powerful approach if either the PCU is connected direct to FlexRay (FlexRay has to be connected to the vehicle connector) or a powerful high speed bus system (e.g. Ethernet) as well as a powerful gateway with a high routing performance are available. In the ahead given network configuration the communication link performance limitation is the CAN bus system. In that case a schedule reconfiguration will have no effect.

Until Ethernet is qualified for automotive usage (refer to chapter 11) all the other methods to reduce data size (compression, partitioning etc.) are necessary to optimise data transfer performance on FlexRay.

11 Conclusion and Outlook

Content

11.1 Summary	189
11.1.1 Method's performance potential	190
11.1.2 Method's potential vs. effort and costs	194
11.1.3 Utilisation in practice	196
11.1.4 Further work.....	197
11.2 Outlook	199
11.2.1 Automotive Ethernet	199
11.2.2 MRAM technology.....	199
11.2.3 Wireless access	200
11.3 Conclusion	200

The ECU software reprogramming process is a necessity within the automotive industry to improve production efficiency/cost, improve reuse and flexibility of the complex embedded systems and perform repair and in field maintenance. Thus, it is in use during the vehicle's complete life cycle. In chapter 1 the main challenges of automotive embedded software and their impacts on the reprogramming method are introduced. Based on several influencing factors, the reprogramming process time is continuously increasing and the commercial benefits of that method are no longer available when compared to changing ECUs. Whilst up to now this time has not caused critical time delays/cost in production or the in-field maintenance period, chapter 1 suggests that this soon will be the case for current systems in development without the introduction of new reprogramming strategies.

This thesis has presented research on new strategies to address the acceleration of the reprogramming process of existing embedded systems technologies and standards by consideration of communication protocol optimisations (chapter 3 and 4) as well as approaches to reduce the total data to transfer (chapter 5). Quantitative models have

been created to allow for predictions of reprogramming times to be calculated during the design development cycle. The thesis also presented recommendations and modelling of new hardware designs within a microcontroller to support faster reprogramming (chapter 6). Network design aspects that influencing the reprogramming time and provide potential for optimisations have been analysed in chapter 7. Quantitative analysis of the reprogramming in parallel as an additional approach was in focus of chapter 8. In chapter 9 a brief introduction to the future MRAM technology, yet to be released, was given and its impact on re-programming quantified. In chapter 10 experimental investigation and analysis have been performed to verify the quantitative theoretical modules previous generated and to evaluate empirically key coefficients and parameters within some of these models.

The thesis does not just analysis the current technologies in production but considers the new technologies and standards currently being considered in the design development cycle and future strategies not yet being considered by designers in prototype research departments. This chapter will introduce to provide the summary of the research work depending on the initially discussed challenges. Also an outlook on how software reprogramming of automotive ECUs will evolve during the next decade and the future challenges will be given.

11.1 Summary

Theoretical work

The thesis has addressed different approaches to reduce the software reprogramming process time for automotive ECUs. The focus was on on-board optimisations. To ensure that the communication on the vehicle connection interface (VCI) bus system was optimised the programming control unit (PCU) was also considered part of the on-board system. The PCU also plays an important part in programming strategy optimisation (e.g. reprogramming in parallel). But PCU implementation details were not part of the research work. Figure 11.1-1 depicts an overview of the thesis' contributions in principle. The research results are summarised below. The different approaches are compared in relation to their power to speed up the ECU's application software reprogramming process.

Case study

The results of the case study confirm the theoretical results and enable important parameters to be quantified. The experimental implementation of the flashloader shows that the methods work in principle and that the discussed limitations are given. Hardware optimisation or implementations for the PCU were not part of the case study. These topics provide potential for additional research work.

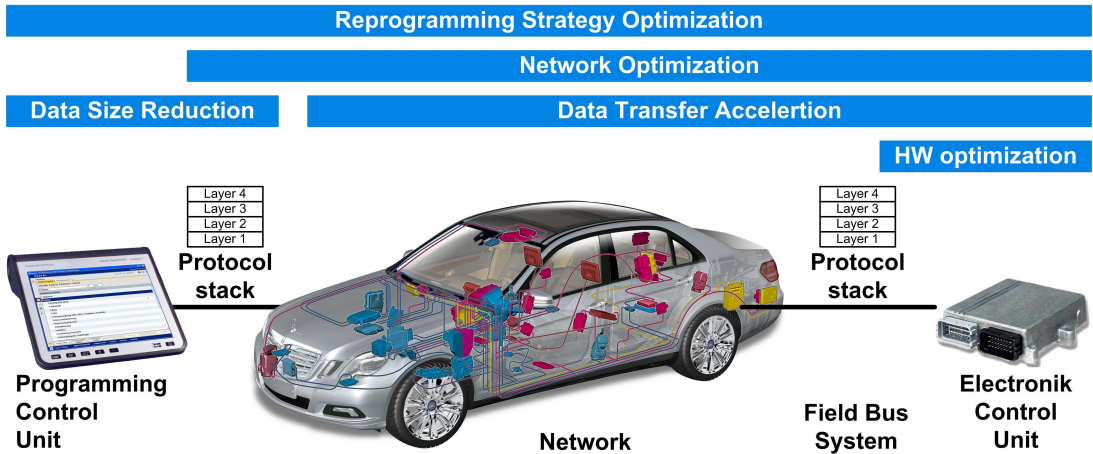


Figure 11.1-1: Optimisation approaches overview

11.1.1 Method's performance potential

The comparison of, and the relation between the different methods is quite difficult because the methods' power depends on the initial system. Figure 11.1-2 depicts the different approaches with their potential to speed up the reprogramming process (graph). The bubbles' position on the graph shows the potential compared to a current state of the art ECU based on a CAN bus system with a bit rate of 500 kBit/s (e.g. for approach of double buffered data transfer (1.1) for this ECU the improvement is approx.15% and for the protocol optimisations (1.2) there would be over 50% improvement).

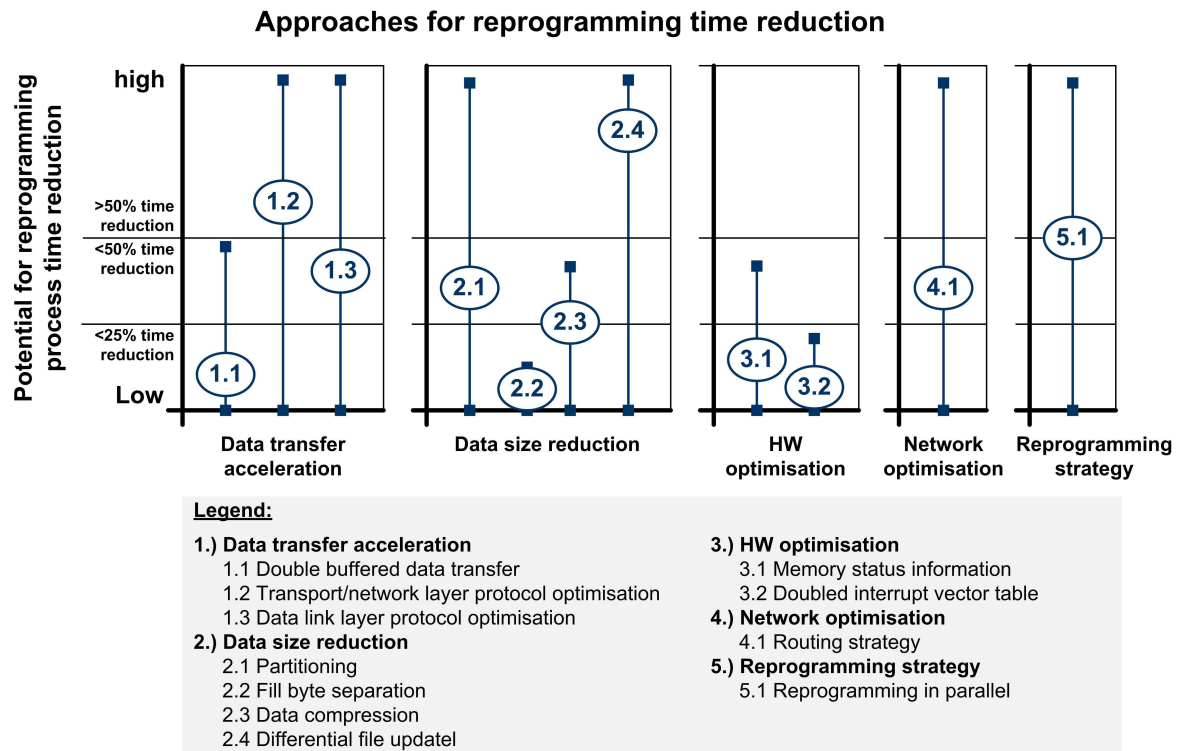


Figure 11.1-2: Optimisation approaches' potential

Data transfer acceleration

Data transfer acceleration is the basic approach to speed up the reprogramming process. The maximum possible bandwidth data transfer rate on the physical bus system can be reduced on several protocol layers within the communication protocol stack.

The potential of **double buffered data transfer** depends on the relation between data transfer time and the microcontroller's reprogramming time. The allocation of a second buffer is typically not a problem because the flashloader has access to the full RAM of the ECU (note that application software is not active during that time). Double buffered data reception is a key functionality for activities in parallel (e.g. data reception and physical reprogramming as well as to compensate the additional time for data de-compression).

Transport layer and network layer optimisations have a high potential to speed up the data transfer. Especially the transport layer's flow control configuration protects the system from additional delays because of separation times between consecutive frames (PDUs).

Data link layer optimisation potential depends on the bus system used and has a wide spread. The effect of increasing the system's bandwidth might be invisible if additional delays are available in the upper layers of the protocol stack. Hence, the delay elimination has the highest implementation priority.

Data size reduction

Partitioning is a very powerful approach to reduce data to be transferred. It is not possible to provide an absolute value for the potential because this value depends on the size relation of the different partitions. Today typically there is a distinction between application software and parameter sets or data sets (characteristic curve etc.) which are allocated in separate partitions. Depending on their size the method's potential is variable. The compatibility aspects of the different partitions have to be taken into account, with focus on complexity.

Fill byte separation was also discussed as a possible approach, but it provides minor effects. Depending on the data transfer rate it might be possible that the gap transfer (requires two additional diagnostic services on UDS) requires more time than the fill byte transfer.

Data compression provides good results. Of course, the compression ratio is limited because a) the usage of lossy compression algorithms is not possible and b) the limited RAM resources do not allow dictionary based algorithms. Nevertheless, a data size reduction up to 30% is possible. To get a good performance for the complete reprogramming

process the additional time for data decompression has to be compensated for. This is possible by the usage of double buffered data transfer in which data reception is ongoing while data decompression and physical reprogramming of the previews data is processed.

Differential file update provides the best theoretical results of all researched approaches. Today the method is only useable with many restrictions. The Flash memory technology used today isn't able to reprogram single bytes. The memory is organised in sections which have to be erased completely before reprogramming is possible. Hence, a difference between current and previews file of only a small number of bytes results in reprogramming of the complete sector. The compiled code memory arrangement (start addresses of functions etc.) also shall not move, because this will result in additional differences of the current and previews file. Nevertheless, this method provides best results if all additional process' requirements are fulfilled.

If the increase of ECU software sizes continues in future, this approach might be the only sustainable one to solve the problem of increasing reprogramming times.

Hardware optimisation

Implementing **memory status information** for a microcontroller's current memory state is helpful to reduce the erase time. The main focus here is on the vehicle manufacturing process, where the differentiation of ECUs is only done by software (e.g. engine control software with different characteristic curves on equal hardware). If this software is reprogrammed within the assembly line, the microcontroller's Flash memory is typically empty and must not be additionally erased. Skipping the physical erase sub-sequence saves time within the complete reprogramming process. The potential of this approach depends on the memory size and the time that is necessary to erase the memory in relation to the data transfer time. A microcontroller with a short erase time and connected to a low bandwidth bus system has only a small potential. When reprogramming an ECU's application software in the case of bug fixing in a garage, the erase process can't be skipped because the previews software must be erased before reprogramming is possible. Hence, the method is not universally usable.

Doubling the **interrupt service routine vector table** is a generic approach. This optimisation provides the possibility to control several activities, e.g. data reception, watchdog triggering or timer handling etc., by interrupts. The benefit compared to the currently necessary polling-mode is the trigger on an event. Only if the event occurs, the trigger is given and the interrupt service routine is processed. The permanent monitoring of the microcontroller's status information (e.g. data reception flags etc.) is no longer necessary and the monitoring time could be used for other activities, e.g. data decompression. In

terms of software reprogramming activities, the real benefit of this method is small because the polling-mode approach is also very fast but with a higher effort for the monitoring routines. Hence, the potential of the pure method is very small.

Network optimisation and design

The communication network (bus systems, gateways etc.) has an impact on the reprogramming performance. Depending on the network type (homogeneous or heterogeneous), the gateways have to route PDUs on different communication protocol layers (according to the ISO/OSI reference model) and has to use different routing strategies (direct routing, store and forward, combinations of both etc.). The higher the layer at which the routing process is executed within the communication protocol stack, the more resources are necessary to speed up this process. Hence, processing runtime (CPU time), as well as resources (RAM for buffers), has to be taken into account for gateway design. The potential of this method is high because of the impact of the routing strategy and the corresponding data transfer time. If timing limits for the reprogramming process are given, (e.g. manufacturing line clock etc.) the network topology must be designed to fulfil the given timing requirements. The fact that the vehicle network cannot be changed during vehicle's life cycle is a problem particular to the automotive industry. Hence, the network design must be able to deal with future ECUs (faster microcontrollers, more memory, more software etc.) and therefore, some performance reserves must be calculated in. Of course, the reserves (additional bus systems or bus systems with higher bandwidth etc.) are expensive and violate cost limits, but the communication network is the most important part in guaranteeing the reprogramming performance during vehicle's life cycle, even when future ECUs are introduced. Hence, communication network design is the key functionality for future vehicle development.

Reprogramming strategy

Reprogramming in parallel is a powerful approach to reduce the total reprogramming time if more than one ECU's application software must be reprogrammed. The potential is high but a corresponding communication network design is a precondition. If bus systems with adequate bandwidth are not available, no communication in parallel is possible. Hence, the network design and the programming strategy are only possible in tandem. A more detailed analysis of the PCU and the ECU ordering for reprogramming was not part of this work, but would be interesting if an additional process optimisation for reprogramming in parallel is possible.

11.1.2 Method's potential vs. effort and costs

The previous section summarised the potential to accelerate the software reprogramming process of all methods discussed in the thesis. Nevertheless, as described in chapter 1, the pressure to maintain or reduce production or in-field service (maintenance) time and keep recurring engineering costs low within the automotive industry is very high because of the high number of cars that are produced per year. Therefore, the relation between the methods' potential and the effort for implementation has to be discussed. This is necessary to support decisions for future implementation strategies for ECU hardware selections, vehicle network architecture and design, the vehicle communication interface bus system etc. A criterion to differentiate costs is the effort to implement the method. It has to be distinguished between different effort types and therefore different cost impacts:

- a) Effort in software (SW) to implement the method
(These are typically singular costs for the initial implementation).
- b) Effort in hardware (HW) to implement the method
(Typically additional hardware costs are costs per ECU or vehicle and have therefore a high weight).
- c) Other efforts e.g. external overhead.

Table 11.1-1 depicts the different approaches in dependency of the necessary effort in case of implementation or realisation.

Table 11.1-1: Reprogramming process acceleration methods' effort

	Method	Effort		
		SW	HW	other
1.1	Double buffered data transfer	low		
1.2	Transport / network layer protocol optimisation	low		
1.3	Data link layer protocol optimisation	low	mid	
2.1	Partitioning	low		mid
2.2	Fill byte separation	low		
2.3	Data compression	low		
2.4	Differential file update	high	high	high
3.1	Memory status information		high	
3.2	Doubled interrupt vector table		high	
4.1	Routing strategy	mid	high	
5.1	Reprogramming in parallel			low

Based on table 11.1-1 and figure 11.1-2 a quantitatively relation of reprogramming performance potential vs. implementation and realisation effort is possible. Figure 11.1-3 depicts that relation in a graph.

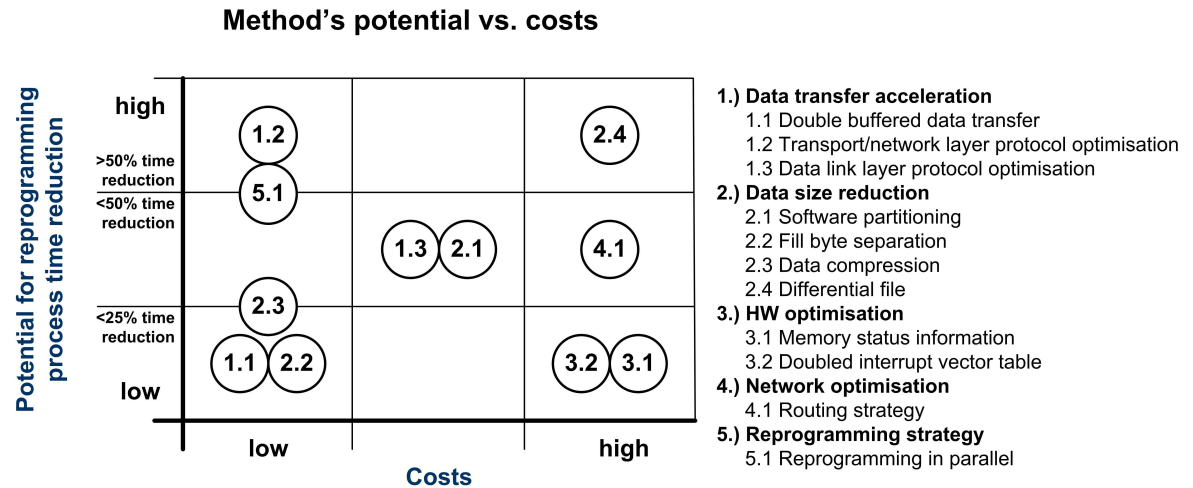


Figure 11.1-3: Method's potential vs. costs

These findings from the research work, based on quantitative evaluation of reprogramming times, make several contributions to future ECU, network and vehicle development decisions. The different areas are discussed below.

Low costs for implementation

Five methods can be implemented with only a small effort. Those methods require only an optimisation in software. **Protocol optimisations** (1.2) provide the best results when only optimising the configuration parameter set. **Reprogramming in parallel** (5.1) only requires an additional algorithm in the PCU to order the different ECU reprogramming activities. There is no change necessary in the flashloader software. The **data compression** method (2.3) is an additional software part within the flashloader. The method has to be implemented once and can be used for all microcontrollers because the method itself is hardware-independent. RAM resources for buffer are not critical because the Flashloader has access to the complete ECU RAM. This reasoning also applies to the **double buffered data transfer** method (1.1). **Fill byte separation** (2.2) is a method implemented in the software development process (linker method).

Medium costs for implementation

Costs for optimisation on the **data link layer protocol** (1.3) depend on the initial system. If a high speed CAN bus system is available, maximum bandwidth can be configured without any additional activities. The only precondition is that the resulting maximum cable lengths are sufficient for the network, because they are reduced in case of increasing

bandwidth⁸¹. The **partitioning method** (2.1) is very efficient but provides logistic effort. The different software parts of an ECU have to be managed (documentation, identification, compatibility etc.). Of course, these are typically software management processes but the effort is given.

High costs for implementation

Each kind of **hardware optimisation** (3.1 and 3.2) provides high costs because the micro-controllers have to be changed. If those methods become state of the art in the future, the cost benefit relation will be better, but today the methods are too expensive for realisation. The **routing strategies** in gateways (4.1) have a high potential but require high resources. Typically a high speed microcontroller with high clock rate is necessary to provide the CPU time for the routing process and the performance to do this for several connections in parallel. The high clock frequency has an impact to the EMC⁸² strategy and results in additional hardware to reduce radiant emittance. The best potential by highest effort is provided by the **differential file** programming method (2.4). The effort is high within the PCU which has to calculate the difference as well as within the flashloader to calculate the new file and reprogram it. The currently used flash memory is the reason for the on-board complexity because it is not possible to reprogram a) without previously erasing and b) only a few bytes. Erasing complete sections of several kByte is required before reprogramming is possible. Hence, the non-different data has to be saved and temporary stored in a RAM mirror of the flash memory. Typically a microcontroller doesn't provide the required RAM size for this method. Finally the development process of embedded systems' application software has to support the differential file approach. That means that software must be generated in a special way that only small differences occur between different software releases. But this was not in focus of this work.

11.1.3 Utilisation in practice

The findings of this study have a number of important implications for utilisation in practice.

Recommendation for implementation from today's point of view

The initial problem of increasing reprogramming times because of increasing software sizes can partly be solved by the short term implementation of the low cost methods. Especially the communication protocol optimisations (refer to chapter 3 and chapter 4) provide short term results on low costs.

⁸¹ This could be a problem for trucks with long cable sizes from truck to trailer.

⁸² EMC - electromagnetic compatibility

For the medium costs methods an analysis about of the real effort is necessary. In some cases (where the hardware impact is small) a benefit is given without any other implications.

The network design is quite expensive but necessary because of the vehicle's long life cycle (refer to chapter 1.3.4). The network must be able to process the data communication links as fast as possible to guarantee the basically required speed performance of 100% bus load. Nevertheless, network design decisions are made several years before vehicles start of production and can not be revised quite easily. Hence, it is strongly recommended to implement all network optimisations in future cars because of vehicle's, and therefore networks, long time life cycle.

For the methods "reprogramming by differential file (2.4)" and "hardware optimisations (3.1 and 3.2)" the implementation is currently not recommended because of the high implementation effort and therefore, high costs. Additional research work and new hardware technologies will be necessary to reduce costs before these approaches will be usable in the automotive industry.

11.1.4 Further work

The focus on this thesis was the acceleration of the embedded systems' software reprogramming process. The research that has been undertaken in this thesis has highlighted a number of different topics to solve the given challenges. First results are provided and a classification of method's potential and their effecting to costs was investigated. Nevertheless, there are several lines of further research arising from this work.

Moving knowledge to future automotive communication protocol stacks

All the discussed topics in chapter 3 and 4 become also important if new bus systems are introduced. For each new bus system the protocol stack has to be optimised with focus on data transfer rate for software reprogramming purpose. In a first step, research work to automotive usable Ethernet (refer to chapter outlook) is necessary as a base technology to solve several challenges within automotive communication aspects.

More effective compression algorithms that consider to embedded system's resources

As discussed in section 5.3 the very special resource situation of embedded systems (e.g. RAM, clock frequency etc.) does not allow the usage of all possible lossless compression algorithms. The good costs to performance relation as discussed above excuses further research work to develop more effective compression algorithms for utilisation in embedded systems.

Tool supported network analysis and design

The discussed complexity of currently available and future developed vehicle networks requires tools to support the network design and analysis process (refer to appendix D). The communication model of chapter 4 can be the base for performance analysis focused on diagnostic and reprogramming communication. Nevertheless, tool development aspects, e.g. internal data models, calculation or simulation speed performance etc., have to be discussed and solved.

Consolidation and concentration of ECUs

A modern vehicle implements up to 80 different ECUs. This high number of independent network nodes makes a strong contribution to the currently available network complexity. An interesting question for further research work is about optimisation potentials if the given functionality is concentrated to only a few, but powerful ECUs. Cost aspects as well as packaging in the vehicle and increasing systems complexity have to be taken into account.

High speed vehicle access

A precondition for processing different communication links in parallel is a high speed vehicle access (e.g. high speed bus system). The currently established CAN bus system will be no longer sufficient (refer to chapter 8 and the case study in chapter 10). FlexRay might be a possible alternative but the complexity of time triggered protocol might provide other disadvantages. Ethernet as a common standard is currently still too expensive for automotive usage (connectors, shielded cable etc.) but different vehicle manufacturers and system suppliers have started an initiative to develop and standardise automotive usable Ethernet (refer to the outlook chapter).

Programming Control Unit

The offboard technology was not in focus of this thesis. But software has to be managed offboard. Because of increasing dependencies of different ECU (software, routines, functionalities etc.) complexity will continuously increase and the documentation of compatibilities becomes more importance.

Compatibility management of embedded software releases.

The differential files approach based on the fact that different software releases have only small differences of their OP-code. It seems that completely new methods and strategies for the embedded software development process are necessary, compared to the today's established processes. Hence, more research is needed to better understand what must be changed to support the differential files approach for software reprogramming.

11.2 Outlook

The recommended further work will be supported by the ongoing development, not only within the automotive industry. Some technology aspects, which have been identified in this thesis as a precondition to raise the next evolution step, will appear on technology's horizon. Hence, it will be only a question of time when these technologies will be ready for automotive usage.

11.2.1 Automotive Ethernet

Ethernet will be the bus system for the next generation of vehicle communication interface. In the past *Ethernet* according to [IEEE 802] standard was too expensive for automotive usage. Shielded cable and unpractical connectors have prevented the introduction. Since November 2011 a new alliance of OEMs, ECU and semiconductor distributors was formed. The aim is to establish chipmaker Broadcom's⁸³ *BroadR-Reach* technology as an open standard for *One-Pair-Ethernet (OPEN)* [Auo11].

Ethernet as the vehicle interface bus system supports the possibility for reprogramming in parallel because of high bandwidth. In that case the limitation of the reprogramming process will be the microcontroller's physical reprogramming process.

11.2.2 MRAM technology

The next evolutionary step in embedded memory technologies will be *Magnetoresistive Random Access Memory (MRAM)*. The advantages of MRAM based systems are quite evident. In contrast to the currently established Flash memory technology MRAM provides byte-wise access and the possibility to overwrite data without an initial memory erase phase.

The byte-wise access allows the usage of the differential file method for software reprogramming as discussed previously. The Flash memory disadvantage of storing the non-changed bytes into RAM mirror is not longer given. Of course, the effort of this method is high (refer to chapter 4) but the benefit is enormous. The data transfer time could be reduced significantly and this will finally solve the initial problem.

Hence, industry is looking forward to the introduction of MRAM based embedded micro-controllers.

⁸³ refer to [Broadcom]

11.2.3 Wireless access

The continuously ongoing trend in vehicle inter-connection (car-to-car) and the inter-connection of vehicles networks and non-vehicle networks (car-to-X) requires wireless access points within the cars. These infrastructures allow software reprogramming via wireless connections, too. The potential is enormous because software reprogramming within the service or after sales market is not longer bound by visiting the garage. The technique provides also benefits during the manufacturing process because the handling of the wired PCU is not longer necessary.

LTE (Long Term Evolution) as a new global standard for mobile communication networks provides high potential. If the vehicle implements an interface, a remote vehicle access is nearly everywhere given. Of course, some additional topics are currently limiting factors, especially with focus on software reprogramming. 1) The power supply (vehicle battery) must provide the power to keep the vehicle network awake until software is reprogrammed without a running engine. On the other hand, the battery's charge condition shall be good enough to restart the vehicle at any time. Especially for new vehicles with electric drive this is an important criterion. 2) The data have to be stored temporary within the vehicle to reduce long online times. Due to that an onboard PCU has to be introduced within a vehicle as well as a large temporary memory unit. Both will increase costs per vehicle. 3) Wireless communication to a vehicle requires high security standards especially if software shall be remote reprogrammed. Security within this context means either security against unauthorised access as well as security to the overall process. It must never be possible that the vehicle is not usable because of an unsuccessful programming attempt. Here some given concepts of the PC world can be moved to the embedded world with the challenges of less resources and computing performance.

Nevertheless all discussed approaches within this thesis to accelerate data transfer will support the wireless activities, too, because reduced data transfer and processing times will reduce the time to be online and therefore, reduce costs.

11.3 Conclusion

The scope of the work reported in this thesis was on the on-board part of the global software reprogramming process for embedded systems. ECU aspects (flashloader, application software, network access, communication protocols etc.) and network aspects (architecture, topology etc.) were the focus on investigation. As described in section 1.4.1 for future ECUs the given time limitation requirement to the maximum reprogramming time will not fulfilled any longer without any optimisations.

As a major outcome of this thesis several methods have been investigated with different potential to solve the initial problem. Depending on future embedded systems' software sizes and the automotive industry's cost aspects, the investigation result methods can be combined to short term, mid term and long term solutions. The impacts of implementation efforts, given technologies and the availability of future technologies have been taken into account. Figure 11.3-1 depicts a quantitative view to the roadmap and the different steps of performance potential.

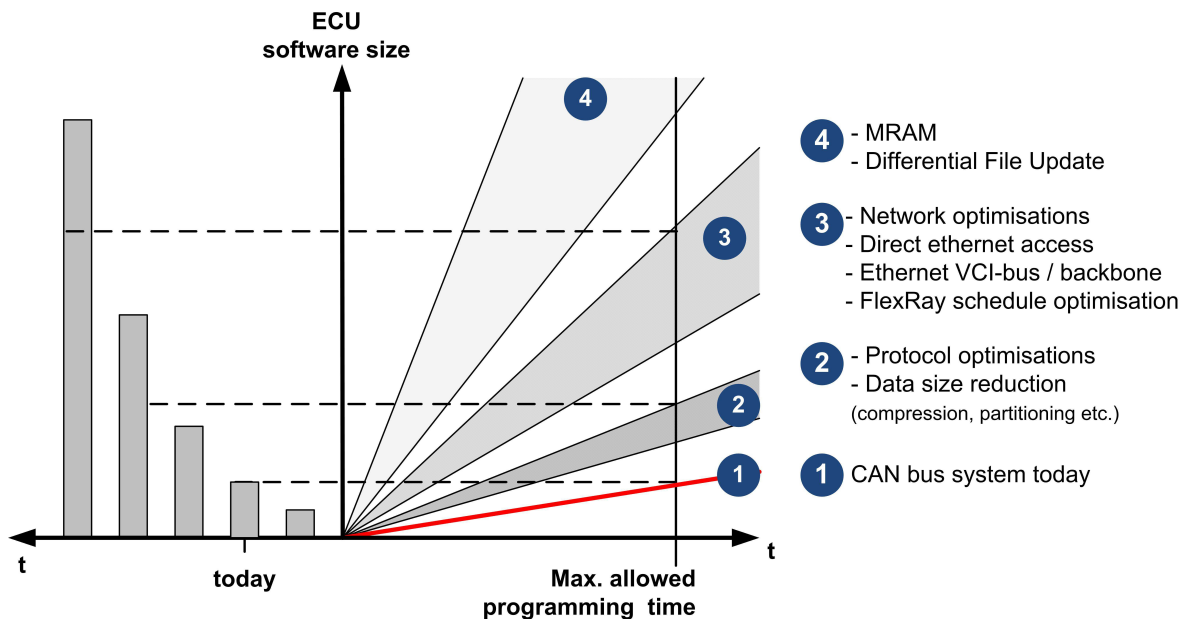


Figure 11.3-1: Reprogramming method's implementation roadmap

The software size for embedded systems will not stagnate. Especially within the vehicle industry the innovation will take place through software functionality. Hence, the problem of increasing software reprogramming times is permanently given. Of course, by most of the discussed approaches to accelerate the reprogramming process, the problem could be solved for a view years but the increasing software sizes will force the problems again. It might be possible that the combination of all methods could enlarge that time but the problem cannot be solved forever by the current memory technology (Flash memory) and automotive bus system technology (CAN).

Hence, the long term solution will be the combination of new memory technologies (e.g. MRAM), the compressed data transfer of ECU's partition specific differential file via high speed bus systems in optimised networks and for software reprogramming purpose optimised microcontrollers.

This thesis made some contributions on the way to that ambitious aim.

12 Figures

Figure 1.1-1: Vehicle model line life cycle	3
Figure 1.2-1: Reprogramming stages within an ECU's life cycle	4
Figure 1.2-2: Automotive ECU software volume.....	5
Figure 1.3-1: Amount of vehicle software of Mercedes-Benz	8
Figure 1.3-2: Most implemented automotive field bus systems	10
Figure 1.3-3: Shift from single to system innovation [Dan07]	11
Figure 1.3-4: Increasing network complexity	13
Figure 1.3-5: Vehicle development trends.....	14
Figure 1.4-1: Software reprogramming process circle.....	15
Figure 1.4-2: Software reprogramming time limitation.....	16
Figure 2.1-1: Embedded system components.....	19
Figure 2.1-2: ECU network of a Mercedes-Benz Model line 221 (S-Class) [Mer09]	20
Figure 2.1-3: Mercedes-Benz Model Line 221 (S-Class) network architecture [Mer09-1] 20	
Figure 2.2-1: Memory Technologies Overview [Rei11].....	22
Figure 2.2-2: ECU Software Components Overview	24
Figure 2.2-3: Overview flashloader component.....	26
Figure 2.4-1: Abstract major programming sequence	29
Figure 2.5-1: Communication structure within a protocol stack	34
Figure 2.5-2: Protocol Stack Overview – Transport Layer	37
Figure 2.7-1: System model for embedded system's software reprogramming	40
Figure 3.1-1: Reprogramming protocol overview.....	42
Figure 3.2-1: Single buffer data transfer.....	43
Figure 3.2-2: Double buffered data transfer – scenario 1	45
Figure 3.2-3: Maximum time reduction for $t_{\text{Data Transfer}} \geq t_{\text{Physical Programming}}$	46
Figure 3.2-4: Total reprogramming time reduction - details	47
Figure 3.2-5: Double buffer data transfer – scenario 2	48
Figure 3.3-1: UDS communication via single buffered system	50
Figure 3.3-2: UDS communication via double buffer system.....	50
Figure 3.3-3: Multi controller system	51
Figure 4.1-1: ISO 15765-2 Protocol Data Units format.....	58
Figure 4.1-2: ISO 15765-2 communication scenarios	59

Figure 4.1-3: Request / response communication scenarios based on ISO 15765-2	60
Figure 4.1-4: Block size analysis for 11 bit CAN identifier	64
Figure 4.1-5: Block size analysis for 29 bit CAN identifier	64
Figure 4.1-6: Data transfer rate depending on STmin	67
Figure 4.1-7: Data transfer rate depending on STmin – detailed diagram	67
Figure 4.1-8: Comparison of impact of STmin and BS	68
Figure 4.2-1: FlexRay Schedule.....	74
Figure 4.2-2: Maximum FlexRay net data rate ($t_{\text{cycle}} = 1 \text{ ms}$)	79
Figure 4.2-3: Maximum FlexRay net data rate ($t_{\text{cycle}} = 5 \text{ ms}$)	79
Figure 4.2-4: FlexRay net data rate ($t_{\text{cycle}} = 1 \text{ ms}$)	80
Figure 4.2-5: FlexRay net data rate ($t_{\text{cycle}} = 5 \text{ ms}$)	80
Figure 4.2-6: FlexRay schedule optimisation	82
Figure 4.2-7: FlexRay communication protocol PDU format.....	85
Figure 4.2-8: FlexRay communication layer scenarios.....	86
Figure 4.2-9: FlexRay Bandwidth assignment.....	86
Figure 4.2-10: Data transfer according to ISO 10681-2.....	87
Figure 5.1-1: Application layer partitions [AUT11]	97
Figure 5.2-1: Data transfer with and without fill bytes.....	100
Figure 5.3-1: LZSS algorithm.....	105
Figure 5.3-2: LZSS Compression Results [Hee04].....	105
Figure 5.3-3: LZSS Optimisation.....	107
Figure 5.4-1: Modified Op-Code in case of bug fixing	108
Figure 5.4-2: Differential file update	109
Figure 5.4-3: Differential file update	110
Figure 5.5-1: method's complexity vs. typical data size reduction	114
Figure 6.1-1: Early variant building vs. late variant building.....	117
Figure 6.1-2: Memory Status Information Register.....	120
Figure 6.2-1: Single ISR vector table vs. multiple ISR vector tables.....	121
Figure 6.3-1: Potential vs. effort of hardware implementation	124
Figure 7.1-1: Network Classification.....	128
Figure 7.2-1: Routing within the AUTOSAR layered software architecture.....	129
Figure 7.3-1: Routing strategy.....	130
Figure 7.3-2: Routing performance	132
Figure 7.3-3: Partly store and forward routing strategy.....	133
Figure 7.3-4: Ideal number of buffers	134
Figure 7.4-1: Routing performance vs. resources	136

Figure 7.4-2: Design based on timing limitations.....	137
Figure 7.4-3: Source bus and target bus definition for reprogramming in parallel.....	139
Figure 7.4-4: Different, scenario oriented views to the same network	140
Figure 8.1-1: Network classification	142
Figure 8.2-1: Bandwidth capacity utilisation	144
Figure 8.2-2: Priority calculation on a network with 4 ECU.....	145
Figure 8.2-3: Schedule calculation on a network with 4 ECU	146
Figure 8.2-4: Schedule calculation for networks with cascaded bus systems.....	147
Figure 8.4-1: ODX integration to support reprogramming in parallel.....	150
Figure 9.1-1: Bit information storage based on the MTJ effect	152
Figure 10.1-1: Test environment	158
Figure 10.1-2: Segmented data transfer according to ISO 15765-2	159
Figure 10.1-3: Segmented data transfer according to ISO 15765-2	162
Figure 10.1-4: Software reprogramming performance on CAN	166
Figure 10.1-5: Impact of Flow Control parameter BS	169
Figure 10.1-6: Impact of Flow Control parameter STmin.....	171
Figure 10.2-1: Measurement results – best case relation for all scenarios	174
Figure 10.2-2: Contribution of the different parameters	174
Figure 10.2-3: Test result timing evaluation	176
Figure 10.3-1: Gateway test system overview.....	178
Figure 10.3-2: Data transfer rate for different buffer scenarios.....	179
Figure 10.4-1: FlexRay test system overview.....	183
Figure 10.4-2: FlexRay data transfer rate.....	185
Figure 11.1-1: Optimisation approaches overview	190
Figure 11.1-2: Optimisation approaches' potential	190
Figure 11.1-3: Method's potential vs. costs.....	195
Figure 11.3-1: Reprogramming method's implementation roadmap.....	201

13 Tables

Table 1.2-1: World automotive production [Vda11]	7
Table 1.3-1: Software release types.....	12
Table 2.2-1: ECU's semiconductor memory overview [Zim10-2].....	23
Table 2.2-2: Physical programming performance.....	23
Table 2.4-1: Software Re-Programming Process according to [HIS06-2] and [Zim10-3] .	31
Table 2.5-1: OSI reference model.....	33
Table 2.5-2: Field bus systems in automotive area	35
Table 2.5-3: Automotive related transport protocol specifications	37
Table 4.1-1: CAN PDU length without stuff bits.....	55
Table 4.1-2: Net data rate for CAN PDUs with 64 bit payload	56
Table 4.1-3: Parameter definition for block size analysis.....	62
Table 4.1-4: $f_{\text{Data_Net max}}$ for different bandwidths	63
Table 4.2-1: FlexRay PDU length	75
Table 4.2-2: FlexRay PDU length and PDU runtime for $f_{\text{bit}}=10$ MBit/s	76
Table 4.2-3: ISO 10681-2 PDU overview	85
Table 4.2-4: ISO 10681-2 Flow Control (FC) PCI Overview [ISO 10681-2]	89
Table 4.2-5: ISO 10681-2 Definition of Bandwidth Control (BC) values [ISO 10681-2]....	89
Table 5.2-1: Break even calculation.....	102
Table 5.5-1: Data transfer time via CAN	113
Table 6.1-1: Infineon TC1197 Flash Parameter [TC1197].....	118
Table 8.1-1: Reprogramming scenarios	142
Table 9.1-1: Comparison of expected MRAM features with other memory technologies [Fre07].....	153
Table 9.2-1: Example of microcontroller's erase time for Flash memory	154
Table 9.3-1: Differential file approach comparison to Flash and MRAM technology	156
Table 10.1-1: PCU and ECU communication configuration parameters.....	160
Table 10.1-2: Measurement results of PCU and ECU processing parameters	160
Table 10.1-3: Communication model's calculation of data transfer rate	161
Table 10.1-4: Communication model's calculation of data transfer rate	161
Table 10.1-5: PCU and ECU communication configuration parameters.....	163
Table 10.1-6: Measurement results of PCU and ECU processing parameters	163

Table 10.1-7: Communication model's calculation of data transfer rate	165
Table 10.1-8: PCU and ECU communication configuration parameters	166
Table 10.1-9: Software reprogramming performance on CAN.....	167
Table 10.1-10: PCU and ECU communication configuration parameters	168
Table 10.1-11: Flow Control parameter "BS" measurement results.....	168
Table 10.1-12: PCU and ECU communication configuration parameters	170
Table 10.1-13: Flow Control parameter "STmin" measurement results	171
Table 10.2-1: Flow Control parameter "STmin" measurement results	173
Table 10.2-2: Measurement results of different data transfer acceleration scenarios	173
Table 10.2-3: Measurement results – detailed analysis.....	175
Table 10.2-4: Decompression routine runtime	177
Table 10.3-1: Trace – data transfer for gateway with 2 buffer resources.....	180
Table 10.3-2: Routing performance on different microcontrollers	181
Table 10.4-1: Data transfer performance based on CAN as vehicle interface bus system.....	185
Table 11.1-1: Reprogramming process acceleration methods' effort.....	194

14 Bibliography

Publications (books, magazines, journal papers etc.) are documented by the author's first three characters followed by the last two digits of the publication years. If an author had several publications within the same year, an additional number was attached.

International standards (e.g. ISO, SAE etc.) are documented by their full number.

Publications within the internet are documented by 6 lower case characters

Companies and Organisations are documented by their full name.

ASAM	Association for standardisation of automation and measurement systems. www.asam.net
AUTOSAR	AUTOSAR, Bernhard-Wicki-Strasse 3, 80636 Munich, Germany www.autosar.org
Auo11	Automotive IT, "Alliance formed to advance ethernet for in-car connectivity", 13.11.2011, AutomotiveIT http://www.automotiveit.com/alliance-formed-to-advance-in-car-ethernet/news/id-004435
Aut11	AUTOSAR - Layered software architecture, Specification 4.0, AUTOSAR_EXP_LayeredSoftwareArchitecture, Document-ID 053 , Version 3.1.0, Revision 2, page 57, www.autosar.org
Bar07	Barr, M.: "Embedded Systems Glossary", Netrino Technical Library. http://www.netrino.com/Embedded-Systems/Glossary . Retrieved 2007-04-21
Bel57	Bellman, R. E.: "Dynamic Programming", Princeton University Press, 1957
BMW	BMW AG, Munich www.bmw-group.com
Broadcom	Broadcom Corporation, 5300 California Avenue, Irvine, California 92617, www.broadcom.com
Bro11	Broy, M.: "Mit welcher Software fährt das Auto der Zukunft?", ATZ extra, Springer Automotive Media, April 2011, Page 92 – 97
Bro11a	Broy, M., Reichart, G., Rothhardt, L.: "Architekturen software-basierter Funktionen im Fahrzeug: von der Anforderung zur Umsetzung", 2011, Informatik Spectrum, ISSN: 0170-6012, Band 34 Heft 1, Page 42-59
Bur07	Burns,A., Davis R., Bril R., Lukkien J.: "CAN Schedulability Analysis: Refuted, Revised and Revisited", Real-Time Systems Journal, Springer Verlag Heft 3, 2007, Page 239-272
CiA102	CiA 102: CAN in Automation - CAN Physical layer for industrial applications, V3.0.0, www.can-cia.org
Cou01-1	Coulouris, G., Dollimore, J., Kindberg, T.: "Distributed Systems – Concepts and Design", 3 rd edition 2001, Addison-Wesley, ISBN 0201-61981-0, Chapter 3.1 – Introduction
Daimler	Daimler AG, 70546 Stuttgart, Germany www.daimler.com

- Dan07 Dannenberg, J., Burgard, J. et. al.: "Car innovation 2015 – A comprehensive study on innovation in the automotive industry", 2007, Oliver Wyman Automotive, www.oliverwyman.com
- dataco Data compression, <http://de.wikibooks.org/wiki/Datenkompression> (last access 18.02.2011)
- Dra11 Draeger, K.: "Das Automobil in einer vernetzten Welt", ATZ extra, Springer Automotive Media, April 2011, Page 22 – 26
- Ets06 Etschberger, K.: "Controller Area Network", Hanser Verlag, 3. Auflage 2006
- Fle05 FlexRay Consortium: "FlexRay Communications System, Protocol Specification Version 2.1", May 2005, Chapter 4: Frame Format
- Fle11 FlexRay consortium, <http://www.flexray.com>, last access: 2011
- For62 Ford, L.R., Fulkerson, D. R.: "Flows in networks", Princeton University Press, 1962
- Fre07 Freescale Semiconductors; MRAM fact sheet, Document number MRAMTECHFS Rev.6, 2007
- Fre07-1 Freescale Semiconductors; Data sheet "256k x 16bit 3.3V Asynchronous MRAM" document number MR2A16A Rev.6, 11/2007. www.freescale.com
- Ham03 Hammerl, A., Bag, H.: "MRAM – Magneteische Speicher", 2003, Studienarbeit an der TU Wien, http://upload.wikimedia.org/wikipedia/de/f/fb/MRAM_V3.pdf (last access 2009)
- Hau11 Haub, M., Mathes, J.: "Fahrerassistenz der Zukunft – Fahren oder gefahren werden", ATZ extra, Springer Automotive Media, April 2011, Page 72 – 76
- Hee04 Hees, F.: "Transfer of Compressed Binary Data", Vector Informatik GmbH, 2004, www.vector-informatik.com
- Hen03 Hennessy, J. L., Patterson D. A.: "Computer Architecture – A Quantitative Approach", 3rd edition 2003, Morgan Kaufmann Publishers, ISBN 1-55860-724-2,
- HIS06-1 Hersteller Initiative Software: "HIS-konforme Programmierung von Steuergeräten auf Basis von UDS", V1.0, 2006, <http://www.automotive-his.de/>
- HIS06-2 Hersteller Initiative Software: "HIS-konforme Programmierung von Steuergeräten auf Basis von UDS", V1.0, 2006, Chapter 3.2.4.1 Boot manager, <http://www.automotive-his.de/>
- huffma Huffman coding
http://en.wikipedia.org/wiki/Huffman_coding (last access: 10.01.2011)
- Huh06 Huhn, W., Schaper, M.: "Getting better software into manufactured products", McKinsey on IT – Innovations in IT management, Number 7, Page 3 – 7, Spring 2006
- IEC 61158 IEC 61158: "Digital data communication for measurement and control - Fieldbus for use in industrial control systems"
- Infineon Infineon technologies AG,
www.infineon.com
- Int88 INTEL: "Hexadecimal Object File Format Specification", Revision A, 06.01.1988
<http://microsym.com/editor/assets/intelhex.pdf> (last access: 01/2010)
- ISO International Organization for Standardization
ISO Central Secretariat, 1, ch. de la Voie-Creuse, CP 56
CH-1211 Geneva 20
Switzerland, www.iso.org
Public standards:
<http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html>

- ISO7498-1 ISO/IEC 7498-1:1994: "Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model"
- ISO10681-2 ISO 10681-2: "Road Vehicles - Communication on FlexRay – Part 2: Communication Layer Services", 2010, www.iso.org
- ISO10681-2_1 ISO 10681-2: "Road Vehicles - Communication on FlexRay – Part 2: Communication Layer Services", 2010, Table 17, www.iso.org
- ISO10681-2_2 ISO 10681-2: "Road Vehicles - Communication on FlexRay – Part 2: Communication Layer Services", 2010, Table 21, www.iso.org
- ISO11898-1 "ISO 11898-1: Road Vehicles - Controller area network (CAN) – Part 1: Data link layer and physical signalling", 2003 and 2006, www.iso.org
- ISO11898-2 ISO 11898-2: "Road Vehicles - Controller area network (CAN) – Part 2: High-speed medium access unit", 2003, www.iso.org
- ISO11898-3 ISO 11898-3: "Road Vehicles - Controller area network (CAN) – Part 3: Low-speed, fault-tolerant, medium dependent interface", 2006, www.iso.org
- ISO11898-5 ISO 11898-5: "Road Vehicles - Controller area network (CAN) – Part 5: High-speed medium access unit with low-power mode", 2007, www.iso.org
- ISO14229 ISO 14229: "Road vehicles – Unified diagnostic services (UDS) – Specification and requirements", 2006-12, www.iso.org
- ISO14230-3 ISO 14230-3: "Road vehicles – Diagnostic systems - Keyword Protocol 2000", 1999-03, Part 3: Application Layer. www.iso.org
- ISO15765-2 ISO 15765-2: "Road vehicles – Diagnostics on Controller Area Networks (CAN) — Part 2: Network layer services", 2004-10, Chapter 4.1 General, www.iso.org
- ISO15765-2_2 ISO 15765-2: "Road vehicles – Diagnostics on Controller Area Networks (CAN) — Part 2: Network layer services", 2004-10, Table 14 - Definition of BS value, www.iso.org
- ISO15765-2_3 ISO 15765-2: "Road vehicles – Diagnostics on Controller Area Networks (CAN) — Part 2: Network layer services", 2004-10, Chapter 7.3 – Mapping of the N_PDU fields, www.iso.org
- ISO15765-2_4 ISO 15765-2 2004-10, Road vehicles – Diagnostics on Controller Area Networks (CAN) — Part 2: Network layer services, Chapter 6.5.5.5 Definition of SeparationTime (STmin) parameter, www.iso.org
- Kar11 Karic, S.: "Evaluierung und Implementierung einer Datenmodellkonvertierung von AUTOSAR-Systemdescription nach ODX-Vehicle-Information-Specification", Wilhelm Büchner Hochschule, 2011
- Mar07 Marscholik, V., Subke, P.: "Datenkommunikation im Automobil", 1. Auflage 2007, Hüthig GmbH & Co. KG, ISBN 978-3-7785-2969-0, Chapter 5: Unified Diagnostic Services
- MC9S12X Freescale Semiconductors; MC9S12XEP100 Reference Manual V1.18 – Chapter 2.3.6.5 Program and data Flash; 2008; www.freescale.com
- merced Mercedes-Benz, Lane Departure Warning System
<http://media.daimler.com/dcmmedia/0-921-614216-1-1147529-1-0-0-0-0-0-11702-0-0-1-0-0-0-0-0.html>
- Mer09 Mercedes-Benz, Model Line 221 (S-Class), 08/2009, www.mercedes-benz.com
- Mer09-1 Mercedes-Benz, Model Line 221 (S-Class), www.mercedes-benz.com

- Mot92 MOTOROLA Inc.: "Programmer's Reference Manual", Appendix-C S-Record Output Format, 1992,
http://www.freescale.com/files/archives/doc/ref_manual/M68000PRM.pdf (last access: 01/2011)
- Nol01 Nolte, T., Hansson, H., Norström, c., Punnekkat, s.: "Using Bit-stuffing Distributions in CAN analysis", IEEE Real-time Embedded Systems Workshop, London, 2001
- OCIA ORGANISATION INTERNATIONALE DES CONSTRUCTEURS D'AUTOMOBILES
4 rue de Berri, 8ème arrondissement, Paris, France
www.oica.net
- oica Organisation internationale des Constructeurs d'Automobiles: "Provisional Production Statistics", 2010, <http://oica.net/category/production-statistics/>
- Pot05 Potter, S.: "Using Binary Delta Compression (BDC)", Technology to Update Windows XP and Windows Server 2003, June 2005, Microsoft Cooperation
<http://www.microsoft.com/downloads/en/details.aspx?FamilyID=4789196c-d60a-497c-ae89-101a3754bad6&DisplayLang=en>
- Rau07 Rausch, M.: "FlexRay – Grundlagen, Funktionsweisen, Anwendungen", 1st Edition 2007, Hanser Verlag, ISBN 978-3446412491
- Rei11 Konrad, R.: "Elektrik und Elektronik - Steuergeräte, Aktoren und Mechatronik", 6th edition 2011, Vieweg+Teubner Verlag, ISBN: 978-3-8348-1274-2, Series: Bosch Fachinformation Automobil
- Ren05 RENESAS (2005) , Renesas Edge 2005-Vol 9, page 12; www.renesas.com
- Ren11 RENESAS Electronics, "Renesas Microcomputer General Catalog", page 1-2, 2011.06, R01CS0001EJ0201, www.renesas.com
- SAE J1939/11 SAE J1939/11 – "Physical Layer – 250 kBits/s, Shielded Twisted Pair", 1999-10, www.sae.org
- Sai04 Said, A.: "Introduction to Arithmetic Coding - Theory and Practice", Hewlett-Packard Laboratories Report, HPL-2004-76, Palo Alto, CA, April 2004.
- Say05 Savood, K.: "Introduction to data compression", 3rd edition 2005, Morgan Kaufmann, ISBN-10: 012620862X
- Say05-1 Savood, K.: "Introduction to data compression", 3rd edition 2005, Morgan Kaufmann, ISBN-10: 012620862X, Chapter 2 – Mathematical preliminaries for lossless compression.
- Scha10 Schäuuffele, J., Zurawka, T.: "Automotive Software Engineering", 4th edition, 2010, Vieweg+Teubner Verlag, ISBN: 978-3-8348-0364-1,
- Sch10 Schmidgall, R.: "Diagnostic Communication within networks based on AUTOSAR configuration", 5th Vector Congress 2010, December 2010, http://www.vector.com/portal/medien/cmc/events/commercial_events/VectorCongress_2010/Diagnostics_3_Schmidgall_V7.pdf
- Sch11 Schmidgall, R.: "Diagnostic Communication – A Challenge for Network Analysis", 5th Symta Vision News Conference on Timing Analysis, October 2011, http://www.symtavision.com/downloads/Events-Info/NC5_Programm_2011.pdf
- Sch11-1 Schmidgall, R.: "Diagnostic Communication – Opportunities and Challenges ", 8th CTI Forum "Automotive Diagnostic Systems", March 2011,
- Schn08 Schnell, G., Wiedemann, B.: "Bussysteme in der Automatisierungs- und Prozesstechnik - Grundlagen, Systeme und Trends der industriellen Kommunikation", 7th edition, Vieweg+Teubner Verlag, Wiesbaden 2008, ISBN 978-3-8348-0425-9.

- shanno Shannon-Fano coding
http://en.wikipedia.org/wiki/Shannon-Fano_coding
(last access: 10.01.2011)
- Sha48 Shannon, C. E.: "A Mathematical Theory of Communication", Bell System Technical Journal. Short Hills N.J. vol. 27, July, October 1948, p. 379–423, 623–656. ISSN 0005-8580
- Siemens Siemens AG Transportation systems, Wittelsbacherplatz 2, 80333 München, Germany, www.siemens.com
- Sie SIEMENS AG Transportation systems, The Multiple-Unit Train for the European High-Speed Network, TH166-031091 199637 PA 12031.5
http://euroferroviarios.net/descargas/Empresas_Publicas/RENFE/Guias-Procedimientos/S-103.pdf
- Smart SMART Elektronik Development GmbH, Röttestr. 17, 70197 Stuttgart, Germany, www.smart-gmbh.de
- Softing Softing AG, Richard-Reitzner-Allee 6, 85540 Haar , Germany
www.softing.com
- Sou05 R.C. Sousa, I.L. Prejbeanu; Non-volatile magnetic random access memories (MRAM); C. R. Physique 6, (2005) 1013–1021
- Sto82 Storer, J., Szymanski, T.G.: "Data compression via textual substitution", Journal of the ACM, Vol. 29, No. 4, page 928-951, October 1982, ISSN:004-5411
- Symtavision Symtavision GmbH, Frankfurter Straße 3C, 38122 Braunschweig, Germany
www.symtavision.com
- Tan10 Tanenbaum, A. S., Wetherall, D. J.: "Computer networks", 5th edition, September 2010, Prentice Hall, ISBN 0132126958, Chapter 4.2 Multiple Access Protocols.
- Tan10-1 Tanenbaum, A. S., Wetherall, D. J.: "Computer networks", 5th edition, September 2010, Prentice Hall, ISBN 0132126958, Chapter 1 Introduction.
- TC1796 INFINEON: TC1796, 32-Bit Single-Chip Microcontroller, Data Sheet, V1.0, Infineon Technologies AG, April 2008, page 37 and Table 36 - Flash Parameters
- TC1197 Infineon; TriCore TC 1197 microcontroller data sheet, "TC1197 32-Bit Single-Chip Microcontroller", data sheet V1.1, Chapter 5.4.3 Flash Memory Parameters, Table 32: Flash Parameters, May 2009, www.infineon.com
- TC1197-1 Infineon; TriCore TC 1197 microcontroller data sheet, "TC1197 32-Bit Single-Chip Microcontroller", data sheet V1.1, Chapter 2.3.6.5 Program and Data Flash, sub-clause "Program Flash Features and Function; page 28, May 2009, www.infineon.com
- Tin95 Tindell, K., Burns, A. , Wellings, A.: "Calculating CAN Message Response Time", Control Engineering Practice, Heft 8, 1995, page 1163-1169
- TMS470 Texas Instruments: TMS470 R1A256, 16/32-Bit RISC Flash Microcontroller, SPNS100B – November 2004 – revised august 2006, Texas Instruments, 2006, page 31
- V850-Ex3 NEC: V850E/Dx3 - DJ3/DL3, 32-Bit Single-Chip Microcontroller, Document No. U20110EE1V0DS00, NEC Electronics, 2009, Table 9-2: Flash Memory Selfprogramming Characteristics
- Vda11 Verband der Automobilindustrie e. V. (VDA), Behrenstr. 35, 10117 Berlin, "Automobile production", <http://www.vda.de/en/zahlen/jahreszahlen/automobilproduktion> (last acces 05/2012)

- Vector VECTOR Informatik GmbH, Ingersheimer Str. 24, 70499 Stuttgart, Germany
www.vector.com
- wikipe Wikipedia – Interrupt. <http://en.wikipedia.org/wiki/Interrupt> (last access 05/2012)
- Zim10-1 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Chapter 3 KfZ-Bussysteme – Physical und Data Link Layer
- Zim10-2 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Chapter 8.4.2 Flashspeicher
- Zim10-3 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Chapter 8.4 Flashprogrammierung von Steuergeräten
- Zim10-4 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Chapter 5.2 Unified Diagnostic Services UDS nach ISO14229/15765-3
- Zim10-5 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Chapter 4 Transportprotokolle
- Zim10-6 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Chapter 3.1.7 Zeitverhalten von CAN-Systemen, Wahl der Botschaftspriorität
- Zim10-7 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Chapter 3.1.2 Bus-Topologie und Physical Layer and , Wahl der Botschaftspriorität
- Zim10-8 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Chapter 6 Anwendungen für Messen, Kalibrieren und Diagnose (ASAM AE MCD)
- Zim10-9 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Chapter 3.3 FlexRay
- Zim10-10 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Chapter 3.3 FlexRay – figure 3.3.5 logisches (Data Link Layer) FlexRay-Botschaftsformat
- Zim10-11 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Table 2.1.2 – Buszugriffsverfahren
- Zim10-12 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Chapter 3.3.2 – Bus Topologie und Physical Layer
- Zim10-13 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Chapter 2.1.2 –Topologie und Kopplung von Bussystemen
- Zim10-14 Zimmermann, W. Schmidgall, R. "Bussysteme in der Fahrzeugtechnik", 4th edition 2010, Vieweg+Teubner Verlag, ISBN 978-3-8348-0907-0, Chapter 6.6.3 – VEHICLE-INFO-SPEC: Fahrzeugzugang und Bustopologie
- Ziv77 Ziv, J. Lempel, A.: "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on information theory, Vol. IT-23, No. 3, page 337-343, May 1977

A Journal Paper – IEEE TVT

The journal paper below was submitted to the IEEE magazine “Transactions on Vehicular Technologies” at 27-03-2012.

The paper has the IEEE identification number VT-2012-00404.

Solutions and Strategies for Faster Embedded System Reprogramming

Will MRAM Technology Solve Reprogramming Problems for Embedded Systems?

Ralf Schmidgall, Dr. Ian Dear

Abstract— Software reprogramming is an important issue during an electronic control unit’s (ECU) life cycle. Software reprogramming takes place at ECU’s development, manufacturing and maintenance. The continuously increasing software size for embedded systems during the last years results in continuously increasing reprogramming times. This is especially applicable to the automotive industry but also in other business areas where cost pressure is high in production or in-field reprogramming. With the currently established Flash memory technology for embedded system’s microcontrollers a significant improvement of the reprogramming process might not be possible. The next evolutionary step in embedded memory technologies will be Magnetoresistive Random Access Memory (MRAM). With focus on reprogramming time the MRAM technology provides essential advantages and the reprogramming process execution time could be decreased significantly. This paper identifies the current problems associated with the embedded system’s software reprogramming process and suggests some new methods for reprogramming software using the newly proposed MRAM technology.

Index Terms—Microcontroller, Software Reprogramming, Flash Memory, MRAM,

I. INTRODUCTION

Today, microcontrollers are no longer used only for simple control and regulation purposes. Owing to the enormous technological progress in this area high performance microcontrollers are available today to solve highly complex control and regulation assignments. Hence more and more functionality is implemented on these microcontrollers which have resulted in the continuously increasing software sizes, the end of which is not foreseeable [20]. In many applications microcontrollers have reached the mega byte (MB) boundary for on-board memory. For example, electronic control units (ECU), used in the automotive industry, provide memory resources of several MB to solve complex control

assignments like engine control or any kind of driver assistance systems.

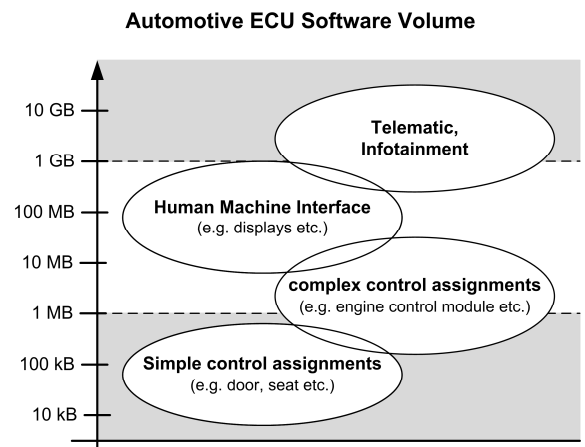


Figure 1: Automotive ECU software volume

ECUs used for human machine interfaces (e.g. displays, instrument cluster etc.) have increased up to several 100 MB of Flash memory space. For telematic and in-vehicle infotainment (IVI) systems memory has reached the GB boundary. Typically these systems are based not on Flash memory but on hard disks. Fig. 1 depicts an overview of typical software volumes within the automotive area.

A Software reprogramming within ECU’s life cycle

Software reprogramming is an important issue within ECU’s life cycle particularly for the automotive industry. During the vehicle’s development phase an ECU is reprogrammed several times to replace the previous software with the current release.

Today several manufacturing strategies exist within the automotive industry. Mainly the ECU’s are delivered fully programmed by the ECU manufacturer to the vehicle manufacturer (OEM). Another method is to deliver the ECU partly programmed or without software and programme the final release within OEM’s vehicle assembly line.

Software reprogramming is also an important repair method for OEMs in the aftersales service. If customer's complaints could be solved by a new software release reprogramming is the preferred repair method.

B. Introduction of Flash memory technology to reduce production costs

Particularly in the automotive industry but also in other industries an enormous cost pressure prevails. With the introduction of the Flash memory technology it became possible to correct an ECU software error in the field without the necessity to replace the node physically. ECU's software is simply reprogrammed via the microcontroller's communication interface. By this approach it was possible to reduce aftersales costs significantly, because neither costs for a new node or worker costs any longer occur. Software reprogramming's benefit is particularly given in the case for ECUs that are difficult to physically access like a vehicle engine controller or gear box controller. For commercial vehicles long idle times in a garage or repair shop are not accepted by transport companies. In other business areas it was no longer necessary to exchange a component on complex machines to eliminate a software error.

A second important benefit of Flash memory technology compared to the thitherto available ROM mask memory is the possibility to reduce ECU's hardware/software variants and therefore logistic costs. If special functionality depends only on software, then the same hardware can be re-used. For example, a 4 door vehicle could utilise the same hardware for each door and the functionality differentiation between front and rear door (e.g. mirror controlling) would be done by software. Logistic costs (i.e. stock control and storage complexity) have been reduced as well as decreased complexity within the OEM's assembly line because only one part has to be selected for vehicle manufacturing. Additionally increased part volume can reduce the purchase price. There is also a cost benefit for field repair and maintenance costs.

C. The cost of long programming times for Flash memories.

A consequence of the increasing ECU software size in embedded systems is the increasing programming time for a software update. This might lead to potential economical disadvantages.

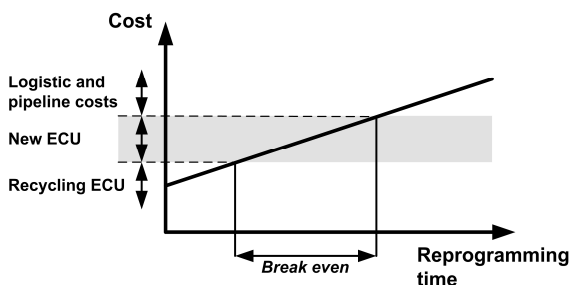


Figure 2: Cost / Time relation

The increasing reprogramming times significantly reduce the cost advantage of reprogramming software. Within the production process the time given to finalize an assembly step can be exceeded resulting in a cost penalty. In the maintenance and service industry there are similar time limitations and cost penalties. Additionally the reprogramming procedure will require specific off-board

equipment (e.g. diagnostic test system, power supply etc.) This complicates the logistics and thus costs within the field test and repair centre e.g. the need to purchase multiple off-board equipment or increased service times in busy periods. As depicted in Fig. 2 the break even point of software reprogramming vs. ECU replacement is moved, if reprogramming time increases. As a result the cost advantage of reprogramming an ECU disappears.

Today replacing and recycling ECUs is not economic but if reprogramming costs continue to increase replacing and recycling ECUs will be an alternative approach.

The aspects above depict that it is necessary to find methods and strategies to reduce reprogramming times significantly to guarantee the economic advantage of onboard software reprogramming. There are two important questions:

- (1) Is it possible to reduce programming time significantly for the Flash memory based systems?
- (2) Could the magnetoresistive random access memory (MRAM) technology solve the reprogramming time and thus cost issues faced by the automotive industry?

II. SOFTWARE REPROGRAMMING PROCESS FOR FLASH MEMORY TECHNOLOGY BASED EMBEDDED SYSTEMS

A. Flashloader and application software

As depicted in Fig. 3 the microcontroller implements two independent software components: The application software and the flashloader. The application software implements the functionality of the ECU. A flashloader component handles the complete reprogramming process if the application software is to be reprogrammed. The flashloader communicates via the normal communication interface of the microcontroller and exchanges data with an off-board programming device (e.g. diagnostic tester).

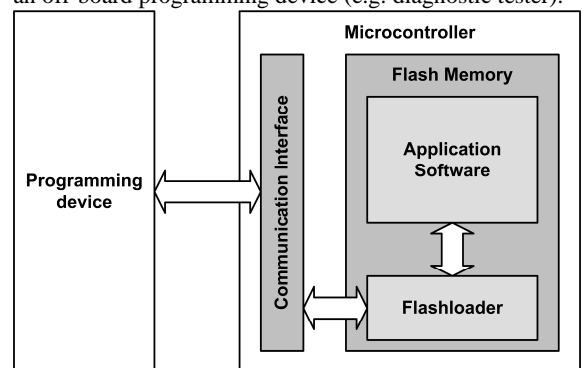


Figure 3: Components overview

B. Reprogramming Sequence

Fig. 4 illustrates the Flash memory programming sequence. The first step in programming a device is to identify the ECU e.g. the microcontroller type, software version, and associated hardware. Step 2 the programming device needs to authenticate itself to the flashloader of the microcontroller. This is achieved by implementing special authentication methods (e.g. seed & key algorithm etc.) the reprogramming sequence could be aborted if authentication fails. Normally this is a first part of a (more or less powerful) security concept to prevent unauthorised software manipulation.

After successful authentication the Flash memory can be erased (step 3). After a successful erasing process the new data can be transmitted to the microcontroller and be programmed into the Flash memory (step 4). This opera-

tion is the most time consuming sequence and depends on the total amount of data to be programmed. The sequence is finalised by a verification of the programmed data (i.e. application software). Typically, methods like cyclic redundancy check (CRC) are used.

The programming of an ECU is more complex when it is embedded within a vehicle network and when it has no direct access between the programming device and the ECU. Within the automotive area reprogramming is established via the communication interfaces CAN [10], [11], FlexRay [9], LIN [8] etc. In some case a multiple network interface is used.

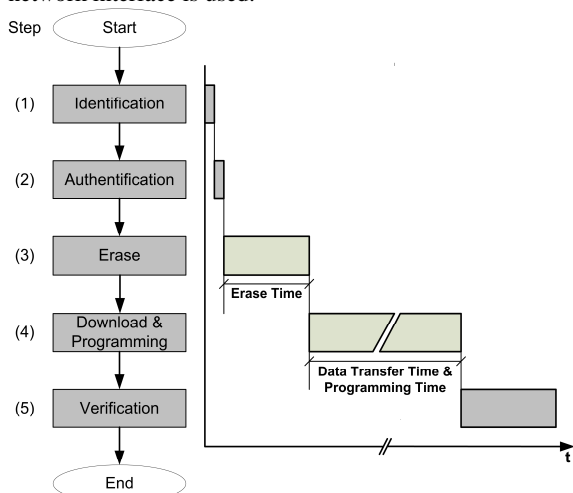


Figure 4: Reprogramming sequence overview.

Microcontrollers can provide other interfaces, e.g. the debugging interface JTAG (Joint Test Action Group [12]). It is also possible to reprogram via this interface. Typically these interfaces are not connected to ECU's communication connector within a vehicle for some reasons:

Assume that the position of an ECU allows neither a physically access for a worker nor is it possible to open the chassis e.g. a vehicle's gear box within the oil sump is here a good example. To get access to the interface JTAG should be connected to the normal ECU's vehicle communication connector. This results in a larger connector (more pins), an additional cable or a second network in parallel. A complex system consists of several ECUs based on different microcontrollers and is supplied by different manufacturers (e.g. vehicle with up to 70 ECUs) the external test system which controls the reprogramming process has to implement all the individual communication protocols available. This complexity/effort is enormous and only practicable for software programming within manufacturer's ECU assembling lines and not for software reprogramming in the field. A unique reprogramming sequence with standardized protocols must be used to reduce the effort and thus cost.

The reprogramming sequence as depicted in Fig. 4 is independent of the different communication protocols. Within the automotive industry software reprogramming is part of the vehicle diagnostics and is based on the UDS protocol (Unified Diagnostic Services) which is standardised in the specification ISO 14229 [15] and ISO 15765-3 [16]. Depending to the available vehicle bus systems the reprogramming sequence is executable via CAN, LIN, FlexRay or K-Line.

The reprogramming sequence could be mapped to other (non-diagnostic) communication protocols like CCP

(CAN Calibration Protocol) or XCP (eXtended Calibration Protocol) [17], [18]. However, these protocols are not available on every communication interface. Werner Zimmermann and Ralf Schmidgall [7] give a detailed overview of the required components to execute a reprogramming process.

III. REPROGRAMMING PROCESS TIME REDUCTION OPTIMIZATION APPROACHES

Due to the reprogramming sequence according to Fig. 4, to reduce the total software reprogramming time significantly two approaches are possible: (1) Accelerate data transfer and (2) reduce to be transferred data size. The other stages within that sequence are hardware dependant and based on technology used.

A. Data transfer acceleration

1) Data transfer acceleration on ISO/OSI layer 2 data link protocols

A simple method to speed up data transfer is to speed up the underlying vehicle bus systems. But the maximum bandwidth of the most common used automotive bus systems is limited. Fig. 5 depicts an overview of the bandwidths for the most common automotive bus systems.

Most common automotive bus systems' bandwidth

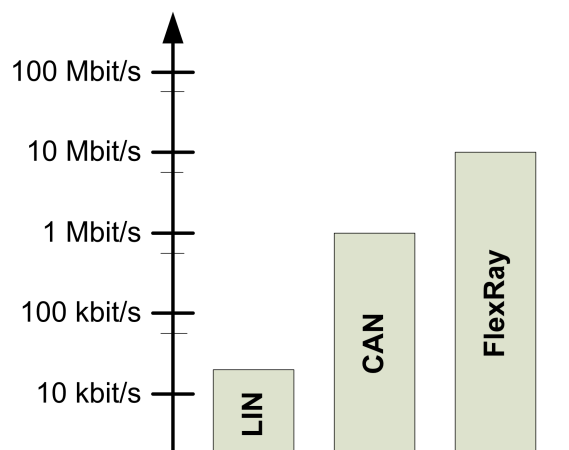


Figure 5: Most common automotive bus system's bandwidth

The CAN bus system is limited to 1 Mbit/s by specification ISO 11898. Also some other limitations have to be taken into account: If bandwidth is increase the maximum cable length is reduced. The split of the bus system results in additional, more expensive and more complex gateways. Also increase shielding is necessary because of electromagnetic compatibility (EMC).

For the FlexRay bus systems the maximum specified system's bandwidth (max 10 Mbit/s) is not the only limiting aspect. The data transfer rate for the time triggered FlexRay bus system is mainly influenced by the fix defined communication schedule and the corresponding communication slot arrangement. If the communication slot is not allocated for a FlexRay flashloader it is not usable for data transmission for a reprogramming process.

Automotive Ethernet (100 Mbit/s) might be an approach to speed up data transfer for an offboard diagnostic test system via a vehicles connector interface (VCI). The

challenge is now the distribution of the received data via the vehicles network. Therefore the new standardisation co-operation OPEN Alliance (**O**ne **P**air **E**thernet) was founded to “encourage wide scale adoption of Ethernet-based, single pair unshielded networks as the standard in automotive applications” [13].

Nevertheless, the slowest bus system section on the communication link (especially within a heterogeneous network with several different bus systems) will dictate the possible bandwidth and therefore the communication performance.

2) Data transfer acceleration on ISO/OSI layers 3-5 - transport and diagnostic layers

Software reprogramming within the automotive industry is done via diagnostic communication and based on the standardized diagnostic protocol “Unified Diagnostic Services (UDS) according to the specification ISO 14229. Depending on the underlying bus systems a standardized transport layer protocol (e.g. for CAN: ISO 15765-2, for FlexRay: ISO 10681-2 etc.) is in use for data segmentation and re-assembling for large data frames. Of course, all the different communication protocol stacks could be optimized and configured to eliminate protocol specific delays e.g. minimum separation time (ST_{min}) for CAN communication, but if all protocols have been optimized the limiting factor is the underlying bus system bandwidth as shown in Fig. 5.

3) Summary

Hence, the possibility of speeding up bus systems and their corresponding communication protocol stacks is given and possible results are formidable, but it will be not enough to solve the challenges of increasing software. An important impact factor is the time limitation for a reprogramming process e.g. as given in a vehicle assembly line in a plant (assembly line clock).

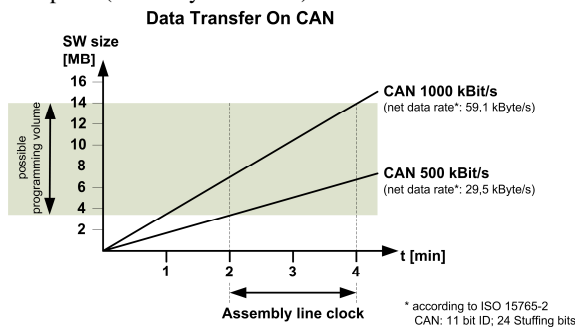


Figure 6: Data transfer acceleration limits on CAN

Fig. 6 depicts an overview of possible data volumes which can be transferred via CAN within the given time limits. Due to the data volumes as highlighted in Fig. 1 and the maximum bandwidth of the currently established automotive bus systems as shown detailed in Fig. 5 the data transfer acceleration approach is not sufficient to solve the problem of increasing programming times for all vehicle domains.

B. Reduce data size

1) Data size reduction by software partitioning

A powerful method to reduce the transferred data size is the *partitioning of the ECU's application software* into several sub parts. Typically the real application could be separated from the data set (e.g. characteristic curves for mathematical algorithm processing etc.). In case of

software reprogramming only the affected partition has then to be transferred. Fig. 7 illustrates the separation into different software partitions where only partition B is affected and has to be reprogrammed. However, an additional logistic overhead is introduced: the partition's software compatibility has to be managed.

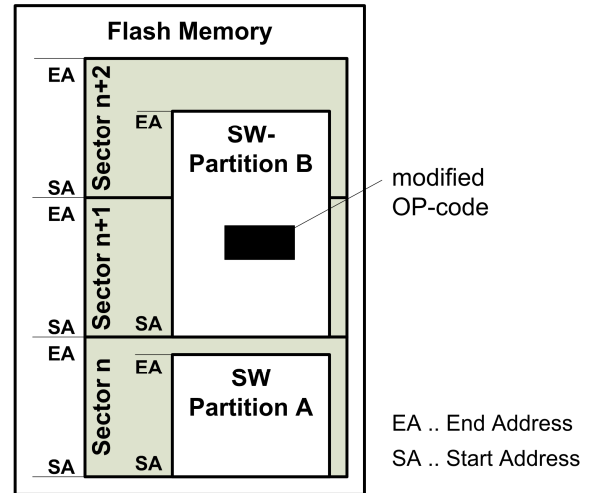


Figure 7: Mapping of SW partitions to physical memory sectors

2) Data size reduction by data compression

Data compression is an alternative standard approach to reduce transferred data size. Compressed data transfer is an established method. The reduction in data transfer time depends on the compression ratio of the used algorithm. Unfortunately not all known compression algorithms are usable within embedded systems. First of all, only lossless compression methods can be used. Also dictionary based algorithms are not possible due to the resource limitations of RAM within a microcontroller. However, substitution strategy based compression algorithms (e.g. LZSS [14]) provide good results for software with high redundancies like characteristic curves for regulation systems etc. As the compression ratios proportional to the redundancy with the actual data the data compression this is not a generic approach to solve the problem of increasing reprogramming times.

3) Data size reduction by differential file transfer

One reason for reprogramming embedded software is bug fixing. In most cases embedded software does not change completely when fixing a bug (e.g. changing a value of a constant or some parameters within a characteristic curve etc.). As a percentage of the total volume of an application the source code modifications required and the resulting OP-code changes, required for bug fixing is often very small. Typical errors in the source code like wrong exit conditions in loops or wrong statements for a comparison are only one character. Changes in characteristic curves implemented as arrays covers only a few bytes. Thus an assumption that 80% of bug fixings result in less than 1 kB OP-code changes and 20% in more than 1 kB is safe and realistic figure. As a result of this assumption only a few bytes within a memory sector/partition needs to be changed. Fig. 7 depicts the small OP-code difference within a software partition.

The today's state of the art and established Flash memory technology provides the technical disadvantage that a byte-wise overwriting of a Flash memory cell is not

possible. Due to that technical fact, the smallest physical memory partition (page, sector etc.) must be previously erased before it can be re-programmed. Hence, erasing the complete physical sector is necessary no matter if a complete memory section or only a few bytes have changed. A temporary storage of these page or sector data (data mirror) requires large RAM resources with at least the size of that page or sector (e.g. INFINEON TC1797: 256 kB). Because of the typically not available RAM resources, thus the data for reprogramming the complete physical section always has to be transferred and programmed. The powerful approach of reprogramming software by differential file is not usable for currently established Flash memory technology.

4) Conclusion

Based on existing Flash based memory architectures current approaches and suggested variations to existing approaches to reduce data transfer time and thus constrain future trends in reprogramming time for vehicle based embedded systems will not solve the initial problem. It has been shown that only relatively small improvements can be achieved; a radically new approach is needed.

IV. MRAM TECHNOLOGY

A real quantum transition will be possible if the currently available and established Flash memory technology is replaced by the new proposed MRAM technology (Magnetoresistive Random Access Memory) in microcontrollers [e.g. Infineon's TriCore family, Freescale's HCx family, Texas Instruments' TMS or Hercules family etc.]. Some disadvantages of Flash memory caused by the inherent technology can be eliminated by the employment of possible MRAM technologies.

TABLE 1: COMPARISON OF EXPECTED MRAM FEATURES WITH OTHER MEMORY TECHNOLOGIES [6]

Comparison of Expected MRAM Features with other Memory technologies [6]					
	MRAM	SRAM	DRAM	Flash	FeRAM
Read Speed	Fast	Fastest	Medium	Fast	Fast
Write Speed	Fast	Fastest	Medium	Low	Medium
Non-Volatile	Yes	No	No	Yes	Yes
Low Voltage	Yes	Yes	Limited	Limited	Limited
Complexity	Medium	Low	Medium	Medium	Medium

In contrast to currently established memory technologies, MRAM semiconductors store the information not using electrical, but by magnetic load elements. The effect is based on the fact that certain materials change their electrical resistance if they are influenced by magnetic fields [4], [19], [21].

Effective fundamental research activities to the magnetoresistive started in 1989. At that time IBM scientists made a set of key discoveries. In the year 2000 IBM and Infineon started a joint MRAM development program. In 2005 Renesas presented a 1MBit memory for a 100MHz clock frequency [5].

In a MRAM cell the information zero (0) and one (1) are represented by the orientation of magnetic fields and is based on the Magnetic Tunnel Junction (MTJ) effect [21]. A MTJ semiconductor has a three-layer structure. It consists of two magnetic layers and an insulation layer. One of the magnetic layers has a fixed orientation (fixed magnetic layer). The other magnetic layer can change its magnetic polarization (floating magnetic layer). It is aligned either in the same orientation as the fixed layer (parallel magnetic orientation) or in the opposite (opposite magnetic orientation). Although not shown in Fig. 8, a bit

line and digit line are located above and below the MTJ. The electrical resistance of the memory cell changes depending to the magnetic orientation of the floating magnetic layer. According to the electrical resistance a high or low current could occur. A current switch converts the binary information low current and high current to voltage levels (low current = 0_{bin} ; high current = 1_{bin}).

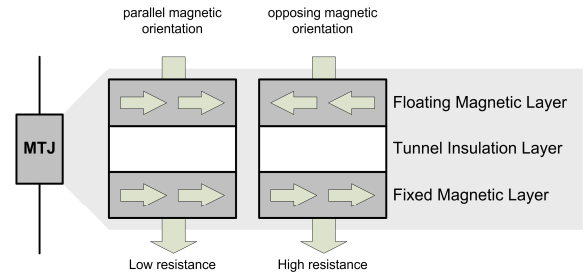


Figure 8 - MRAM

The MRAM technology does not need any electrical current in order to hold the stored information. Once the magnetic adjustment is made the variable magnetic layer remains static, i.e. no further current is required.

MRAM adopts the advantages of several memory technologies available today. Similar to Flash memory or EEPROM (Electrical Erasable and Programmable Read Only Memory) a non-volatile data retention takes place, i.e. program code and data are sustained without power supply. MRAM reduces the power consumption because the refresh pulses as required for DRAM are not longer necessary. The data access is very fast (cf. SRAM) and MRAM cells are small which results in a high device integration level.

V. REPROGRAMMING PROCESS OPTIMISATION USING MRAM TECHNOLOGY

As depict in Fig. 4 the steps erasing memory (step 3) and download and reprogramming (step 4) of a Flash memory based system have a significant impact on total reprogramming time. MRAM technology can make significant improvements in these areas.

A. Reduce Memory Erase Time

As mentioned above, normally Flash memory technology does not allow the overwriting of programmed memory cells without prior erasing memory partitions or sectors. It is currently not possible to erase a single memory cell. MRAM technology allows overwriting of individual programmed memory cells without prior erasing of the cell. Therefore step 3 of the reprogramming process is no longer required.

TABLE 2: MICROCONTROLLER'S ERASE TIME FOR FLASH MEMORY

TriCore TC 1797 (INFINEON) [1]			
	min	typ.	max
Program Flash Erase Time per 256-kByte Sector	-	-	5 s
HCS 12X (FREESCALE) [2]			
	min	typ.	max
P-Flash sector erase time 1024 Byte	-	20	21 ms
Normalize to 256 kByte	-	5.1	5.4 s

Table 2 shows the normalized erase time values for 256 kB on-chip Flash memory of two different microcontrollers. Based on this data given by the manufacturer's data sheets [1], [2] the predicted total erase time for a 256 kB sector of on-chip Flash memory is up to 5 seconds. This

time could be saved potentially in case of using MRAM. The benefit is still higher in case of the Op-code modifications are not only located to a single physical memory section.

It is not possible to make a precise generic statement for the saved erase time because this value depends on several parameters e.g. memory technology, oscillator frequency, and the size of memory that is to be erased. As depict in Fig. 1, within the automotive industry ECUs exist with a total amount of Flash memory up to several 100 MB.

B) Reprogramming by differential File

For a detailed analysis of reprogramming process acceleration it is helpful to divide step 4 of Fig. 4 into the two sub-sequences “data transfer” and “physical reprogramming”.

The MRAM technology allows read/write access basically for each single byte (alignment has to be taken into account). Hence, MRAM allows an optimisation to the reprogramming process were only the real differences of the old and new compiler/linker output file (OP-code) have to be transferred and reprogrammed (refer to Fig. 9). This results in significant time reductions for the data transfer and the corresponding physical programming process.

1) MRAM vs. Flash memory

In contrast, the differential file approach for a Flash memory technology based system requires large RAM resources to mirror the current memory sector content (refer to III-2c) and therefore typically the complete sector content will be transferred.

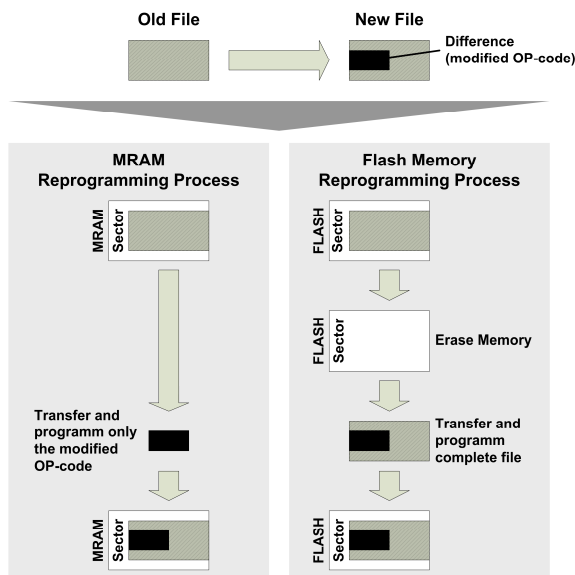


Figure 9: Usage of a differential file for physical memory sector reprogramming

In table 3 a comparison of both approaches (MRAM with differential file transfer and Flash memory with complete file transfer) is given. The data volume to be transferred and reprogrammed is the main influencing factor for the total reprogramming time. Based on the Flash memory sector sizes of Infineon’s TriCore TC1767 [3] microcontroller we assume that the modified OP-code is less than 1kByte (refer to III-2c) within on memory section. The corresponding data transfer times on a CAN bus system with 500 kbit/s bandwidth are calculated according to formula 1 and the given assumptions. To

simplify the model neither upper communication protocols (e.g. transport protocol for CAN according to ISO 15765-2 etc.) nor communication delays (inter-frame times between two CAN-PDUs) have been taken into account.

Fehler! Es ist nicht möglich, durch die Bearbeitung von Feldfunktionen Objekte zu erstellen. Formula 1 – data transfer time

Assumption:
 Payload for CAN 8 Byte / frame
 Approximate frame length: 123 bit
 BitRate: 500 kbit/s

We assume also that the write speed to MRAM is equal to existing Flash memories (a safe assumption as shown by predictions in table 1).

Fehler! Es ist nicht möglich, durch die Bearbeitung von Feldfunktionen Objekte zu erstellen. Formula 2 – programming time

Assumption:
 Programming rate 50 kByte/s [1,3]

TABLE 3: DATA VOLUME AND TRANSFER TIME FOR INFINEON’S TRICORE 1797 FLASH MEMORY SECTOR SIZES

TriCore TC 1797 [3] Memory sector size	Data volume to transfer and programm	Transfer time	Programming time	Process time
Flash Memory Technology				
16 kByte	16 kByte	0.50 s	0.32 s	0.82 s
128 kByte	128 kByte	4.03 s	2.56 s	6.59 s
256 kByte	256 kByte	8.06 s	5.12 s	13.18 s
MRAM Technology				
16 kByte	< 1 kByte	< 0.031 s	< 0.02 s	< 0.051s
128 kByte	< 1 kByte	< 0.031 s	< 0.02 s	< 0.051s
256 kByte	< 1 kByte	< 0.031 s	< 0.02 s	< 0.051s

Table 3 illustrates the power of the differential file approach. Especially for large physical memory sections the benefit of reduced transfer time and reduced programming time is quite evident. Upper layer communication protocols will reduce the data transfer rate in addition and results in increasing transfer times. Of course, the data transfer time depends fundamentally on the underlying bus systems and the network architecture. A slow bus system with small bandwidth will increase the data transfer time compared to a faster bus system. But even for small bandwidth bus systems data reduction has a significant impact to the data transfer time and the total reprogramming time.

2) Comparison to data size reduction approaches

The benefit of the MRAM based differential file approach for software reprogramming is also quite evident if the method is compared to other typical data reduction methods. Table 4 shows a comparison depicting saving due to typical data reduction methods and differential file approaches based on formula 1. The transport protocol overhead or differential file overhead has not been taken into account. Table 4 has been generated by making the following assumptions based on typical data for an ECU that process complex control assignments, e.g. driver assistance systems (refer to Fig. 1):

Assumption:
 File size: 32 MByte

Compression ratio:	75%
Modified OP code size:	1 kByte
CAN Payload:	8 Byte / frame
Approximate frame length:	123 bit
CAN Baud rate:	125 kbit/s, 500 kBit/s 1 Mbit/s

The table 4 shows that if only the differences of both files will be transferred the data transfer time is significantly reduced compared to conventional data size reduction methods e.g. partitioning and compression. According to the reprogramming sequence in Fig. 4 the benefit of step 4 (download & programming) is visible. For Flash memory the step 3 (erase memory) still has to be processed. Within the given example the interpolation erase time for 32 MB is up to 640 s or 240 s for 12 MB (refer to table 1). For an MRAM based system this time is not relevant. The step 5 (verification) is necessary for both memory approaches. At least a CRC must be calculated to verify the correct programming and the consistency of the new software. Due to the equal read access speed of both memory technologies (refer to table 1) this step provides no differences. The required execution time of step 5 depends on the CRC calculation algorithm, microcontroller's clock frequency etc. and can take up to several minutes.

Of course, the model is simplified and is not complete but it depicts that an approach of differential file transfer based on MRAM technology provides significant potential saving.

TABLE 4: DATA TRANSFER TIME VIA CAN

Data reduction methods (modified Op code = 1kByte)					
Description	File size	Data volume to transfer	Data transfer time on CAN		
			125 kbit/s	500 kbit/s	1 Mbit/s
Original File	32 MByte	32 MByte	4127,2 s	1031,8 s	515,9 s
Compressed (-25%)	24 MByte	24 MByte	3095,4 s	773,8 s	386,9 s
Partitioning (2 parts)	16 MByte	16 MByte	2063,6 s	515,9 s	257,9 s
Partitioning (2 parts) + Compression (-25%)	12 MByte	12 MByte	1547,7 s	386,9 s	193,5 s
Differential file (modified OP code = 1kByte)					
Memory Type	Sector Size	Data volume to transfer	Data transfer time on CAN		
			125 kbit/s	500 kbit/s	1 Mbit/s
Flash	256 kByte	256 kByte	32,244 s	8,061 s	4,030 s
Flash	128 kByte	128 kByte	16,122 s	4,030 s	2,015 s
Flash	16 kByte	16 kByte	2,015 s	0,504 s	0,252 s
MRAM	256 kByte	1 kByte	0,126 s	0,031 s	0,016 s
MRAM	128 kByte	1 kByte	0,126 s	0,031 s	0,016 s
MRAM	16 kByte	1 kByte	0,126 s	0,031 s	0,016 s

3) Restrictions

Of course, the differential file transfer approach provides some restrictions and has some additional requirements to the software development process.

In contrast to the file oriented software storage PC world with its virtual addresses, an embedded system's microcontroller works physical address oriented. A microcontroller provides neither a memory managing system nor that much memory to squander memory space. This is why during the embedded software generation process all source code elements (e.g. in C-language: functions, arrays etc.) are linked consecutively without any larger gaps within the address space. Consequently, if a routine expands all other compiled elements will change the allocation address. In that case the differences between a previous file and a new compiled and linked file will be quite high whereas the source code changes are only a few lines of code. To be able to reduce the differences of the embedded software files it is necessary to allocate all the software parts always on the same position (address). This requires a fixed linking concept to guarantee that the

smallest possible difference of both files can be calculated. However, a link process with fix addresses can be implemented by different approaches. A fix position for at least each source code module (e.g. c-file, object-file etc.) must be configured within the linker command file. Best results provide the fix allocation on source code function level. Here each function or array etc. is allocated on a fix position.

The disadvantage is the necessity to have address gaps (empty space) between the single linking objects to prevent the system from overwriting other allocated code objects in case of further upgrades of another code object. Hence, the commercial relation of higher costs for a larger memory vs. reprogramming time and cost reduction has to be taken into account.

Especially within the automotive industry a stringent version and compatibility control management is required because software on a car is only reprogrammed if it is in a repair shop. Because of the large service intervals of modern vehicles it might be possible that several software versions are in between the current vehicle software and the current OEM software.

To guarantee equally high process' safeness and security of the MRAM based on differential file approach compared to the established Flash memory programming process these basic issues have to be taken into account.

VI. CONCLUSION AND OUTLOOK

The paper has discussed the rapidly approaching limitation of Flash technologies in embedded vehicle systems for in-system reprogramming. The main advantages of MRAM vs. Flash memory technology with a focus on reprogramming have been presented. The benefits of new programming approaches have been discussed and the possibilities of bit is the byte-wise access of MRAM memories with the possibility to overwriting data without an initial memory erase phase highlighted. Byte-wise access allows software updates by transferring and overwriting only differences between the old and new software. Due to the reduced amount of data to transfer, the data transfer time and the physical programming time significant time savings can be made. Thus the potential cost savings of the new technologies could solve the rapidly approaching technological limitation of Flash memories in modern complex embedded vehicle systems.

The paper has not quantified the possible overheads associated with differential file programming, however, neither has it detailed such factors as the increased risk of process interruption as programming time increases for conventional Flash technologies.

For the resulting cost aspects two different scenarios occur: If high speed communication bus systems are available data transfer time could reduced. A total process time reduction provides cost advantages for production and within some business areas (e.g. vehicle industry etc.) in service/after sales activities, too. On the other hand differential file transfer would make it possible to use low cost small bandwidth bus systems but maintain the current data transfer times.

In conclusion the presented study shows that the problem of increasing costs because of increasing software sizes and resulting reprogramming times could be partly solved when MRAM becomes commercially available.

This paper has concentrated on the problem facing all business areas where software programming time provides

a rapidly increasing potential cost. It has proposed that the new MRAM technologies will potentially resolve this issue. However, there are other significant benefits that the technology can offer related to the types of bus architecture needed.

REFERENCES

- [1] Infineon; TriCore 1797 Data Sheet “TC1797_DS_v1.1”; Table 32 Flash Parameters; 2009; available: www.infineon.com
- [2] Freescale Semiconductors; “MC9S12XEP100 Reference Manual V1.18”; Chapter 2.3.6.5 Program and data Flash; 2008; available: www.freescale.com
- [3] Infineon; TriCore 1797 Data Sheet “TC1797_DS_v1.1”; Chapter 2.3.6.5 Program and Data Flash; 2009; available: www.infineon.com
- [4] A. Hammerl, H. Bag; “MRAM – Magnetische Speicher”; 2003; Vienna University of Technology;
- [5] RENESAS ; Renesas Edge 2005-Vol 9, page 12, 2005 available: www.renesas.com
- [6] Freescale Semiconductors; MRAM fact sheet, Document number MRAMTECHFS Rev.6, 2007. available: www.freescale.com
- [7] W. Zimmermann, R. Schmidgall, „Bussysteme in der Fahrzeugtechnik“, 4th edition 2011, ViewegTeubner Verlag
- [8] W. Zimmermann, R. Schmidgall, „Bussysteme in der Fahrzeugtechnik“, 4th edition 2011, ViewegTeubner Verlag
- [9] FlexRay Communication system – Protocol Specification, Version 2.1, 2005, available: www.flexray.com
- [10] ISO 11898-1, Road Vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signaling, 2003, available: www.iso.org
- [11] ISO 11898-2, Road Vehicles - Controller area network (CAN) - Part 2: High speed medium access unit, 2003, available: www.iso.org.
- [12] IEEE Standard 1149.1-2001, IEEE standard test access port and boundary-scan architecture, doi:10.1109/IEEESTD.2001.92950, available: www.ieee.org.
- [13] OPEN ALLIANCE SIG, www.opensig.org
- [14] J. A. Storer and T. G. Szymanski, Data compression via textual substitution, Journal of the ACM, Volume 29, Number 4, Pages 928-951, ISSN:0004-5411, October 1982
- [15] ISO 14229-1, Road vehicles - Unified diagnostic services (UDS) - Specification and requirements, 2006-12
- [16] ISO 15765-3:2004 – Diagnostics on Controller Area Networks (CAN) - Part 3: Implementation of unified diagnostic services (UDS on CAN) - chapter 10
- [17] ASAM MCD-1.CCP V2.1.0, CAN Calibration Protocol, available: www.asam.net
- [18] ASAM MCD-1.XCP, “The Universal Measurement and Calibration Protocol Family”, V1.1.0, available: www.asam.net
- [19] R.C. Sousa, I.L. Prejbeanu, “Non-volatile magnetic random access memories (MRAM)”, C. R. Physique 6, p1013–1021, 2005
- [20] J. Dannenberg, J. Burgard, “Car Innovation 2015 – A comprehensive study on innovation in the automotive industry”, Oliver Wyman, 2007; available: www.oliverwyman.com
- [21] R.Gross, Magnetic Tunnel Junction based on Half-MetallicOxids, Nanoscale Devices, Fundamentals and Applications, p.49, ISBN 1-4020-5106-9, 2006, Springer Verlag