

XCP – Das Standardprotokoll für die Steuergeräte-Entwicklung

Grundlagen und Einsatzgebiete

Andreas Patzer | Rainer Zaiser

Andreas Patzer | Rainer Zaiser

XCP – Das Standardprotokoll für die Steuergeräte-Entwicklung



Stand März 2013 | Nachdruck nur mit Genehmigung der
Vector Informatik GmbH, Ingersheimer Str. 24, 70499 Stuttgart

© 2013 by Vector Informatik GmbH. Alle Rechte vorbehalten. Das Buch ist nur zum persönlichen Gebrauch bestimmt, nicht aber für technischen oder kommerziellen Gebrauch. Es darf nicht als Grundlage von Verträgen gleichwelcher Art dienen. Alle Informationen in diesem Buch sind mit größtmöglicher Sorgfalt zusammengestellt worden, dennoch übernimmt Vector Informatik keinerlei Gewährleistung oder Garantie für die Richtigkeit der enthaltenen Informationen. Die Haftung von Vector Informatik ist außer für Vorsatz und grobe Fahrlässigkeit ausgeschlossen, soweit nicht gesetzlich zwingend gehaftet wird.

Informationen in diesem Buch können urheberrechtlich und/oder patentrechtlich geschützt sein. Produktnamen von Software, Hardware und andere Produktnamen, die in diesem Buch benutzt werden, können eingetragene Marken sein oder anderweitig markenrechtlich geschützt sein, unabhängig davon, ob diese als eingetragene Marken gekennzeichnet sind oder nicht.

XCP

Das Standardprotokoll für die Steuergeräte-Entwicklung

Grundlagen und Einsatzgebiete

Dipl.-Ing. Andreas Patzer, Dipl.-Ing. Rainer Zaiser
Vector Informatik GmbH

Inhaltsverzeichnis

Einführung	7
1 Grundlagen des XCP-Protokolls	13
1.1 XCP-Protokollschicht	19
1.1.1 Identification Field	21
1.1.2 Timestamp	21
1.1.3 Data Field	22
1.2 Austausch von CTOs	22
1.2.1 XCP-Kommandostruktur	22
1.2.2 CMD	25
1.2.3 RES	28
1.2.4 ERR	28
1.2.5 EV	29
1.2.6 SERV	29
1.2.7 Verstellen von Parametern im Slave	29
1.3 Austausch von DTOs – synchroner Datenaustausch	32
1.3.1 Messverfahren: Polling versus DAQ	33
1.3.2 Das DAQ-Messverfahren	34
1.3.3 Das STIM-Verstellverfahren	41
1.3.4 XCP-Paket-Adressierung für DAQ und STIM	42
1.3.5 Bypassing = DAQ + STIM	44
1.4 XCP-Transportschichten	45
1.4.1 CAN	45
1.4.2 FlexRay	48
1.4.3 Ethernet	51
1.4.4 SxI	53
1.4.5 USB	53
1.4.6 LIN	53
1.5 XCP-Services	54
1.5.1 Speicherseitenumschaltung	54
1.5.2 Sicherung von Speicherseiten – Data Page Freezing	56
1.5.3 Flash-Programmierung	56
1.5.4 Automatische Erkennung des Slaves	58
1.5.5 Blocktransfer-Modus für Upload, Download und Flashen	59
1.5.6 Kaltstartmessung (Start der Messung bei Power-On)	60
1.5.7 Schutzmechanismen mit XCP	61

2 Steuergeräte-Beschreibungsdatei A2L

63

2.1 Aufbau einer A2L-Datei für einen XCP-Slave

66

2.2 Manuelle Erstellung einer A2L-Datei

67

2.3 A2L-Inhalt versus ECU-Implementierung

68

3 Kalibrierkonzepte

71

3.1 Parameter im Flash

72

3.2 Parameter im RAM

74

3.3 Flash-Overlay

76

3.4 Dynamic Flash-Overlay Allocation

77

3.5 RAM-Pointer-basiertes Kalibrierkonzept nach AUTOSAR

78

3.5.1 Single-Pointer-Konzept

78

3.5.2 Double-Pointer-Konzept

80

3.6 Flash-Pointer-basiertes Kalibrierkonzept

81

4 Einsatzgebiete für XCP

83

4.1 MIL: Model in the Loop

85

4.2 SIL: Software in the Loop

86

4.3 HIL: Hardware in the Loop

87

4.4 RCP: Rapid Control Prototyping

89

4.5 Bypassing

90

4.6 Verkürzung der Iterationszyklen mit virtuellen Steuergeräten

93

5 Beispiel einer XCP-Implementierung

97

5.1 Beschreibung der Funktionen

100

5.2 Parametrierung des Treibers

102

Abkürzungsverzeichnis

104

Literaturverzeichnis

105

Web-Adressen

105

Abbildungsverzeichnis

106

Anhang – XCP-Lösungen bei Vector

108

Sachwortverzeichnis

110

Einführung

Bei der optimalen Parametrierung (Kalibrierung) von elektronischen Steuergeräten verstellen Sie während der Laufzeit des Systems die Parameterwerte und erfassen gleichzeitig Messsignale. Die physikalische Anbindung zwischen dem Entwicklungs-Werkzeug und dem Steuergerät erfolgt über ein Mess- und Kalibrierprotokoll. XCP hat sich hier als Standard etabliert. Zunächst werden die Grundlagen und Mechanismen von XCP kurz erläutert, um anschließend die Einsatzgebiete und den Mehrwert für die Steuergeräte-Kalibrierung zu diskutieren.

Vorab einige Fakten über XCP:

- > XCP bedeutet „Universal Measurement and Calibration Protocol“. Das „X“ steht für die variable und austauschbare Transportschicht.
- > Standardisiert wurde es von einem ASAM-Arbeitskreis (Association for Standardisation of Automation and Measuring Systems). ASAM ist eine Organisation von Fahrzeugherstellern, Zulieferern und Tool-Herstellern.
- > XCP ist das Nachfolgeprotokoll von CCP (CAN Calibration Protocol).
- > Die konzeptionelle Idee des CAN Calibration Protocols war es, lesenden und schreibenden Zugriff auf steuergeräteinterne Daten über CAN zu erlauben. XCP wurde entwickelt, um diese Möglichkeit über unterschiedliche Übertragungsmedien zu realisieren. Man spricht dann von XCP on CAN, XCP on FlexRay oder XCP on Ethernet.
- > Die Hauptanwendungen von XCP sind das Messen und Verstellen von steuergeräteinternen Größen. Dabei bietet das Protokoll die Möglichkeit, Messdaten „eventsynchron“ zu Vorgängen in Steuergeräten zu erfassen. Das sichert die Konsistenz der Daten untereinander.

Zur Visualisierung der zugrunde liegenden Idee betrachten wir das Steuergerät und die darin laufende Software einmal als Blackbox. Bei einer Blackbox lassen sich nur der Input (z. B. CAN-Botschaften und Sensorwerte) in das Steuergerät und der Output des Steuergerätes (z. B. CAN-Botschaften und Aktuator-Ansteuerungen) erfassen. Details über den internen Ablauf der Algorithmen sind nicht direkt erkennbar und Rückschlüsse lassen sich nur aus der Analyse der Ein- und Ausgangsdaten ziehen.

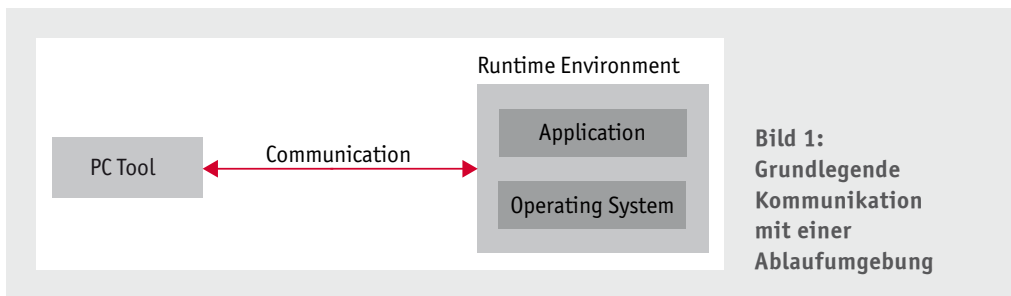
Stellen Sie sich nun vor, Sie hätten zu jedem Berechnungszyklus einen Einblick in das Verhalten Ihres Steuergerätes. Jederzeit können Sie erfassen, wie der Algorithmus im Detail abläuft. Sie hätten keine Blackbox, sondern eine Whitebox! Eine völlige Transparenz der internen Abläufe. Genau das realisieren Sie mit XCP!

Welchen Beitrag kann XCP für den kompletten Entwicklungsprozess leisten? Um die Funktionalität des erreichten Entwicklungsstandes zu überprüfen, lässt der Entwickler den Code immer wieder ablaufen. Auf diese Weise erkennt er, wie sich der Algorithmus verhält und was gegebenenfalls optimiert werden kann. Ob ein kompilierter Code auf einer spezifischen Hardware abläuft oder ob er modellbasiert entwickelt wird und die Anwendung in Form eines Modells abläuft, spielt keine Rolle.

Im Mittelpunkt steht die Bewertung des Algorithmusablaufs. Läuft beispielsweise der Algorithmus als Modell in einer Entwicklungsumgebung, wie Simulink von MathWorks, so ist es für den Entwickler hilfreich, auch die Zwischenergebnisse seiner Anwendung zu erfassen, um auf weitere Veränderungen rückzuschließen. Im Endeffekt bedeutet diese Methode nichts anderes, als

einen lesenden Zugriff auf Größen zu erhalten, um sie visualisieren und analysieren zu können – und das Ganze zur Laufzeit des Modells oder aus der Retrospektive nach Ablauf eines zeitlich befristeten Testlaufs. Ein schreibender Zugriff wird benötigt, wenn Parametrierungen verändert werden, beispielsweise wenn der Proportionalanteil eines PID-Reglers geändert wird, um das Verhalten des Algorithmus an das zu regelnde System anzupassen. Egal, wo Ihre Anwendung abläuft – Schwerpunkte sind immer die detaillierte Analyse von Algorithmusabläufen und die Optimierung durch eine Veränderung der Parametrierung.

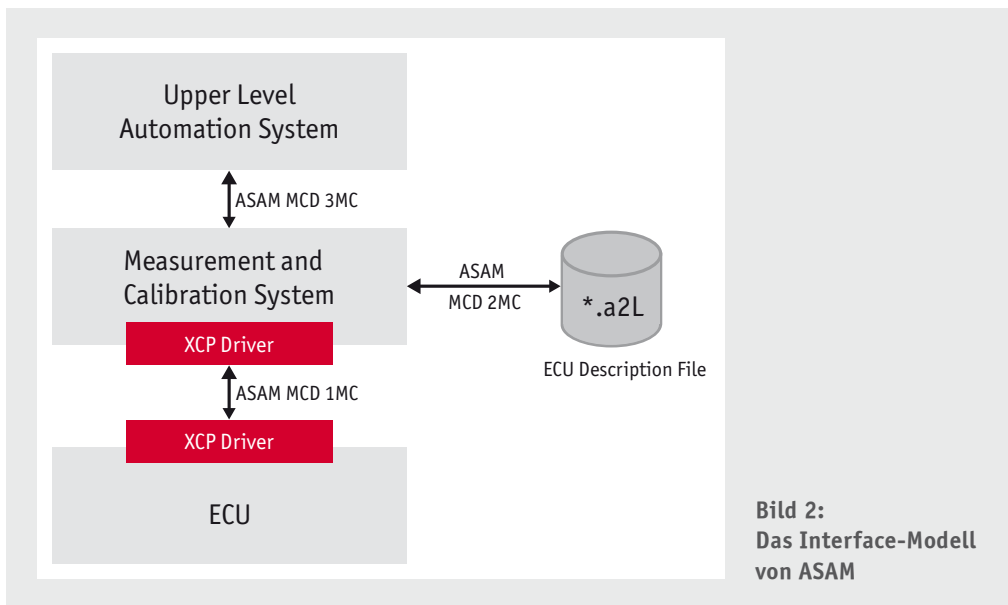
Verallgemeinert kann man festhalten: Die Algorithmen liegen in einer beliebig ablauffähigen Form (Code oder Modellbeschreibung) vor. Als Ablaufumgebung kommen unterschiedliche Systeme zum Einsatz (Simulink, als DLL auf dem PC, auf einer Rapid-Prototyping-Plattform, im Steuergerät usw.). Zur Analyse der Abläufe wird lesend auf Daten zugegriffen und deren zeitlicher Verlauf erfasst. Iterativ werden Parametrierungen verändert, um die Algorithmen zu optimieren. Zur Vereinfachung der Darstellung kann man die Erfassung der Daten in ein externes PC-basiertes Tool verlagern, wohl wissend, dass auch Ablaufumgebungen selbst bereits Möglichkeiten zur Analyse bieten können.



Die Art der Ablaufumgebung und die Form der Kommunikation unterscheiden sich im Allgemeinen sehr stark voneinander. Der Grund liegt in den von verschiedenen Herstellern entwickelten Ablaufumgebungen, die auf unterschiedlichen Lösungsansätzen basieren. Verschiedenartige Protokolle, Konfigurationen, Messdatenformate etc. machen den Austausch von Bedatungen und Ergebnissen in allen Entwicklungsschritten zu einer Sisyphusarbeit. Letztendlich lassen sich aber all diese Lösungen auf einen lesenden und schreibenden Zugriff zur Laufzeit reduzieren. Und dafür gibt es einen Standard: XCP.

XCP ist ein ASAM-Standard, der im Jahr 2003 in der Version 1.0 veröffentlicht wurde. Die Abkürzung ASAM steht für „Association for Standardisation of Automation and Measuring Systems“. Im ASAM-Arbeitskreis sind Zulieferer, Fahrzeug- und Tool-Hersteller vertreten. Der XCP-Arbeitskreis beabsichtigte, ein verallgemeinertes Mess- und Kalibrierprotokoll zu definieren, das unabhängig von einem bestimmten Transportmedium genutzt werden kann. Dabei flossen die Erfahrungen, die aus der Arbeit mit CCP (CAN Calibration Protocol) vorlagen, in die Entwicklung mit ein.

XCP wurde basierend auf dem ASAM-Schnittstellenmodell definiert. Das folgende Bild zeigt die Schnittstellen eines Mess- und Verstellwerkzeugs zum XCP-Slave, zu der Beschreibungsdatei und die Verbindung zu einem übergeordneten Automatisierungssystem.



Schnittstelle 1: „ASAM MCD-1 MC“ zwischen ECU und Mess- und Kalibriersystem

Diese Schnittstelle beschreibt den physikalischen und den protokollspezifischen Teil. Streng genommen wird hier zwischen Schnittstelle ASAP1a und ASAP1b unterschieden. Die ASAP1b-Schnittstelle konnte sich allerdings nicht durchsetzen und hat heute praktisch keine Relevanz. Das XCP-Protokoll ist so flexibel, dass es die Rolle einer allgemeinen herstellerübergreifenden Schnittstelle praktisch alleine übernehmen kann. So gibt es heute von allen Mess- und Kalibrier-Hardware-Herstellern Systeme (xETK, VX1000 ...), die über den Standard XCP on Ethernet angebunden werden. Eine ASAP1b-Schnittstelle – wie sie noch für CCP beschrieben wurde – ist nicht mehr erforderlich.

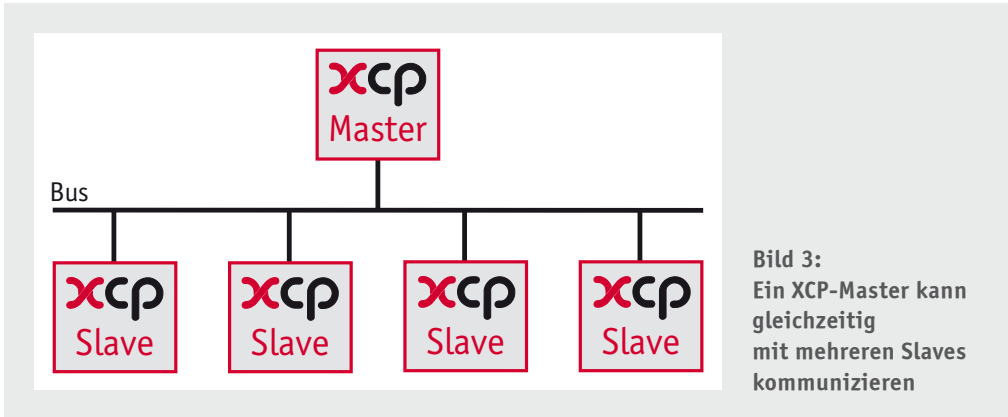
Schnittstelle 2: „ASAM MCD-2 MC“ Steuergeräte-Beschreibungsdatei A2L

Wie bereits erwähnt, arbeitet XCP adressorientiert. Lesende oder schreibende Zugriffe auf Objekte basieren immer auf der Angabe einer Adresse. In letzter Konsequenz würde das aber bedeuten, dass der Anwender im Master seine Objekte im Steuergerät anhand der Adresse ausfinden müsste. Das wäre im höchsten Maße unkomfortabel. Damit der Anwender unter anderem mit symbolischen Objektname arbeiten kann, ist eine Datei notwendig, aus der der Zusammenhang zwischen Objektname und Adresse hervorgeht. Das nächste Kapitel ist dieser A2L-Beschreibungsdatei gewidmet.

Schnittstelle 3: „ASAM MCD-3 MC“ Automatisierungs-Interface

Über dieses Interface erfolgt die Anbindung eines weiteren Systems an das Mess- und Verstellwerkzeug, z. B. zur Prüfstandautomatisierung. Die Schnittstelle wird im vorliegenden Dokument nicht weiter erklärt, da sie für das Verständnis von XCP nicht relevant ist.

XCP basiert auf dem Master-Slave-Prinzip. Das Steuergerät ist der Slave und das Mess- und Verstellwerkzeug der Master. Ein Slave kommuniziert zu einem Zeitpunkt nur mit einem Master, während der Master mit vielen Slaves gleichzeitig kommunizieren kann.



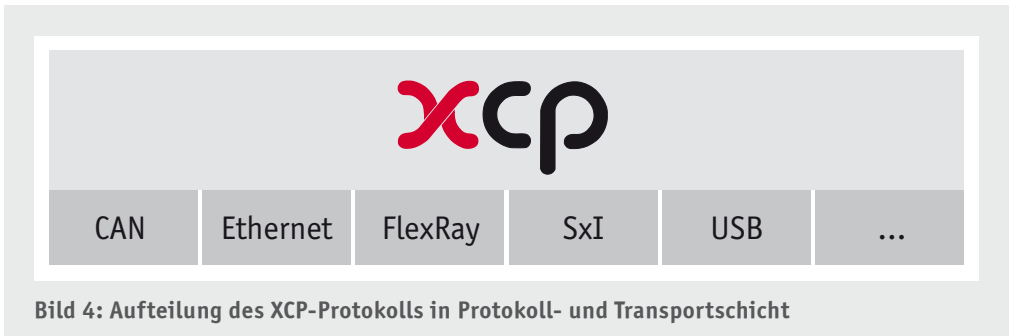
Um über den kompletten Entwicklungsprozess hinweg auf Daten und Konfigurationen zugreifen zu können, muss XCP in jeder Ablaufumgebung genutzt werden. Weniger Tools müssten gekauft, bedient und gewartet werden. Fehleranfällige manuelle Übernahmen von Konfigurationen aus einem Tool in ein anderes entfielen. Iterationsschleifen, bei denen Ergebnisse aus zeitlich später liegenden Arbeitsschritten wieder an vorangegangene Arbeitsschritte übergeben werden, würden sich vereinfachen.

Doch trennen wir uns vom Konjunktiv und halten fest, was heute schon möglich ist: alles! XCP-Lösungen kommen bereits in verschiedensten Arbeitsumgebungen zum Einsatz. Intention dieses Buches ist es, die Haupteigenschaften des Mess- und Kalibrierprotokolls zu beschreiben und die Nutzung in den unterschiedlichen Ablaufumgebungen vorzustellen. Was Sie in diesem Buch jedoch nicht finden: Weder wird die XCP-Spezifikation ganzheitlich in allen Details vorgestellt, noch wird eine exakte Anleitung zur Integrierung von XCP-Treibern in eine bestimmte Ablaufumgebung gegeben. Erläutert werden die Zusammenhänge, nicht die einzelnen Protokoll- und Implementierungsdetails. Im Anhang verweisen Internetlinks auf den frei verfügbaren XCP-Treiber-Quellcode und auf Beispielimplementierungen, die es Ihnen erlauben, sowohl die technischen Details zu verstehen als auch zu sehen, wie die Implementierung erfolgt.

Die im vorliegenden Buch verwendeten Screenshots des PC-Tools wurden mit Hilfe des Mess- und Kalibrierwerkzeugs CANape der Vector Informatik GmbH angefertigt. Anhand von CANape werden noch weitere Abläufe erklärt, unter anderem die Erstellung einer A2L-Datei und einiges mehr. Mit einer kostenlosen Demoversion, die Ihnen im Download-Center der Vector Website unter www.vector.com/canape_demo zur Verfügung steht, können Sie dies jederzeit nachvollziehen.

1 Grundlagen des XCP-Protokolls

Die Schnittstelle 1 des ASAM-Schnittstellenmodells beschreibt das Versenden und Empfangen von Kommandos und Daten zwischen dem Slave und dem Master. Um die Unabhängigkeit von einer physikalischen Transportschicht zu realisieren, wurde XCP in eine Protokoll- und eine Transportschicht unterteilt.



Je nach Transportschicht spricht man von XCP on CAN, XCP on Ethernet usw. Die Erweiterbarkeit auf neue Transportschichten wurde spätestens 2005 bewiesen, als XCP on FlexRay aus der Taufe gehoben wurde. Aktuell liegt das XCP-Protokoll in der Version 1.1 vor, die 2008 verabschiedet wurde.

Beim Design des Protokolls wurde großer Wert auf die Einhaltung folgender Prinzipien gelegt:

- > minimaler Ressourcenverbrauch im Steuergerät
- > effiziente Kommunikation
- > einfache Slave-Implementierung
- > Plug-and-play-Konfiguration mit nur geringer Parameterzahl
- > Skalierbarkeit

XCP gestattet als wesentliche Funktionalität den lesenden und schreibenden Zugriff auf den Speicher des Slaves.

Der lesende Zugriff bietet die Möglichkeit, den zeitlichen Verlauf einer steuergeräteinternen Größe zu messen. Steuergeräte sind Systeme mit diskretem Zeitverhalten, deren Größen sich nur in bestimmten Zeitintervallen verändern: immer nur dann, wenn der Prozessor den Wert neu ermittelt bzw. errechnet und ihn im RAM aktualisiert. Eine der großen Stärken von XCP liegt darin, Messdaten aus dem RAM zu erfassen, die sich synchron zu Abläufen bzw. Ereignissen im Steuergerät ändern. Somit kann der Anwender auf die unmittelbaren Zusammenhänge zwischen den zeitlichen Abläufen im Steuergerät und den sich verändernden Werten schließen. Man spricht dabei von eventsynchronen Messungen. Die entsprechenden Mechanismen werden später im Detail erklärt.

Der schreibende Zugriff erlaubt dem Anwender, Parameter von Algorithmen im Slave zu optimieren. Die Zugriffe erfolgen adressorientiert, d. h., dass sich die Kommunikation zwischen Master und Slave auf die Adressen im Speicher bezieht. Die Messung einer Größe stellt also eine Anfrage des Masters an den Slave dar: „Gib mir den Wert der Speicherstelle 0x1234“. Die Verstellung eines Parameters – der schreibende Zugriff – an den Slave bedeutet: „Setze auf der Adresse 0x9876 den Wert auf 5“.

Ein XCP-Slave muss nicht unbedingt ausschließlich in Steuergeräten zum Einsatz kommen. Er kann in unterschiedlichen Umgebungen realisiert werden: von einer modellbasierten Entwicklungsumgebung über Hardware- und Software-in-the-Loop-Umgebungen bis hin zu Hardware-Interfaces, die über Debug-Schnittstellen wie z. B. JTAG, NEXUS und DAP zum Zugriff auf den Steuergerätespeicher verwendet werden.

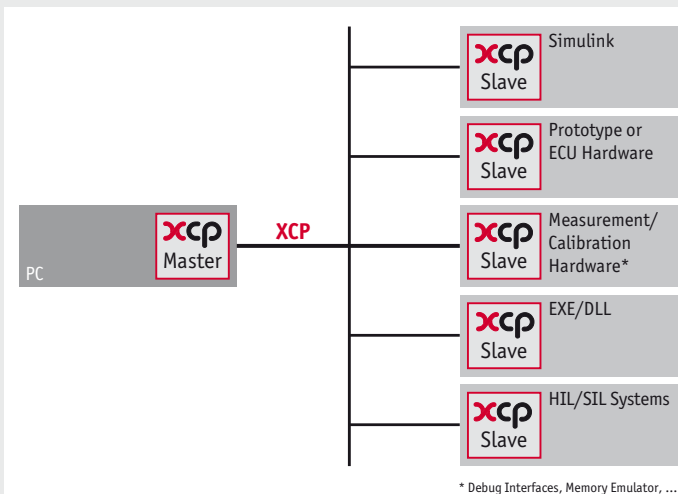


Bild 5: Slaves können in vielfältigen Ablaufumgebungen genutzt werden

Wie ist es möglich, Algorithmen mit Hilfe eines lesenden und schreibenden Zugriffs im Steuergerät zu optimieren und welchen Nutzen bringt dies? Um einzelne Größen zur Laufzeit im Steuergerät verändern zu können, muss es eine Möglichkeit geben, schreibend auf sie zuzugreifen. Nicht jede Art von Speicher erlaubt den Vorgang. Nur im RAM (das EEPROM wird bewusst außen vor gelassen) ist ein lesender und schreibender Zugriff auf Speicheradressen möglich. Im Folgenden soll ein kurzer Abriss über Speichertechnologien die Unterschiede der einzelnen Speicherarten verdeutlichen, deren Kenntnis im weiteren Verlauf für das Verständnis sehr wichtig ist.

Speichergrundlagen

Flash-Speicher sind heutzutage meist in die Mikrocontroller-Chips für Steuergeräte integriert und dienen der dauerhaften Speicherung von Code und Daten auch ohne Stromversorgung. Die Besonderheit eines Flash-Speichers ist, dass zwar jederzeit lesend auf einzelne Bytes zugegriffen werden kann, das Schreiben neuer Inhalte aber immer nur blockweise, in meist recht großen Blöcken, möglich ist.

Flash-Speicher haben eine begrenzte Lebensdauer, die in einer maximalen Anzahl an Löschzyklen angegeben wird (je nach Technologie sind das bis zu eine Million Zyklen). Dies entspricht gleichzeitig der maximalen Anzahl von Schreibzyklen, da der Speicher jeweils blockweise gelöscht werden muss, bevor er wiederum beschrieben werden kann. Der Grund hierfür liegt im Aufbau des Speichers: Über Tunnelioden werden Elektronen „gepumpt“. Die Speicherung eines Bits auf einer Speicherstelle funktioniert wie folgt: Elektronen müssen über eine elektrisch isolierende Schicht hinweg in die Speicherstelle transportiert werden. Befinden sich die Elektronen dann hinter der isolierenden Schicht, bauen sie durch ihre Ladung ein elektrisches Feld auf, das beim Lesen der Speicherstelle als 1 interpretiert wird. Befinden sich keine Elektronen hinter der Schicht, wird die Zelleninformation als 0 interpretiert. Auf diese Art und Weise kann zwar eine 1 gesetzt werden, aber keine 0. Das Setzen der 0 (= Löschen der 1) erfolgt über eine gesonderte Löschroutine, bei der vorhandene Elektronen hinter der isolierenden Schicht abgeleitet werden. Aus architektonischen Gründen funktioniert aber eine solche Löschroutine nicht auf Byte-, sondern nur auf Gruppen- oder Blockebene. Je nach Architektur sind Blöcke von z. B. 128 oder 256 Bytes üblich. Möchte man ein Byte innerhalb eines solchen Blockes überschreiben, muss zunächst der gesamte Block gelöscht werden. Danach kann der gesamte Inhalt des Blocks neu geschrieben werden.

Wird diese Löschroutine mehrmals wiederholt, kann die isolierende Schicht („Tunnel Oxide Film“) beschädigt werden. Das führt dazu, dass die Elektronen langsam abfließen können und somit die Information im Laufe der Zeit von 1 auf 0 wechselt. Deshalb wird die Anzahl der zulässigen Flash-Zyklen in einem Steuergerät stark limitiert. Sie liegt im Seriensteuergerät oft nur im einstelligen Bereich. Die Einschränkung erfolgt durch den Flash Boot Loader, der unter Zuhilfenahme eines Zählers erfasst, wie viele Flash-Vorgänge schon durchgeführt worden sind. Wird die festgelegte Anzahl überschritten, lehnt der Flash Boot Loader eine erneute Flash-Aufforderung ab.

Ein RAM (Random Access Memory) benötigt eine permanente Stromversorgung, da es sonst seinen Inhalt verliert. Während der Flash-Speicher zur dauerhaften Speicherung der Anwendung dient, wird das RAM zur Zwischenspeicherung von errechneten Daten und anderen temporären

Informationen verwendet. Durch das Abschalten der Stromversorgung geht der Inhalt im RAM verloren. Im Gegensatz zum Flash-Speicher kann auf das RAM einfach lesend und schreibend zugegriffen werden.

Halten wir also fest: Sollen Parameter zur Laufzeit geändert werden, so muss sichergestellt sein, dass sie sich im RAM befinden. Diesen Umstand zu verstehen ist wirklich sehr wichtig. Deshalb betrachten wir den Ablauf einer Anwendung im Steuergerät an folgendem Beispiel:

In der Applikation errechnet sich die Größe y aus dem Sensorwert x .

// Pseudocode-Darstellung

$a = 5;$

$b = 2;$

$y = a * x + b;$

Wird die Anwendung in das Steuergerät geflasht, so handhabt der Controller nach dem Booten diesen Code folgendermaßen: Der Wert der Größe x entspricht einem Sensorwert. Zu irgendeinem Zeitpunkt muss also die Anwendung den Sensorwert abfragen und der Wert wird dann in einer Speicherstelle gespeichert, die der Größe x zugeordnet ist. Da dieser Wert zur Laufzeit immer wieder geschrieben werden muss, kann der Speicherplatz nur im RAM liegen.

Die Größe y wird errechnet. Die Werte a und b als Faktor und Offset stehen als Information im Flash-Speicher. Sie sind als Konstanten dort abgelegt. Der Wert von y muss ebenfalls im RAM hinterlegt werden, da wiederum nur dort Schreibzugriff möglich ist. An genau welcher Stelle im RAM die Größen x und y bzw. wo im Flash a und b liegen, wird beim Compiler/Linker-Lauf festgelegt. Hier erfolgt die eindeutige Zuordnung der Objekte zu Adressen. Der Zusammenhang zwischen Objektname, Datentyp und Adresse wird in der Linker-Map-Datei dokumentiert. Die Linker-Map-Datei wird durch den Linker-Lauf erzeugt und kann in unterschiedlichen Formaten vorliegen. Allen Formaten gemeinsam ist aber, dass zumindest Objektname und Adresse darin enthalten sind.

Sind in dem Beispiel der Offset b und der Faktor a abhängig vom Fahrzeug, so müssen die Werte von a und b individuell an die Verhältnisse angepasst werden. Das heißt, der Algorithmus bleibt, wie er ist, aber die Parameterwerte ändern sich von Fahrzeug zu Fahrzeug.

Im normalen Betriebsmodus eines Steuergerätes läuft die Anwendung aus dem Flash-Speicher heraus. Dieser erlaubt keinen Schreibzugriff auf einzelne Objekte. Das bedeutet, dass Parameterwerte, die sich im Flash-Bereich befinden, nicht zur Laufzeit verändert werden können. Soll eine Änderung der Parameterwerte während der Laufzeit möglich sein, müssen die zu verändernden Größen im RAM und nicht im Flash liegen. Wie kommen nun die Parameter und ihre Initialwerte in das RAM? Wie löst man das Problem, wenn mehr Parameter geändert werden sollen, als gleichzeitig im RAM untergebracht werden können? Diese Fragestellungen führen uns zum Thema Kalibrierkonzepte (siehe Kapitel 3).

Zusammenfassung der XCP-Grundlagen

Mit den Mechanismen des XCP-Protokolls stehen lesende und schreibende Zugriffe auf Speicherinhalte zur Verfügung. Die Zugriffe erfolgen adressorientiert. Der lesende Zugriff erlaubt das Messen von Größen aus dem RAM und der schreibende Zugriff das Verstellen der Parameter im RAM. XCP gestattet, die Messung synchron zu den Ereignissen im Steuergerät auszuführen. Damit ist sichergestellt, dass die Messwerte miteinander korrelieren. Bei jedem Neustart einer Messung können die zu messenden Signale frei ausgewählt werden. Für den schreibenden Zugriff müssen die Parameter, die verstellt werden sollen, im RAM stehen. Dies erfordert ein Kalibrierkonzept.

Daraus ergeben sich zwei wichtige Fragestellungen:

- > Woher kennt der Anwender des XCP-Protokolls die richtige Adresse der Mess- und Verstellgrößen im RAM?
- > Wie sieht das Kalibrierkonzept aus?

Die erste Frage wird im Kapitel 2 „Steuergeräte-Beschreibungsdatei A2L“ beantwortet. Auf das Thema Kalibrierkonzept wird im Kapitel 3 eingegangen.

1.1 XCP-Protokollschicht

XCP-Daten werden botschaftsorientiert zwischen Master und Slave ausgetauscht. Der gesamte „XCP Message Frame“ ist eingeschlossen in einen Frame des Transport Layers (im Falle von XCP on Ethernet mit UDP in ein UDP-Paket). Der Frame besteht aus drei Teilen: dem XCP Header, dem XCP Packet und dem XCP Tail.

In der folgenden Abbildung ist der Teil einer Botschaft rot hinterlegt. Darin wird die eigentliche XCP-Nachricht versendet. XCP Header und XCP Tail sind abhängig vom Transportprotokoll.

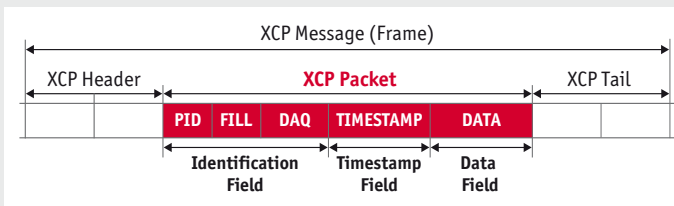


Bild 6: XCP-Paket

Das XCP-Paket selbst ist unabhängig vom verwendeten Transportprotokoll. Es beinhaltet immer drei Komponenten: „Identification Field“, „Timestamp Field“ und das eigentliche Datenfeld „Data Field“. Jedes Identification Field beginnt mit dem Packet Identifier (PID), der das Paket identifiziert.

Die folgende Übersicht zeigt, welche PIDs festgelegt sind:

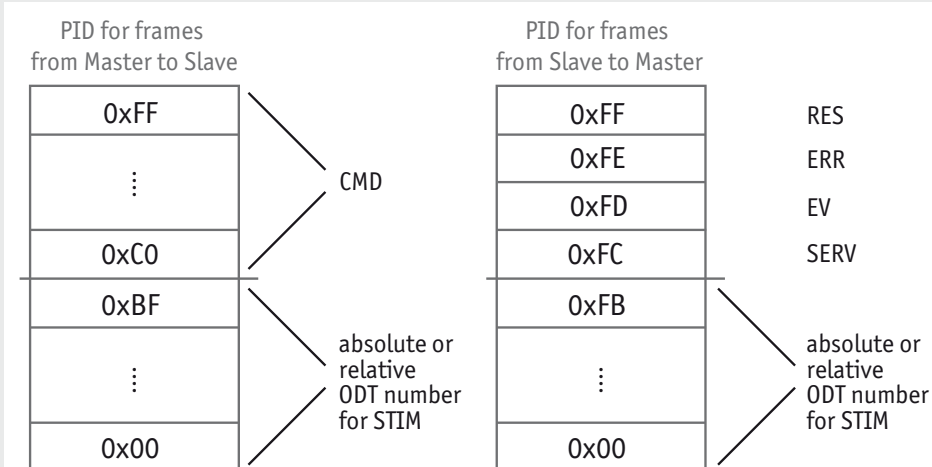


Bild 7: Überblick über die XCP Packet Identifier (PID)

Die Kommunikation über das XCP-Paket ist dabei unterteilt in einen Bereich für Kommandos (CTO) und in einen Bereich für die Versendung synchroner Daten (DTO).

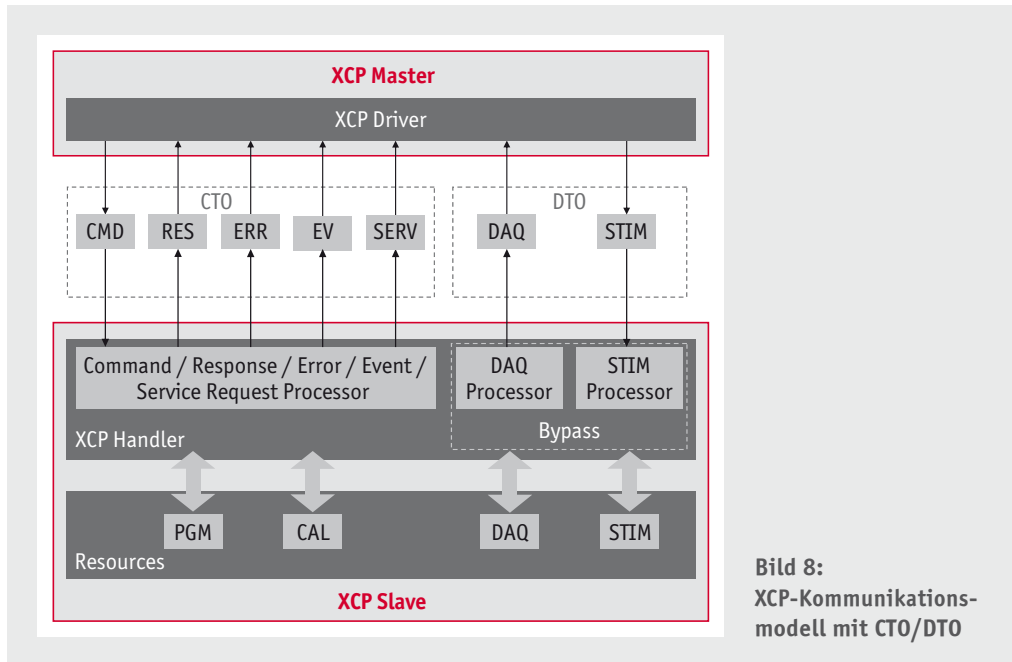


Bild 8:
XCP-Kommunikations-
modell mit CTO/DTO

Die verwendeten Abkürzungen stehen für:

CMD	Command Packet	Kommandos versenden
RES	Command Response Packet	positive Antwort
ERR	Error	negative Antwort
EV	Event Packet	asynchrones Ereignis
SERV	Service Request Packet	Serviceanfrage
DAQ	Data Acquisition	zyklische Messdaten senden
STIM	Stimulation	zyklische Stimulation des Slaves

Der Austausch von Kommandos erfolgt über CTOs (Command Transfer Objects). Auf diesem Weg findet beispielsweise die Kontaktaufnahme durch den Master statt. Ein CMD muss immer vom Slave mit RES oder ERR beantwortet werden. Die anderen CTO-Botschaften werden asynchron versendet. Die Data Transfer Objects (DTO) dienen dem Austausch synchroner Mess- und Stimulationsdaten.

1.1.1 Identification Field

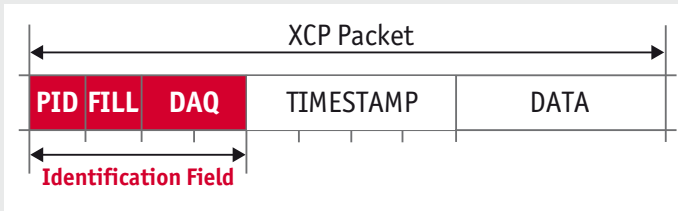


Bild 9:
Die Botschafts-
Identifikation

Beim Austausch von Botschaften müssen sowohl Master als auch Slave erkennen können, welche Botschaft vom jeweils anderen versendet wurde. Dieser Vorgang erfolgt mit Hilfe des Identifikationsfeldes. Daher startet jede Botschaft mit dem Packet Identifier (PID).

Bei der Übertragung von CTOs genügt das PID-Feld vollständig, um ein CMD-, RES- oder ein anderes CTO-Paket zu identifizieren. In Bild 7 ist zu erkennen, dass die Kommandos des Masters an den Slave eine PID von 0xC0 bis 0xFF nutzen. Der XCP-Slave antwortet bzw. informiert den Master mit PIDs von 0xFC bis 0xFF. Damit gibt es eine eindeutige Zuordnung der PID zu den jeweils gesendeten CTOs.

Werden DTOs übertragen, kommen weitere Bestandteile des Identifikationsfeldes zum Einsatz (siehe Kapitel 1.3.4 „XCP-Paket-Adressierung für DAQ und STIM“).

1.1.2 Timestamp

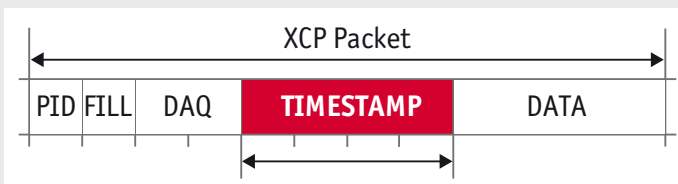


Bild 10:
Zeitstempel

DTO-Pakete nutzen Zeitstempel, bei der Übertragung einer CTO-Botschaft ist dies nicht möglich. Mit dem Zeitstempel gibt der Slave den Messwerten eine Zeitinformation mit. Im Master steht nicht nur der Messwert, sondern auch der Zeitpunkt, wann der Messwert erfasst wurde, zur Verfügung. Die Zeit, die der Messwert benötigt, um zum Master zu gelangen, spielt keine Rolle mehr, da der Zusammenhang zwischen Messwert und Zeitpunkt direkt aus dem Slave heraus erfolgt.

Die Übertragung eines Zeitstempels aus dem Slave ist optional. Mehr dazu ist in der ASAM XCP Teil 2 Protocol Layer Specification 1.1.0 beschrieben.

1.1.3 Data Field

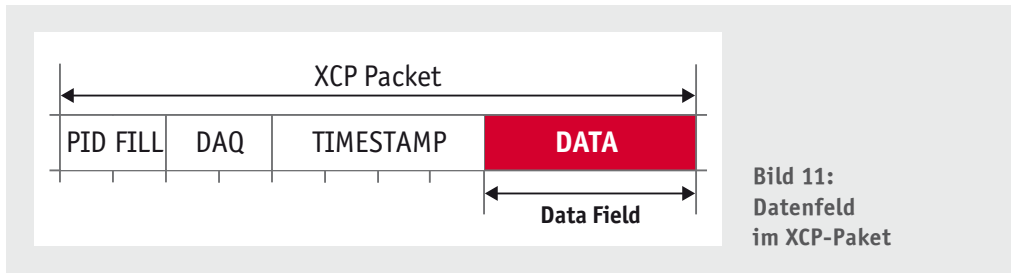


Bild 11:
Datenfeld
im XCP-Paket

Letztendlich beinhaltet das XCP-Paket auch im Datenfeld abgelegte Daten. Im Falle von CTO-Paketen besteht das Datenfeld aus spezifischen Parametern für die unterschiedlichen Kommandos. DTO-Pakete enthalten die Messwerte aus dem Slave und bei der Versendung von STIM-Daten die Werte aus dem Master.

1.2 Austausch von CTOs

Über CTOs werden sowohl Kommandos vom Master an den Slave als auch die Antworten des Slaves an den Master übertragen.

1.2.1 XCP-Kommandostruktur

Der Slave empfängt ein Kommando des Masters und muss darauf mit einer positiven oder negativen Antwort reagieren. Die Kommunikationsstruktur ist dabei immer gleich:

Kommando (CMD):

Position	Type	Description
0	BYTE	Command Packet Code CMD
1..MAX_CTO-1	BYTE	Command specific Parameters

Jedem Kommando ist eine eindeutige Nummer zugeordnet. Zusätzlich können weitere spezifische Parameter mit übertragen werden. Die maximale Anzahl an Parametern ist dabei definiert als MAX_CTO-1. MAX_CTO gibt die maximale Länge der CTO-Pakete in Bytes an.

Positive Antwort:

Position	Type	Description
0	BYTE	Command Positive Response Packet Code = RES 0xFF
1..MAX_CTO-1	BYTE	Command specific Parameters

Negative Antwort:

Position	Type	Description
0	BYTE	Error Packet Code = 0xFE
1	BYTE	Error code
2..MAX_CTO-1	BYTE	Command specific Parameters

Nicht nur bei der positiven, sondern auch bei der negativen Antwort können spezifische Parameter als zusätzliche Information übermittelt werden. Als Beispiel soll der Aufbau der Verbindung zwischen Master und Slave dienen. Zu Beginn einer Kommunikation zwischen Master und Slave sendet der Master eine Connect-Anforderung an den Slave, die dieser positiv beantworten muss, damit eine kontinuierliche Punkt-zu-Punkt-Verbindung entsteht.

Master → Slave: Connect

Slave → Master: positive Response

Connect-Kommando:

Position	Type	Description
0	BYTE	Command Code = 0xFF
1	BYTE	Mode 00 = Normal 01 = user defined

Mode 00 bedeutet, dass der Master eine XCP-Kommunikation mit dem Slave wünscht. Verwendet der Master 0xFF 0x01 beim Verbindungsaufbau, so fordert der Master eine XCP-Kommunikation mit dem Slave. Dabei teilt er dem Slave gleichzeitig mit, dass dieser in einen bestimmten – benutzerdefinierten – Modus wechseln soll.

Positive Antwort des Slaves:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	RESOURCE
2	BYTE	COMM_MODE_BASIC
3	BYTE	MAX_CTO, Maximum CTO size [BYTE]
4	WORD	MAX_DTO, Maximum DTO size [BYTE]
6	BYTE	XCP Protocol Layer Version Number (most significant byte only)
7	BYTE	XCP Transport Layer Version Number (most significant byte only)

Die positive Antwort des Slaves kann etwas umfangreicher ausfallen. Der Slave teilt dem Master schon beim Verbindungsaufbau kommunikationsspezifische Informationen mit. RESOURCE ist beispielsweise eine Information, mit der der Slave meldet, ob er etwa Page Switching unterstützt oder ob Flashen über XCP möglich ist. Mit MAX_DTO informiert der Slave den Master darüber, welche maximale Paketlänge für die Messdaten-Übertragung der Slave unterstützt usw. Die Details zu den Parametern finden Sie in ASAM XCP Teil 2 Protocol Layer Specification 1.1.0.

XCP erlaubt drei unterschiedliche Modi zum Austausch von Kommandos und Reaktionen zwischen Master und Slave: Standard, Block und Interleaved Mode.

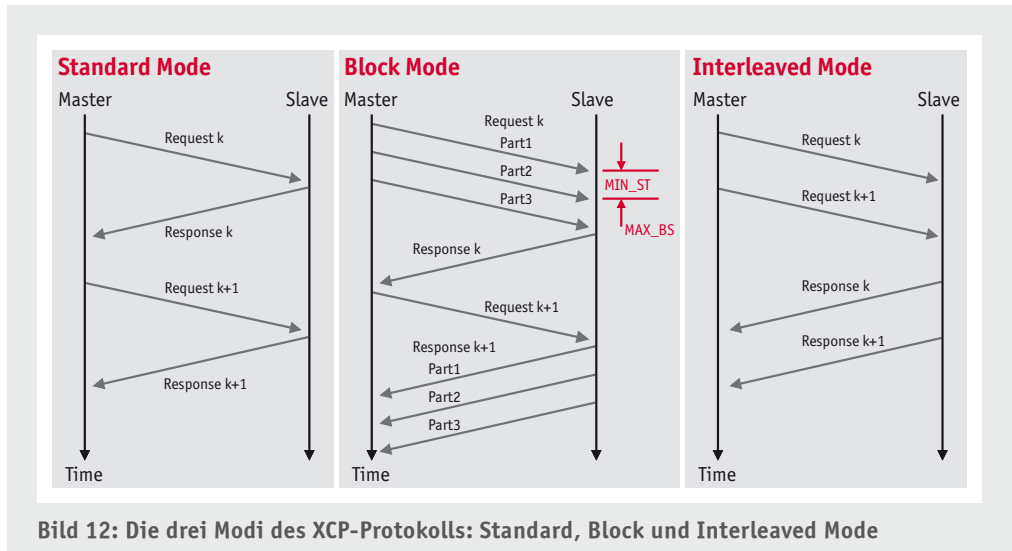


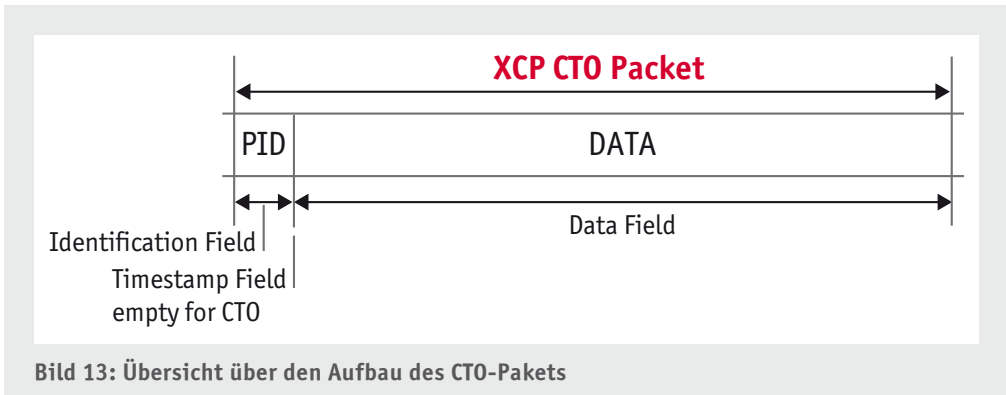
Bild 12: Die drei Modi des XCP-Protokolls: Standard, Block und Interleaved Mode

Bei dem Standard-Kommunikationsmodell folgt auf jede Anforderung an einen Slave eine einzige Response. Bis auf eine Ausnahme bei XCP on CAN ist es nicht erlaubt, dass mehrere Slaves auf ein Kommando des Masters reagieren. Die XCP-Botschaften müssen folglich eindeutig den Slaves zuordenbar sein. Dieser Modus ist der Standardfall der Kommunikation.

Der Blocktransfer-Modus ist optional und spart bei umfangreichen Datenübertragungen (beispielsweise bei Upload- oder Download-Vorgängen) Zeit ein. Allerdings muss bei diesem Modus in Richtung Slave dessen Leistungsfähigkeit berücksichtigt werden. Deshalb müssen Mindestabstand-Zeiten (MIN_ST) zwischen zwei Kommandos eingehalten und die Anzahl der Kommandos insgesamt muss auf eine Obergrenze MAX_BS begrenzt werden. Optional können diese Kommunikationseinstellungen vom Master über GET_COMM_MODE_INFO aus dem Slave ausgelesen werden. Die eben erwähnten Beschränkungen müssen beim Blocktransfer-Modus in Richtung Master nicht beachtet werden, da die Performance des PCs in der Regel immer ausreicht, die Daten aus einem Mikrocontroller entgegenzunehmen.

Der Interleaved-Modus ist ebenfalls aus Performance-Gründen vorgesehen. Aber auch dieses Verfahren ist optional und hat – im Gegensatz zum Blocktransfer-Modus – keine Praxisrelevanz.

1.2.2 CMD



Der Master sendet eine allgemeine Anforderung über CMD an den Slave. Das PID (Packet Identifier)-Feld beinhaltet die Identifikationsnummer des Kommandos. Im Datenfeld werden die zusätzlichen spezifischen Parameter transportiert. Danach wartet der Master auf eine Reaktion des Slaves in Form einer RESponse oder eines ERRors.

XCP ist auch bei der Implementierung sehr skalierbar, sodass nicht jedes Kommando implementiert werden muss. In der A2L-Datei sind in der sogenannten XCP IF_DATA die zur Verfügung stehenden CMDs aufgeführt. Sollte es eine Diskrepanz zwischen der Definition in der A2L-Datei und der Realisierung im Slave geben, kann der Master aufgrund der Reaktion des Slaves erkennen, dass der Slave das Kommando doch nicht unterstützt. Versendet der Master ein Kommando, das nicht im Slave implementiert worden ist, so muss der Slave mit ERR_CMD_UNKNOWN quittieren und es werden keine weiteren Aktivitäten im Slave ausgelöst. Somit erkennt der Master rasch, dass ein optionales Kommando nicht im Slave implementiert worden ist.

Zum Teil sind noch weitere Parameter in den Kommandos enthalten. Die genauen Details entnehmen Sie bitte der Protokollschicht-Spezifikation des ASAM XCP Teil 2 Protocol Layer Specification 1.1.0.

Die Kommandos sind in Gruppen aufgeteilt: Standard-, Kalibrier-, Seiten-, Programmier- und DAQ-Messungs-Kommandos. Wird eine Gruppe gar nicht benötigt, müssen ihre Kommandos nicht implementiert sein. Ist die Gruppe notwendig, müssen die entsprechenden Kommandos auf jeden Fall im Slave zur Verfügung stehen, andere aus der Gruppe sind optional.

Die folgende Übersicht dient als Beispiel. Die Kommandos SET_CAL_PAGE und GET_CAL_PAGE in der Gruppe Seitenumschaltung sind als nicht optional gekennzeichnet. Das heißt, in einem XCP-Slave, der das Page Switching unterstützt, müssen mindestens diese beiden Befehle implementiert sein. Ist die Page-Switching-Unterstützung im Slave nicht notwendig, müssen diese Befehle nicht implementiert sein. Das Gleiche gilt entsprechend für andere Kommandos.

Standardkommandos:

Kommando	PID	Optional
CONNECT	0xFF	No
DISCONNECT	0xFE	No
GET_STATUS	0xFD	No
SYNCH	0xFC	No
GET_COMM_MODE_INFO	0xFB	Yes
GET_ID	0xFA	Yes
SET_REQUEST	0xF9	Yes
GET_SEED	0xF8	Yes
UNLOCK	0xF7	Yes
SET_MTA	0xF6	Yes
UPLOAD	0xF5	Yes
SHORT_UPLOAD	0xF4	Yes
BUILD_CHECKSUM	0xF3	Yes
TRANSPORT_LAYER_CMD	0xF2	Yes
USER_CMD	0xF1	Yes

Kalibrierkommandos:

Kommando	PID	Optional
DOWNLOAD	0xF0	No
DOWNLOAD_NEXT	0xEF	Yes
DOWNLOAD_MAX	0xEE	Yes
SHORT_DOWNLOAD	0xED	Yes
MODIFY_BITS	0xEC	Yes

Seitenumschaltung:

Kommando	PID	Optional
SET_CAL_PAGE	0xEB	No
GET_CAL_PAGE	0xEA	No
GET_PAG_PROCESSOR_INFO	0xE9	Yes
GET_SEGMENT_INFO	0xE8	Yes
GET_PAGE_INFO	0xE7	Yes
SET_SEGMENT_MODE	0xE6	Yes
GET_SEGMENT_MODE	0xE5	Yes
COPY_CAL_PAGE	0xE4	Yes

Zyklischer Datenaustausch – Basics:

Kommando	PID	Optional
SET_DAQ_PTR	0xE2	No
WRITE_DAQ	0xE1	No
SET_DAQ_LIST_MODE	0xE0	No
START_STOP_DAQ_LIST	0xDE	No
START_STOP_SYNCH	0xDD	No
WRITE_DAQ_MULTIPLE	0xC7	Yes
READ_DAQ	0xDB	Yes
GET_DAQ_CLOCK	0xDC	Yes
GET_DAQ_PROCESSOR_INFO	0xDA	Yes
GET_DAQ_RESOLUTION_INFO	0xD9	Yes
GET_DAQ_LIST_INFO	0xD8	Yes
GET_DAQ_EVENT_INFO	0xD7	Yes

Zyklischer Datenaustausch – statische Konfiguration:

Kommando	PID	Optional
CLEAR_DAQ_LIST	0xE3	No
GET_DAQ_LIST_INFO	0xD8	Yes

Zyklischer Datenaustausch – dynamische Konfiguration:

Kommando	PID	Optional
FREE_DAQ	0xD6	Yes
ALLOC_DAQ	0xD5	Yes
ALLOC_ODT	0xD4	Yes
ALLOC_ODT_ENTRY	0xD3	Yes

Flash-Programmierung:

Kommando	PID	Optional
PROGRAM_START	0xD2	No
PROGRAM_CLEAR	0xD1	No
PROGRAM	0xD0	No
PROGRAM_RESET	0xCF	No
GET_PGM_PROCESSOR_INFO	0xCE	Yes
GET_SECTOR_INFO	0xCD	Yes
PROGRAM_PREPARE	0xCC	Yes
PROGRAM_FORMAT	0xCB	Yes
PROGRAM_NEXT	0xCA	Yes
PROGRAM_MAX	0xC9	Yes
PROGRAM_VERIFY	0xC8	Yes

1.2.3 RES

Kann der Slave eine Master-Anforderung erfolgreich umsetzen, bestätigt er positiv mit RES.

Position	Type	Description
0	BYTE	Packet Identifier = RES 0xFF
1..MAX_CTO-1	BYTE	Command response data

Weiterführende Informationen über die Parameter finden Sie in ASAM XCP Teil 2 Protocol Layer Specification 1.1.0.

1.2.4 ERR

Ist die Anforderung des Masters unbrauchbar, meldet er sich mit der Fehlermeldung ERR und einem Fehlercode.

Position	Type	Description
0	BYTE	Packet Identifier = ERR 0xFE
1	BYTE	Error code
2..MAX_CTO-1	BYTE	Optional error information data

Die möglichen Error-Codes finden Sie in ASAM XCP Teil 2 Protocol Layer Specification 1.1.0.

1.2.5 EV

Ein EV kann gesendet werden, wenn der Slave den Master über ein asynchrones Ereignis informieren möchte. Die Implementierung ist optional.

Position	Type	Description
0	BYTE	Packet Identifier = EV 0xFD
1	BYTE	Event code
2..MAX_CTO-1	BYTE	Optional event information data

Weiterführende Informationen über die Parameter finden Sie in ASAM XCP Teil 2 Protocol Layer Specification 1.1.0.

Im Zusammenhang mit Messungen und Stimulation wird noch häufiger die Rede von Ereignissen sein. Die Aktion des XCP-Slaves, die das Versenden eines EVENTS auslöst, hat damit nichts zu tun. Hier geht es darum, dass der Slave beispielsweise eine Störung meldet, wie den Ausfall einer Funktionalität.

1.2.6 SERV

Über diesen Mechanismus kann der Slave beim Master die Ausführung eines Service anfordern.

Position	Type	Description
0	BYTE	Packet Identifier = SERV 0xFC
1	BYTE	Service request code
2..MAX_CTO-1	BYTE	Optional service request data

Die Service-Request-Code-Tabelle finden Sie in ASAM XCP Teil 2 Protocol Layer Specification 1.1.0.

1.2.7 Verstellen von Parametern im Slave

Um einen Parameter in einem XCP-Slave zu verändern, muss der XCP-Master sowohl die Adresse, auf der sich der Parameter befindet, als auch den Wert selbst zum Slave senden.

XCP definiert Adressen immer mit fünf Bytes: vier für die eigentliche Adresse und ein Byte für die Adress-Extension. Basierend auf einer CAN-Übertragung stehen für XCP-Botschaften nur sieben Nutz-Bytes zur Verfügung. Setzt man beispielsweise einen 4-Byte-Wert und will beide Informationen in einer CAN-Botschaft versenden, reicht der Platz nicht aus. Da für die Übertragung der Adresse und des neuen Wertes insgesamt neun Bytes benötigt werden, kann die Änderung in einer CAN-Botschaft (sieben Nutz-Bytes) nicht übermittelt werden. Die Verstellenanforderung erfolgt also mit zwei Botschaften vom Master zum Slave. Der Slave muss beide Botschaften bestätigen, in der Summe werden vier Botschaften ausgetauscht.

Die folgende Abbildung zeigt die Kommunikation zwischen Master und Slave, die zur Setzung eines Parameterwertes notwendig ist. In der Linie mit dem Briefumschlagsymbol befindet sich die eigentliche Botschaft. Durch das „Aufklappen“ wird die Interpretation der Botschaft sichtbar.

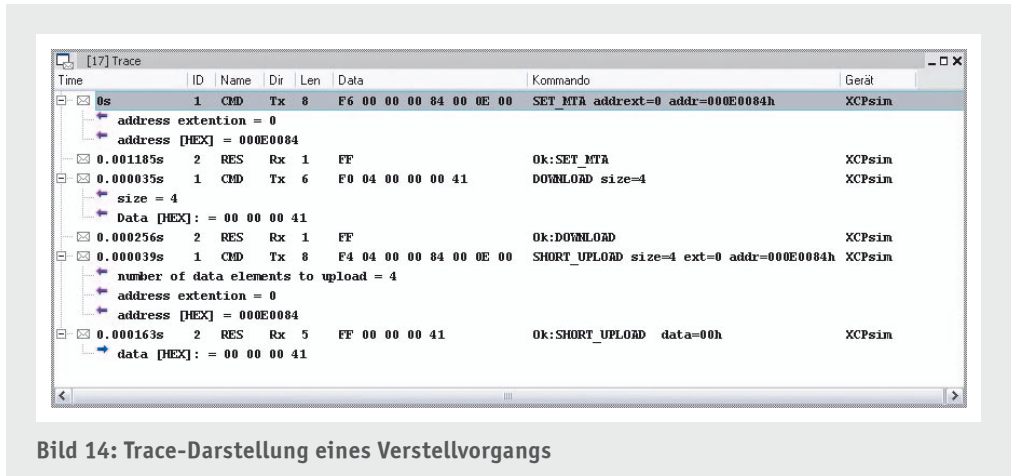


Bild 14: Trace-Darstellung eines Verstellvorgangs

Die erste Botschaft des Masters (in Bild 14 grau markiert) überträgt mit dem Befehl SET_MTA die Adresse, auf die ein neuer Wert geschrieben werden soll, an den Slave. Mit der zweiten Botschaft bestätigt der Slave das Kommando positiv mit Ok:SET_MTA.

Die dritte Botschaft DOWNLOAD überträgt den Hex-Wert und zusätzlich die gültige Anzahl an Bytes. In diesem Beispiel beträgt die gültige Bytes-Anzahl vier, da es sich um einen Float-Wert handelt. Der Slave bestätigt mit der vierten Botschaft wieder positiv.

Damit ist der eigentliche Verstellvorgang komplett abgeschlossen. In der Trace-Darstellung können Sie einen anschließenden SHORT_UPLOAD erkennen – eine Besonderheit von CANape, dem Mess- und Kalibrierwerkzeug von Vector. Um sicherzugehen, dass die Verstellung funktioniert hat, wird nach dem Vorgang der Wert nochmals ausgelesen und die Anzeige mit dem ausgelesenen Wert aktualisiert. So kann der Anwender direkt erkennen, ob sein Verstellbefehl umgesetzt worden ist. Auch dieses Kommando wird positiv mit Ok:SHORT_UPLOAD bestätigt.

Parametersatzdatei speichern

Mit der Veränderung des Parameters im RAM des Steuergerätes verarbeitet die Anwendung den neuen Wert. Ein erneutes Booten des Steuergerätes führt jedoch zur Löschung des Wertes und zu einem Überschreiben des Wertes im RAM mit dem ursprünglichen Wert aus dem Flash (siehe Kapitel 3 „Kalibrierkonzepte“). Wie kann also die geänderte Parametrierung dauerhaft gespeichert werden?

Grundsätzlich gibt es zwei Möglichkeiten:

A) Speichern der Parameter im Steuergerät

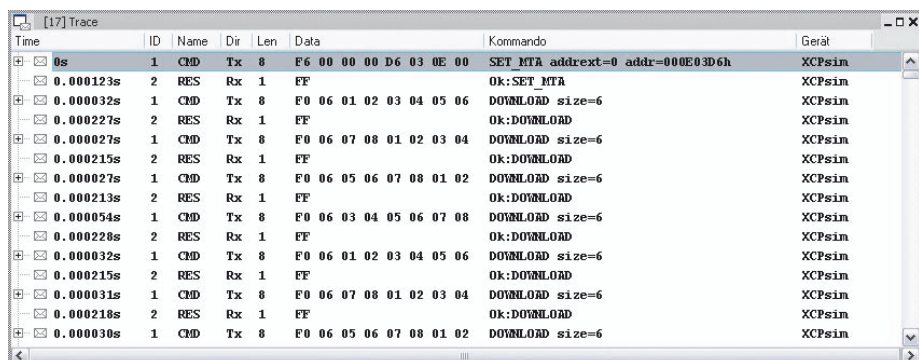
Die geänderten Daten im RAM können beispielsweise im EEPROM im Steuergerät gesichert werden, entweder automatisch, z. B. beim Runterfahren des Steuergerätes, oder manuell durch den Anwender. Voraussetzung ist, dass die Daten in einem nicht flüchtigen Speicher des Slaves gespeichert werden können. Bei einem Steuergerät sind das EEPROM oder Flash. Steuergeräte mit Tausenden von Parametern halten allerdings selten so viel ungenutzten EEPROM-Speicherplatz bereit, sodass diese Vorgehensweise eher selten zum Einsatz kommt.

Eine weitere Möglichkeit ist das Zurückschreiben der RAM-Größen in den Flash-Speicher des Steuergerätes. Dieses Verfahren ist relativ komplex. Ein Flash-Speicher muss erst gelöscht werden, bevor er neu beschrieben werden kann. Das wiederum kann nur blockweise erfolgen. Das Zurückschreiben einzelner Bytes ist somit nicht ohne Weiteres möglich. Mehr dazu finden Sie im Kapitel 3.

B) Speichern der Parameter in Form einer Datei auf dem PC

Viel gängiger ist die Ablage der Parameter auf dem PC. Alle Parameter – oder Teilmengen davon – werden in Form einer Datei abgelegt. Dazu stehen unterschiedliche Formate zur Verfügung, im einfachsten Fall eine ASCII-Textdatei, die nur den Namen des Objektes und den Wert beinhaltet. Andere Formate erlauben auch die Abspeicherung weiterer Informationen, wie z. B. Aussagen über den Reifegrad des Parameters oder die Historie der Veränderungen.

Szenario: Nach getaner Arbeit möchte der Applikateur in den wohlverdienten Feierabend gehen. Er speichert die im RAM des Steuergerätes durchgeführten Änderungen in Form einer Parametersatzdatei auf seinem PC ab. Am nächsten Tag möchte er dort weiterarbeiten, wo er am Vortag aufgehört hat. Er startet das Steuergerät. Beim Booten werden die Parameter im RAM initialisiert. Dazu verwendet das Steuergerät aber die Werte, die im Flash gespeichert sind. Das bedeutet, dass die Veränderungen vom Vortag nicht mehr im Steuergerät vorhanden sind. Um nun wieder mit dem Stand vom Vortag weiterarbeiten zu können, überträgt der Applikateur per XCP den Inhalt der Parametersatzdatei in das RAM des Steuergerätes per DOWNLOAD-Befehl.



Time	ID	Name	Dir	Len	Data	Kommando	Gerät
0s	1	CMD	Tx	8	F6 00 00 00 D6 03 0E 00	SET MTA addrest=0 addr=000E03D6h	XCPsin
0.000123s	2	RES	Rx	1	FF	Ok:SET_MTA	XCPsin
0.000032s	1	CMD	Tx	8	F0 06 01 02 03 04 05 06	DOWNLOAD size=6	XCPsin
0.000227s	2	RES	Rx	1	FF	Ok:DOWNLOAD	XCPsin
0.000027s	1	CMD	Tx	8	F0 06 07 08 01 02 03 04	DOWNLOAD size=6	XCPsin
0.000215s	2	RES	Rx	1	FF	Ok:DOWNLOAD	XCPsin
0.000027s	1	CMD	Tx	8	F0 06 05 06 07 08 01 02	DOWNLOAD size=6	XCPsin
0.000213s	2	RES	Rx	1	FF	Ok:DOWNLOAD	XCPsin
0.000054s	1	CMD	Tx	8	F0 06 03 04 05 06 07 08	DOWNLOAD size=6	XCPsin
0.000228s	2	RES	Rx	1	FF	Ok:DOWNLOAD	XCPsin
0.000032s	1	CMD	Tx	8	F0 06 01 02 03 04 05 06	DOWNLOAD size=6	XCPsin
0.000215s	2	RES	Rx	1	FF	Ok:DOWNLOAD	XCPsin
0.000031s	1	CMD	Tx	8	F0 06 07 08 01 02 03 04	DOWNLOAD size=6	XCPsin
0.000218s	2	RES	Rx	1	FF	Ok:DOWNLOAD	XCPsin
0.000030s	1	CMD	Tx	8	F0 06 05 06 07 08 01 02	DOWNLOAD size=6	XCPsin

Bild 15: Übertragung einer Parametersatzdatei in das RAM des Steuergerätes

Parametersatzdatei in Hex-Files speichern und flashen

Das Flashen eines Steuergerätes ist eine weitere Möglichkeit, die Parameter im Flash zu ändern. Diese werden dann beim Booten des Steuergerätes als neue Parameter in das RAM geschrieben. Eine Parametersatzdatei kann dazu in ein C- oder H-File überführt werden und per erneutem Compiler/Linker-Lauf wird dann daraus das neue Flash-File. Je nach Größe des Codes kann die Generierung eines flashbaren Hex-Files aber einige Zeit in Anspruch nehmen. Dazu kommt, dass – je nach Arbeitsprozess – der Applikateur gar keinen Source Code des Steuergerätes hat. Somit steht ihm dieser Weg nicht zur Verfügung.

Alternativ kann er die Parametersatzdatei in das vorhandene Flash-File kopieren.

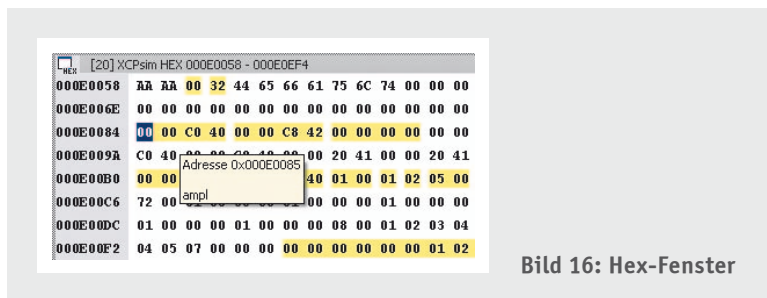


Bild 16: Hex-Fenster

Im Flash-File befindet sich ein Hex-File, in dem sowohl die Adressen als auch die Werte stehen. Nun kann ein Parameter-File in ein Hex-File kopiert werden. Dazu entnimmt CANape die Adresse und den Wert aus der Parametersatzdatei und aktualisiert den Wert des Parameters an der entsprechenden Stelle des Hex-Files. Damit entsteht ein neues Hex-File, das die geänderten Parameterwerte enthält. Dieses Hex-File muss nun aber möglicherweise weitere Prozessschritte durchlaufen, um daraus wiederum eine flashbare Datei zu erhalten. Ein Problem sind dabei immer die Check-Summen, die das Steuergerät überprüft, um zu erkennen, ob es die Daten korrekt erhalten hat. Liegt dann das flashbare File vor, kann es in das Steuergerät geflasht werden und nach dem Reboot stehen die neuen Parameterwerte im Steuergerät zur Verfügung.

1.3 Austausch von DTOs – synchroner Datenaustausch

Wie in Bild 8 dargestellt ist, stehen zum Austausch von synchronen Mess- und Verstelldaten DTOs (Data Transfer Objects) bereit. Per DAQ werden Daten vom Slave – synchron zu internen Ereignissen (Events) – an den Master gesendet. Diese Kommunikation ist in zwei Phasen unterteilt: In einer Initialisierungsphase teilt der Master dem Slave mit, welche Daten der Slave bei verschiedenen Events versenden soll. Nach dieser Phase löst der Master beim Slave die Messung aus und die eigentliche Messphase beginnt. Ab diesem Zeitpunkt versendet der Slave die gewünschten Daten an den Master, der nur noch „zuhört“, bis er ein „Messungsstopp“ an den Slave schickt. Das Auslösen der Messdatenerfassung und die Versendung erfolgen durch Ereignisse im Steuergerät.

Per STIM versendet der Master Daten zum Slave. Auch diese Kommunikation besteht aus zwei Phasen:

In der Initialisierungsphase teilt der Master dem Slave mit, welche Daten er an den Slave senden wird. Nach dieser Phase schickt der Master die Daten an den Slave und der STIM-Prozessor speichert die Daten. Sobald ein entsprechendes STIM-Ereignis im Slave ausgelöst wird, werden die Daten in den Speicher der Anwendung übertragen.

1.3.1 Messverfahren: Polling versus DAQ

Bevor erläutert wird, wie ereignissynchrone, korrelierende Daten aus einem Slave heraus gemessen werden, erfolgt die kurze Beschreibung eines anderen Messverfahrens, des sogenannten Pollings. Es basiert nicht auf DTOs, sondern auf CTOs. Eigentlich müsste das Thema in einem anderen Kapitel beschrieben werden. Aber aus der Beschreibung des Pollings kann man sehr schön die Notwendigkeit der DTO-basierten Messung herleiten, sodass der kleine Exkurs an dieser Stelle sinnvoll ist.

Über das Kommando SHORT_UPLOAD kann der Master den Wert einer Messgröße vom Slave anfordern. Man spricht hierbei vom Polling. Dies ist der einfachste Fall einer Messung: den Messwert einer Messgröße zu dem Zeitpunkt versenden, an dem das SHORT_UPLOAD-Kommando empfangen und ausgeführt worden ist.

In folgendem Beispiel wird die Messgröße „Triangle“ aus dem Slave heraus gemessen:

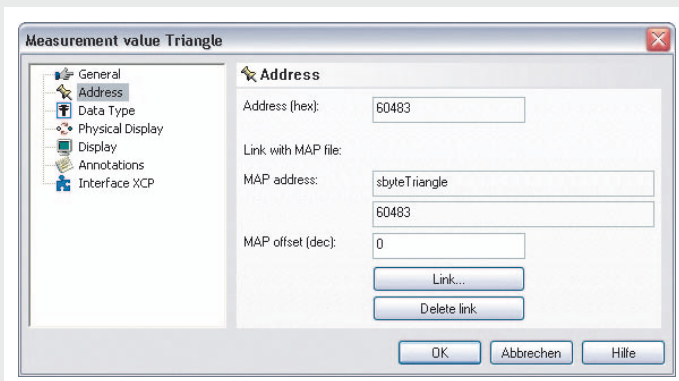


Bild 17:
Adressinformation
der Größe „Triangle“
aus der A2L-Datei

Die Adresse 0x60483 findet sich im CAN-Frame als eine Adresse mit fünf Bytes wieder: ein Byte für die Adress-Extension und vier Bytes für die eigentliche Adresse.

Time	Id	Name	Dir	Len	Data	Command	Device
0.2006s	554	CMD	Tx	8	F4 01 00 00 83 04 06 00	SHORT_UPLOAD size=1 ext=0 addr=00060483h	XCPsim
<ul style="list-style-type: none"> number of data elements to upload = 1 address extension = 0 address [HEX] = 00060483 							
0.20064s	555	RES	Rx	2	FF E8	Ok:SHORT_UPLOAD data=E8h	XCPsim
<ul style="list-style-type: none"> data [HEX]: = E8 							

Bild 18: Polling-Kommunikation im Trace-Fenster von CANape

In der XCP-Spezifikation ist für Polling festgelegt, dass der Wert jeder Messgröße einzeln abgefragt werden muss. Für jeden über Polling zu messenden Wert müssen zwei Botschaften über den Bus gehen: die Anfrage des Masters an den Slave und die Antwort des Slaves an den Master.

Neben dieser zusätzlichen Buslast gibt es noch einen weiteren Nachteil des Polling-Verfahrens: Bei der Abfrage von mehreren Daten legt der Anwender normalerweise Wert darauf, dass die Daten miteinander korrelieren. Mehrere Werte, die nacheinander mit Polling gemessen werden, müssen aber nicht in Korrelation zueinander stehen, d. h., sie müssen nicht aus dem gleichen Steuergeräte-Rechenzyklus stammen.

Polling ist somit nur begrenzt zur Messung geeignet, da es unnötig viel Datenverkehr produziert und die Messdaten nicht in Abhängigkeit von den Abläufen im Steuergerät berücksichtigt.

Ein optimiertes Messen muss also zwei Aufgaben lösen:

- > Bandbreitenoptimierung während der Messung
- > Sicherstellung der Datenkorrelation

Diese Aufgabe übernimmt das schon erwähnte DAQ-Verfahren. DAQ steht für Data Acquisition und wird über die Versendung von DTOs (Data Transfer Objects) vom Slave zum Master realisiert.

1.3.2 Das DAQ-Messverfahren

Das DAQ-Verfahren löst die beiden Probleme des Pollings folgendermaßen:

- > Die Korrelation der Messdaten wird dadurch erreicht, dass die Messwerterfassung an die Vorgänge (Events) im Steuergerät gekoppelt wird. Erst wenn sichergestellt ist, dass alle Berechnungen abgeschlossen sind, werden die Messwerte erfasst und übertragen.
- > Zur Verringerung der Buslast wird der Messvorgang in zwei Phasen unterteilt: In einer Konfigurationsphase teilt der Master dem Slave mit, an welchen Werten er interessiert ist, und in der zweiten Phase erfolgt nur noch die Übertragung der Messdaten des Slaves an den Master.

Wie kann nun die Erfassung von Messwerten an Vorgänge im Steuergerät gekoppelt werden? In Bild 19 ist der Zusammenhang zwischen den Kalkulationszyklen im Steuergerät und der Wertentwicklung der Größen X und Y nochmals leicht verändert dargestellt.

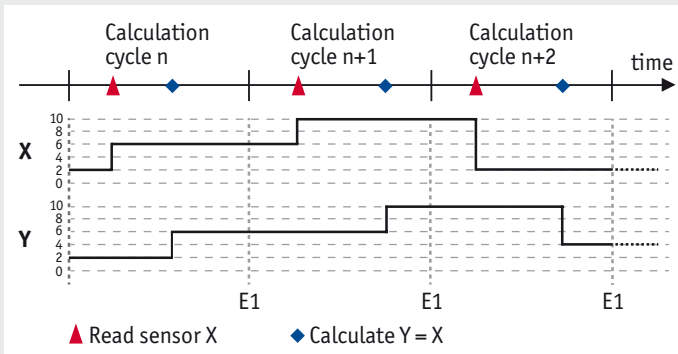


Bild 19:
Ereignisse im
Steuergerät

Betrachten wir den Ablauf im Steuergerät: Beim Erreichen des Events E1 (= Ende des Berechnungszyklus) wurden alle Größen erfasst und die Kalkulation ist erfolgt. Also müssen zu diesem Zeitpunkt alle Werte zueinander passen und korrelieren. Das bedeutet, dass wir ein eventsynchrones Messverfahren anwenden. Genau das wird mit Hilfe des DAQ-Mechanismus realisiert: Erreicht der Algorithmus im Slave das Ereignis „Berechnungszyklus abgeschlossen“, sammelt der XCP-Slave die Werte der Messgrößen ein, speichert sie in einem Zwischenspeicher und versendet sie an den Master. Vorausgesetzt, der Slave weiß, welche Größen bei welchem Ereignis gemessen werden sollen.

Ein Event ist nicht zwangsläufig ein zyklisches, zeitlich äquidistantes Ereignis, sondern kann im Falle eines Motorsteuergerätes z. B. auch winkelsynchron sein. Somit ist der Zeitabstand zwischen zwei Ereignissen von der Motordrehzahl abhängig. Ein singuläres Ereignis, wie die Betätigung eines Tasters durch den Fahrer, ist ebenfalls ein Event, das in keiner Weise zeitlich äquidistant ist.

Die Auswahl der Signale erfolgt durch den Anwender. Neben dem eigentlichen Messobjekt muss er den zugrunde liegenden Event für die Messgröße auswählen. Die Events ebenso wie die Zuordnungsmöglichkeiten der Messobjekte zu den Events müssen in der A2L-Datei hinterlegt sein.

Event List					
General Expert settings					
Event name	Event number	Rate	Unit	Priority	DAQ/STIM
Key T	00h	0	Not cyclic	0	DAQ
10 ms	01h	10	ms	0	DAQ/STIM
100ms	02h	100	ms	0	DAQ/STIM
1ms	03h	1	ms	0	DAQ/STIM
FilterBypassDaq	04h	0	Not cyclic	0	DAQ/STIM
FilterBypassSt	05h	0	Not cyclic	0	DAQ/STIM

Bild 20:
Eventdefinition
in einer A2L

Im Normalfall macht es keinen Sinn, einen Messwert mehreren Events gleichzeitig zuordnen zu können. In aller Regel wird eine Größe nur innerhalb eines einzigen Zyklus verändert (z. B. nur im 10-ms-Raster) und nicht in mehreren (z. B. im 10-ms- und gleichzeitig im 100-ms-Raster).

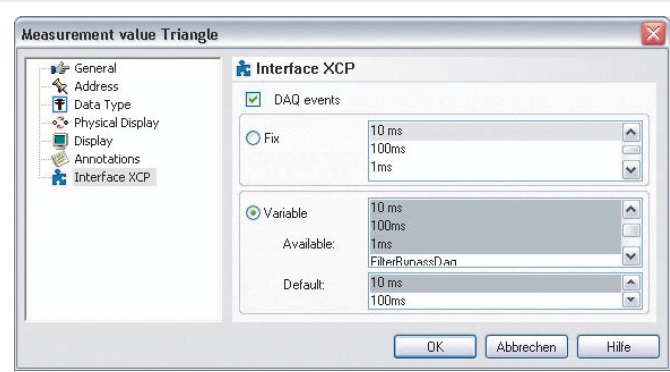


Bild 21:
Zuordnung von
„Triangle“ zu den
möglichen Events
in der A2L

Bild 21 zeigt, dass die Größe „Triangle“ grundsätzlich mit den Events 1 ms, 10 ms und 100 ms gemessen werden kann. Die Default-Einstellung beträgt 10 ms.

Die Zuordnung der Messgrößen zu den Events im Steuergerät erfolgt bei der Konfiguration der Messung durch den Anwender.

No.	Type	Active	Name	Measurement mode	Default_Recorder
11		<input checked="" type="checkbox"/>	Triangle	10 ms	<input checked="" type="checkbox"/>
12		<input checked="" type="checkbox"/>	byte3	100ms	<input checked="" type="checkbox"/>
13		<input checked="" type="checkbox"/>	Shifter_B0	1ms	<input checked="" type="checkbox"/>

Bild 22:
Auswahl der Events
(Measurement mode)
pro Messgröße

Nach der Konfiguration der Messsignale startet der Anwender die Messung. Der XCP-Master fasst die gewünschten Messgrößen in sogenannten DAQ-Listen zusammen. Dabei werden die Messsignale den jeweils ausgewählten Events zugeordnet. Diese Konfigurationsinformation wird vor dem eigentlichen Messbeginn an den Slave gesendet. Der Slave weiß danach, welche Adressen er beim Erreichen welches Ereignisses auslesen und übertragen soll. Diese Aufteilung der Messung in eine Konfigurations- und eine Messphase wurde ganz am Anfang des Kapitels bereits erwähnt.

Damit sind beide beim Polling auftretenden Probleme gelöst: sowohl die optimierte Nutzung der Bandbreite, da der Master während der Messung nicht mehr jeden Wert einzeln anfragen muss, als auch die Korrelation der Messdaten.

[5] Trace								[-] X
Time	Id	Name	Dir	Len	Data	Command	Device	
-0.251873s	554	CMD	Tx	4	DE 02 01 00	START_STOP_DAQ_LIST mode=02h daq=1	XCpsim	
-0.251844s	555	RES	Rx	2	FF 01	Ok:START_STOP_DAQ_LIST firstPid=01h	XCpsim	
-0.25164s	554	CMD	Tx	8	E0 10 02 00 03 00 01 00	SET_DAQ_LIST_MODE mode=10h daq=2 even...	XCpsim	
-0.251609s	555	RES	Rx	1	FF	Ok:SET_DAQ_LIST_MODE	XCpsim	
-0.251565s	554	CMD	Tx	4	DE 02 02 00	START_STOP_DAQ_LIST mode=02h daq=2	XCpsim	
-0.251527s	555	RES	Rx	2	FF 02	Ok:START_STOP_DAQ_LIST firstPid=02h	XCpsim	
0.000389s	554	CMD	Tx	1	DC	GET_DAQ_CLOCK	XCpsim	
0.000415s	555	RES	Rx	8	FF 02 70 61 00 00 00 00	Ok:GET_DAQ_CLOCK timestamp=0	XCpsim	
0.00046s	554	CMD	Tx	2	DD 01	START_STOP_SYNCH mode=01h	XCpsim	
0.00049s	555	RES	Rx	1	FF	Ok:START_STOP_SYNCH	XCpsim	
0.004179s	Measurement started at							
0.0088s	555	DAQ	Rx	7	00 02 53 00 00 00 04	Data	XCpsim	
0.00882s	555	DAQ	Rx	7	00 02 54 00 00 00 04	Data	XCpsim	
0.00884s	555	DAQ	Rx	7	00 02 54 00 00 00 04	Data	XCpsim	
0.00886s	555	DAQ	Rx	7	00 02 54 00 00 00 04	Data	XCpsim	
0.00888s	555	DAQ	Rx	7	00 02 54 00 00 00 04	Data	XCpsim	
0.0089s	555	DAQ	Rx	7	00 02 55 00 00 00 04	Data	XCpsim	
0.00892s	555	DAQ	Rx	7	00 02 55 00 00 00 04	Data	XCpsim	
0.00893s	555	DAQ	Rx	7	00 02 55 00 00 00 04	Data	XCpsim	
0.00905s	555	DAQ	Rx	7	00 02 55 00 00 00 04	Data	XCpsim	

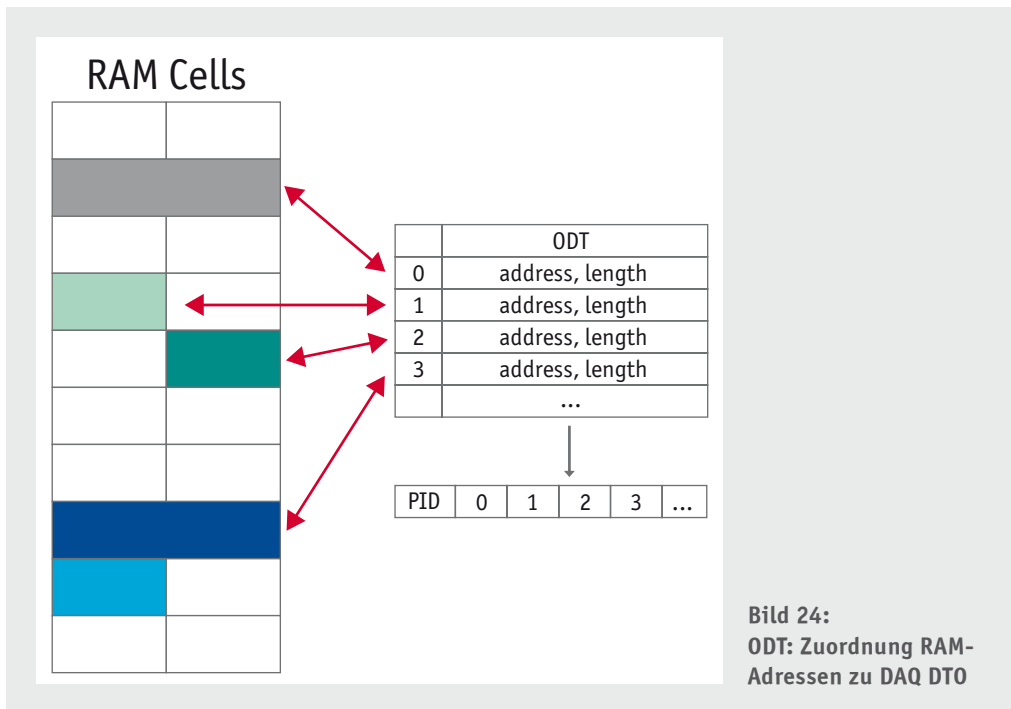
Bild 23: Auszug aus dem CANape Trace-Fenster einer DAQ-Messung

Bild 23 verdeutlicht eine Command-Response-Kommunikation (farbig hinterlegt) zwischen Master und Slave (sie ist insgesamt deutlich umfangreicher und wird aus Platzgründen hier nur teilweise dargestellt). Dabei handelt es sich um die Übertragung der DAQ-Konfiguration an den Slave. Danach erfolgt das Auslösen des Messungsstarts und der Slave sendet die angeforderten Werte, während der Master nur noch zuhört.

Bisher wurden die Auswahl der Signale anhand ihres Namens und die Zuordnung zu einem Messereignis beschrieben. Wie aber erfolgt genau die Übergabe der Konfiguration an den XCP-Slave?

Betrachten wir das Problem von der Seite des Speicheraufbaus im Steuergerät: Der Anwender hat Signale ausgewählt und möchte sie messen. Damit für das Versenden eines Signalwertes nicht eine komplette Botschaft verwendet werden muss, werden die Signale vom Slave zu Botschaftspaketen zusammengefügt. Die Definition der Zusammensetzung erstellt der Slave nicht selbstständig, sonst könnte der Master beim Erhalten der Botschaften die Daten nicht interpretieren. Der Slave empfängt daher vom Master die Anweisung, wie er die Werte auf die Botschaften verteilen soll.

Die Zuordnung, wie der Slave die Bytes zu Botschaften zusammensetzen soll, erfolgt über sogenannte Object Description Tables (ODTs). Um ein Messobjekt eindeutig zu identifizieren, sind die Adresse und die Objektlänge wichtig. Somit erhält man mit einer ODT die Zuordnung der RAM-Inhalte aus dem Slave zum Aufbau einer Botschaft auf dem Bus. Gemäß dem Kommunikationsmodell wird diese Botschaft als DAQ DTO (Data Transfer Object) übertragen.



Genauer gesagt, referenziert ein Eintrag in einer ODT-Liste auf einen Speicherbereich im RAM über die Adresse und die Länge des Objektes.

Nach dem Empfang des Messstartkommandos wird irgendwann ein Event erreicht, das mit einer Messung verknüpft ist. Der XCP-Slave beginnt mit der Erfassung der Daten. Er fügt die Einzelobjekte zu Paketen zusammen und sendet sie auf den Bus. Der Master liest die Busbotschaft und kann die einzelnen Daten interpretieren, da er die Zuordnung der Einzelobjekte zu den Paketen selbst definiert hat und somit die Zusammenhänge kennt.

Nun hat aber jedes Paket eine maximale Anzahl an Nutz-Bytes, die vom genutzten Transportmedium abhängt. Im Falle von CAN sind das sieben Bytes. Sollen mehr Daten gemessen werden, reicht eine ODT nicht aus. Wenn für die Übertragung der Messdaten zwei oder mehr ODTs verwendet werden müssen, dann muss sowohl der Slave die Daten in die richtige ODT kopieren als auch der Master die empfangenen ODTs eindeutig identifizieren können. Werden mehrere Messraster des Steuergerätes verwendet, muss zudem der Zusammenhang zwischen ODT und Messraster eindeutig zuordenbar sein.

Im XCP-Protokoll sind die ODTs zu DAQ-Listen zusammengefügt. Jede DAQ-Liste enthält eine Reihe von ODTs und ist einem Event zugeordnet.

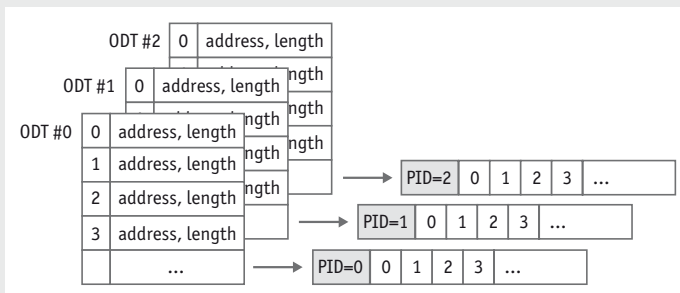


Bild 25:
DAQ-Liste mit drei
ODTs

Verwendet der Anwender z. B. zwei Messraster (= zwei unterschiedliche Events im Steuergerät), kommen auch zwei DAQ-Listen zum Einsatz. Pro verwendeten Event wird eine DAQ-Liste benötigt. Jede DAQ-Liste enthält die Einträge bzgl. der ODTs und jede ODT beinhaltet die Referenzen zu den Werten in den RAM-Zellen.

DAQ-Listen unterteilen sich in statische, vordefinierte und dynamische.

Statische DAQ-Listen:

Wenn die DAQ-Listen und ODT-Tabellen fest im Steuergerät definiert sind, wie man das von CCP her kennt, spricht man von statischen DAQ-Listen. Dabei ist nicht festgelegt, welche Messgrößen in den ODT-Listen stehen, sondern der Rahmen, der gefüllt werden kann (im Unterschied dazu siehe vordefinierte DAQ-Listen).

Bei statischen DAQ-Listen werden die Definitionen im Steuergeräte-Code festgelegt und in der A2L beschrieben. Bild 26 zeigt einen Ausschnitt aus einer A2L, in der statische DAQ-Listen definiert sind:

DAQ List				
General Extended settings Expert settings				
General				
DAQ configuration: static				
DAQ list No.	Event channel	DAQ Id.	MAX ODT	MAX ODT ENTRY
0	10 ms	DTO_ID 2	2	2
1	100ms	DTO_ID 4	4	4

Bild 26:
Statische DAQ-Listen

Im obigen Beispiel gibt es eine DAQ-Liste mit der Nummer 0, die einem 10-ms-Event zugeordnet ist und maximal zwei ODTs tragen kann. Die DAQ-Liste mit der Nummer 1 verfügt über vier ODTs und ist mit dem Event 100 ms verknüpft.

Die A2L entspricht dem Inhalt des Steuergerätes. Mit dem Herunterladen der Anwendung in das Steuergerät ist im Falle von statischen DAQ-Listen die Anzahl der DAQ-Listen und der jeweils darin enthaltenen ODT-Listen festgelegt. Versucht der Anwender nun mehr Signale mit einem Event zu messen, als in die zugeordnete DAQ-Liste passen, wird der Slave im Steuergerät die Anforderungen nicht erfüllen können und der Konfigurationsversuch wird mit einem Fehler abgebrochen. Dass die andere DAQ-Liste noch komplett zur Verfügung steht und damit eigentlich noch Übertragungskapazität frei ist, spielt keine Rolle.

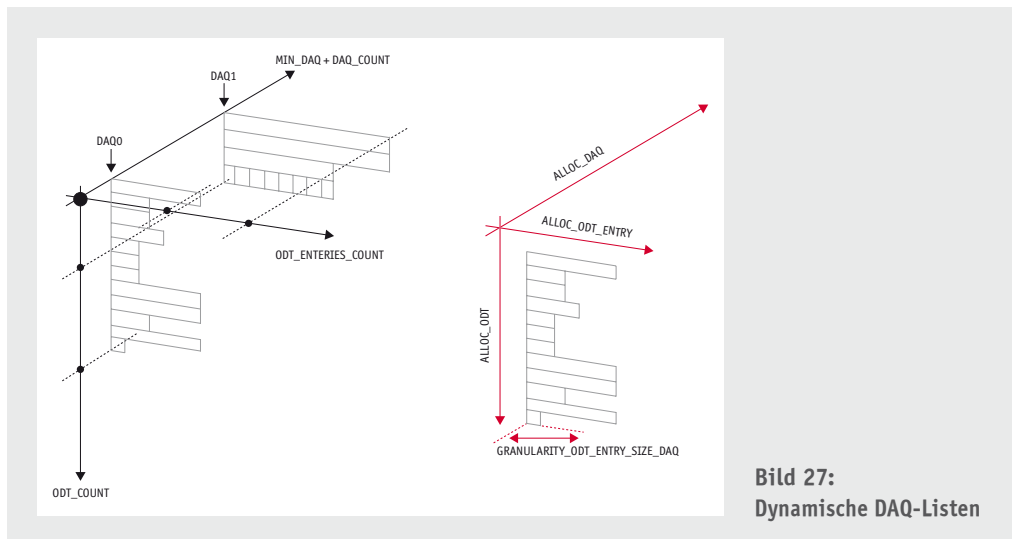
Vordefinierte DAQ-Listen:

Im Steuergerät können auch komplett vordefinierte DAQ-Listen bestimmt werden. Allerdings kommt dieses Verfahren in Steuergeräten aufgrund der fehlenden Flexibilität des Anwenders praktisch nicht zum Einsatz. Anders sieht es bei Analog-Messsystemen aus, die ihre Daten per XCP übertragen: Hier ist die Flexibilität nicht notwendig, da der physikalische Aufbau des Messsystems über die Lebensdauer hinweg gleich bleibt.

Dynamische DAQ-Listen:

Eine Besonderheit des XCP-Protokolls sind die dynamischen DAQ-Listen. Nicht die absoluten Größen der DAQ- und ODT-Listen sind fest im Steuergeräte-Code definiert, sondern lediglich die Größe des Speicherbereichs, der für die DAQ-Listen genutzt werden kann. Der Vorteil besteht darin, dass das Messwerkzeug beim Zusammenfassen der DAQ-Listen mehr Spielraum besitzt und die Struktur der DAQ-Listen dynamisch verwalten kann.

Speziell für diese dynamische Verwaltung stehen bei XCP verschiedene Funktionen wie z. B. `ALLOC_ODT` zur Verfügung, mit denen der Master die Struktur einer DAQ-Liste im Slave festlegen kann.



Bei der Zusammenstellung der DAQ-Listen muss der Master unterscheiden können, ob dynamische oder statische DAQ-Listen genutzt werden, wie die Größen und Strukturen der DAQ-Listen aussehen etc.

1.3.3 Das STIM-Verstellverfahren

Im Kapitel über den Austausch von CTOs wurde das XCP-Verstellverfahren bereits vorgestellt. Diese Art der Verstellung ist in jedem XCP-Treiber vorhanden und bildet die Basis für das Verstellen von Objekten im Steuergerät. Allerdings existiert keine Synchronisierung zwischen dem Versenden eines Verstellbefehls und einem Ereignis im Steuergerät.

Im Gegensatz dazu basiert die Verwendung von STIM nicht auf dem Austausch von CTOs, sondern auf der Nutzung von DTOs mit einer zu einem Ereignis im Slave synchronisierten Kommunikation. Der Master muss also wissen, auf welche Ereignisse im Slave er überhaupt synchronisieren kann. Diese Informationen müssen ebenfalls in der A2L vorliegen.

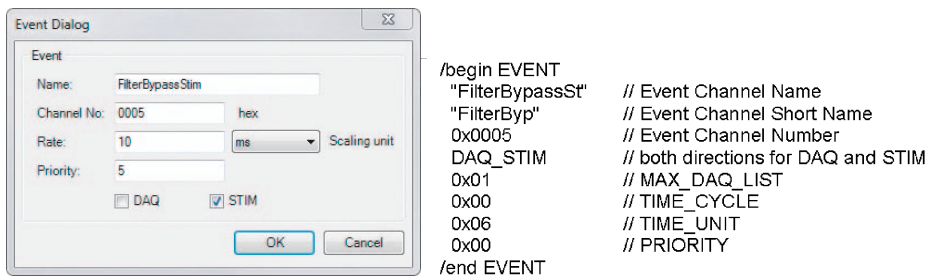


Bild 28: Event für DAQ und STIM

Sendet der Master Daten per STIM an den Slave, muss dem XCP-Slave die Stelle in den Paketen, an denen die Verstellgrößen vorliegen, bekannt sein. Dabei werden die gleichen Mechanismen wie für die DAQ-Listen genutzt.

1.3.4 XCP-Paket-Adressierung für DAQ und STIM

Am Anfang des Kapitels wurde bereits die Adressierung der XCP-Pakete diskutiert. Nachdem nun die Begriffe DAQ, ODT und STIM eingeführt sind, wird die XCP-Paket-Adressierung detaillierter dargestellt.

Während bei der Übertragung von CTOs die Nutzung einer PID völlig ausreicht, um ein Paket eindeutig zu identifizieren, genügt dies bei der Übertragung von Messdaten nicht mehr. Das folgende Bild gibt einen Überblick über die möglichen Adressierungen, die bei DTOs zum Einsatz kommen können:

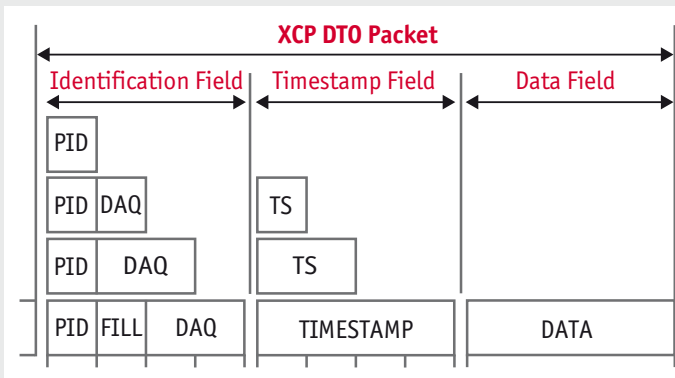


Bild 29:
Aufbau XCP-Paket bei
DTO-Übertragungen

Übertragungstyp „absolute ODT-Nummern“

Absolut bedeutet, dass während der gesamten Kommunikation – also über alle DAQ-Listen hinweg – die ODT-Nummern eindeutig sind. Insofern setzt die Nutzung von absoluten ODT-Nummern einen Transformationsschritt voraus, in dem eine sogenannte „FIRST_PID für die DAQ-Liste“ verwendet wird.

Fängt eine DAQ-Liste mit der PID j an, so hat die PID des ersten Paketes den Wert j , das zweite Paket die PID $j + 1$, das dritte Paket PID $j + 2$ usw. Dabei muss natürlich vom Slave darauf geachtet werden, dass die Summe aus FIRST_PID + relative ODT-Nummer unterhalb des PIDs der nächsten DAQ-Liste bleibt.

DAQ-Liste: $0 \leq \text{PID} \leq k$

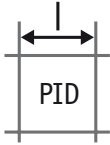
DAQ-Liste: $k + 1 \leq \text{PID} \leq m$

DAQ-Liste: $m + 1 \leq \text{PID} \leq n$

usw.

In diesem Fall ist das Identifikationsfeld sehr einfach:

Identification Field



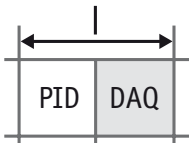
| absolute ODT number

Bild 30:
Identifikationsfeld
mit absoluten
ODT-Nummern

Übertragungstyp „relative ODT-Nummern und absolute DAQ-Listen-Nummern“

In diesem Fall werden sowohl die DAQ-Listen-Nummer als auch die ODT-Nummer im Identifikationsfeld übertragen. Spielraum gibt es aber noch bei der Anzahl der Bytes, die für die Informationen zur Verfügung stehen:

Identification Field

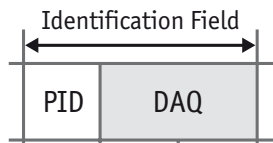


| absolute DAQ List number
| relative ODT number

Bild 31:
ID-Feld mit relativen
ODT- und absoluten
DAQ-Nummern
(ein Byte)

In der Abbildung steht für die DAQ-Nummer und die ODT-Nummer jeweils ein Byte bereit.

Maximal kann die Anzahl der DAQ-Listen mit Hilfe von zwei Bytes übertragen werden:



| absolute DAQ list number
| relative ODT number

Bild 32:
ID-Feld mit relativen
ODT- und absoluten
DAQ-Nummern
(zwei Bytes)

Kommt eine Übermittlung mit drei Bytes nicht in Frage, kann unter Zuhilfenahme eines Füll-Bytes auch mit vier Bytes gearbeitet werden:

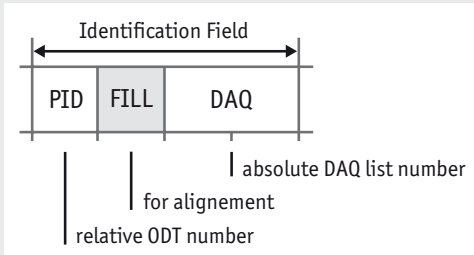


Bild 33:
ID-Feld mit relativen ODT- und absoluten DAQ-Nummern sowie Füll-Byte (gesamt vier Bytes)

Wie erfährt nun der XCP-Master, welches Verfahren der Slave verwendet? Zum einen durch den Eintrag in der A2L und zum anderen durch die Abfrage an den Slave, welche Kommunikationsvariante implementiert ist.

Mit der Antwort auf die Anfrage `GET_DAQ_PROCESSOR_INFO` wird das `DAQ_KEY_BYTE` gesetzt, mit dessen Hilfe der Slave den Master darüber informiert, welcher Übertragungstyp zum Einsatz kommt. Soll nicht nur DAQ, sondern auch STIM eingesetzt werden, muss der Master für STIM das gleiche Verfahren nutzen, das der Slave für DAQ verwendet.

1.3.5 Bypassing = DAQ + STIM

Bypassing kann durch die gemeinsame Nutzung von DAQ und STIM realisiert werden (siehe Bild 8) und stellt eine spezielle Form einer Rapid-Prototyping-Lösung dar. Für ein tieferes Verständnis sind jedoch weitere Details notwendig, sodass dieses Verfahren erst in Kapitel 4.5 „Bypassing“ erläutert wird.

1.4 XCP-Transportschichten

Eine Hauptanforderung an das Design des Protokolls bestand in der Unterstützung unterschiedlicher Transportschichten. Folgende Schichten sind zum Zeitpunkt der Erstellung dieses Dokumentes definiert: XCP on CAN, FlexRay, Ethernet, SxI und USB. Die Bussysteme CAN, LIN und FlexRay werden auf der Vector-E-Learning-Plattform erklärt, ebenso finden Sie dort eine Einführung in AUTOSAR, siehe Website www.vector-elearning.com.

1.4.1 CAN

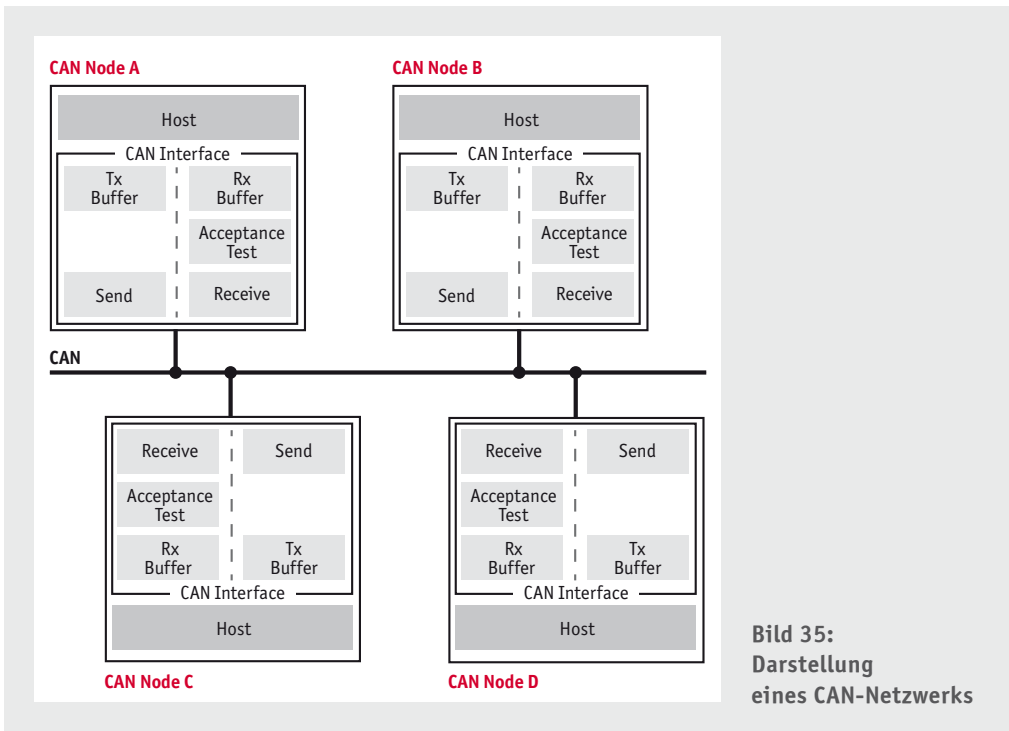
XCP ist als Nachfolgeprotokoll des CAN Calibration Protocols (CCP) entwickelt worden und muss deshalb zwingend den Anforderungen des CAN-Busses gerecht werden. Die Kommunikation über den CAN-Bus definiert sich durch die dazugehörige Beschreibungsdatei. Üblicherweise kommt das Format DBC zum Einsatz, vereinzelt bereits das AUTOSAR-Format ARXML.

Eine CAN-Botschaft wird durch einen eindeutigen CAN-Identifizier gekennzeichnet. In der Beschreibungsdatei ist die Kommunikationsmatrix festgelegt: Wer sendet welche Botschaft und wie sind die acht Nutz-Bytes des CAN-Busses belegt. Das folgende Bild verdeutlicht den Vorgang:

Data Frame	CAN Node A	CAN Node B	CAN Node C	CAN Node D
ID=0x12	Sender	Receiver		
ID=0x34		Sender	Receiver	Receiver
ID=0x52		Receiver		Sender
ID=0x67	Receiver	Receiver	Sender	Receiver
ID=0xB4	Receiver		Sender	
ID=0x3A5	Sender	Receiver	Receiver	Receiver

Bild 34:
Festlegung, welcher Busteilnehmer welche Botschaft sendet

Die Botschaft mit der ID 0x12 wird vom CAN-Knoten A gesendet und alle anderen Knoten auf dem Bus empfangen diese Nachricht. Die CAN-Knoten C und D kommen im Rahmen der Akzeptanzprüfung zum Ergebnis, dass sie die Botschaft nicht benötigen, und verwerfen diese. CAN-Knoten B hingegen stellt fest, dass die weiter oben liegenden Schichten die Nachricht benötigen, und stellt sie über den Empfangspuffer zur Verfügung. Die CAN-Knoten sind folgendermaßen miteinander verknüpft:



Die XCP-Botschaften sind nicht in der Kommunikationsmatrix beschrieben! Werden, beispielsweise mit Hilfe von XCP, Messdaten über dynamische DAQ-Listen aus dem Slave gesendet, erfolgt die Zusammenstellung der Botschaften in Abhängigkeit von den ausgewählten Signalen durch den Anwender. Ändert sich die Signalzusammensetzung, ändert sich auch der Botschaftsinhalt. Dennoch gibt es einen Zusammenhang zwischen der Kommunikationsmatrix und XCP: Zur Übertragung der XCP-Botschaften über CAN werden CAN-Identifizier benötigt. Um nun möglichst wenige CAN-Identifizier zu belegen, beschränkt sich die XCP-Kommunikation auf die Verwendung von nur zwei CAN-Identifiern, die nicht in der DBC für die „normale“ Kommunikation genutzt werden. Ein Identifizier wird benötigt, um Informationen vom Master zum Slave zu senden, der andere dient dem Slave zur Antwort an den Master.

Der Auszug aus dem CANape Trace-Fenster zeigt unter der Spalte „ID“ die verwendeten CAN-Identifizier. In diesem Beispiel werden nur zwei unterschiedliche Identifizier eingesetzt: 554 als ID für die Botschaft vom Master zum Slave (Richtung/Direction Tx) und 555 für die Versendung der Botschaften des Slaves an den Master (Richtung/Direction Rx).

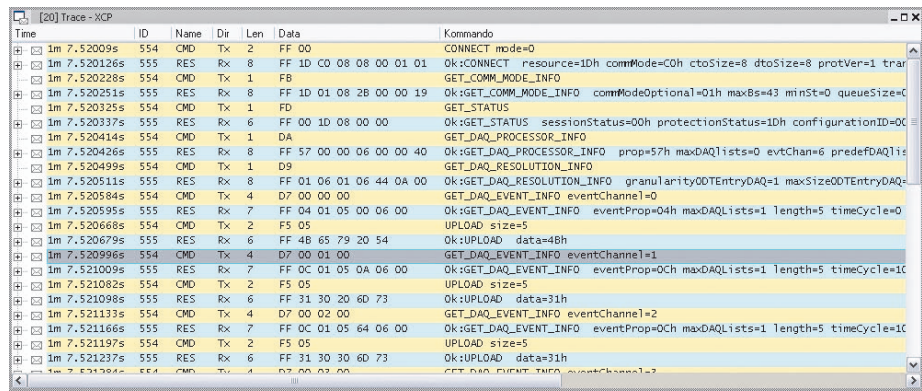


Bild 36: Beispiel einer XCP-on-CAN-Kommunikation

Die gesamte XCP-Kommunikation läuft in diesem Beispiel über die beiden CAN-Identifizier 554 und 555 ab. Diese beiden IDs dürfen in diesem Netzwerk nicht für andere Zwecke vergeben sein.

Der CAN-Bus überträgt maximal acht Nutz-Bytes pro Botschaft. Im Falle von XCP benötigen wir aber eine Information über den verwendeten Befehl bzw. die versendete Antwort. Dafür dient das erste Byte der CAN-Nutzdaten. Somit stehen für den Transport von Nutzdaten pro CAN-Botschaft sieben Bytes zur Verfügung.

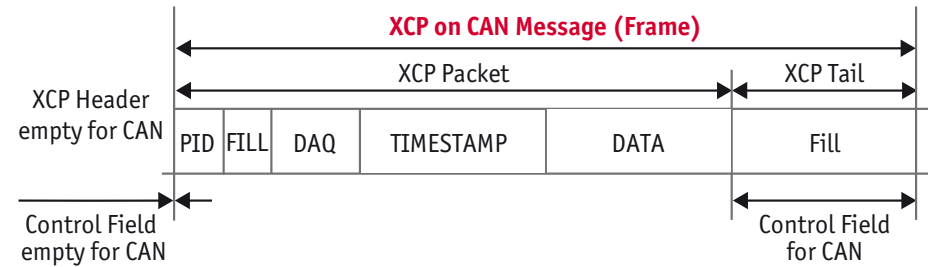


Bild 37: Darstellung einer XCP-on-CAN-Botschaft

In CANape finden Sie eine XCP-on-CAN-Demo mit dem virtuellen Steuergerät XCPsim. Weitere Details über den Standard erfahren Sie in ASAM XCP on CAN Teil 3 Transport Layer Specification 1.1.0.

1.4.2 FlexRay

Ein Grundgedanke bei der Entwicklung von FlexRay war die Realisierung eines redundanten Systems mit deterministischem Zeitverhalten. Die Verbindungsredundanz wurde durch die Nutzung von zwei Kanälen geschaffen: Kanal A und Kanal B. Werden mehrere FlexRay-Knoten (= Steuergeräte) redundant zusammengeschaltet und fällt eine Strecke aus, können die Knoten auf den anderen Kanal umschalten und somit eine Verbindungsredundanz schaffen.

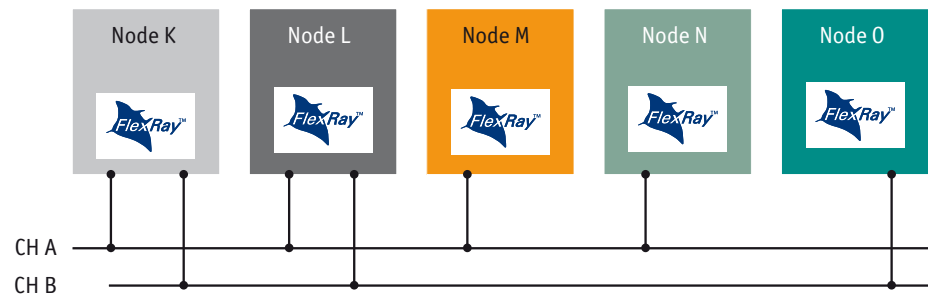


Bild 38: Die Knoten K und L sind redundant miteinander verbunden

Das deterministische Verhalten wird erreicht, indem Daten innerhalb festgelegter Zeit-Slots übertragen werden. Dabei wird auch definiert, welcher Teilnehmer in welchem Zeit-Slot welche Inhalte sendet. Diese Zeit-Slots werden zu einem Zyklus zusammengefasst. Dabei wiederholen sich die Zyklen, solange der Bus aktiv ist. Die Zusammenstellung der Zeit-Slots und der Übertragungsinhalte (wer sendet was zu welchem Zeitpunkt) nennt man Scheduling.

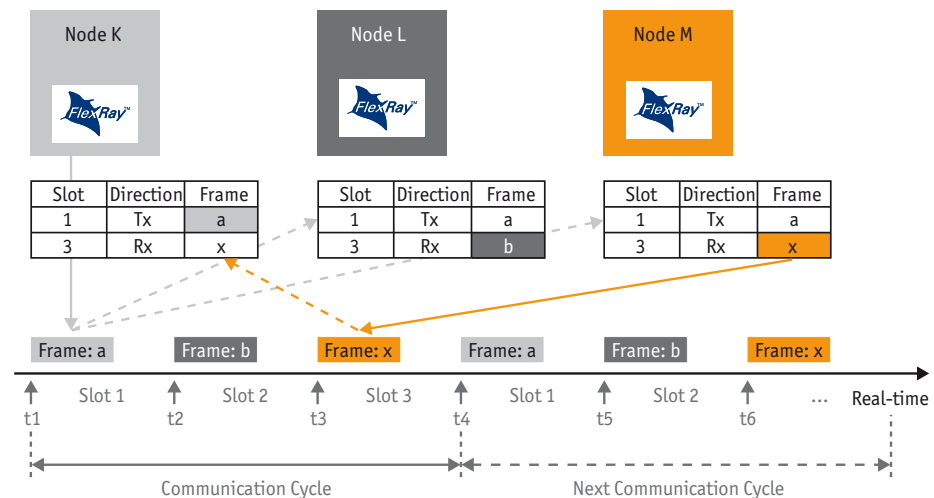


Bild 39: Kommunikation per Slot-Definition

Im ersten Kommunikationszyklus sendet Knoten K die Nachricht a im Slot 1. In der Software der Knoten L und M ist das Scheduling ebenfalls hinterlegt. Deshalb wird der Inhalt der Nachricht a an die jeweils höheren Kommunikationsschichten weitergegeben.

Das Scheduling wird in einer Beschreibungsdatei zusammengefasst. Diese ist allerdings keine DBC-Datei, wie im Falle von CAN, sondern eine FIBEX-Datei. FIBEX steht für „Field Bus Exchange Format“ und könnte auch für andere Bussysteme verwendet werden. Die derzeitige Nutzung liegt aber praktisch allein bei der Beschreibung des FlexRay-Busses. FIBEX ist ein XML-Format und die XCP-on-FlexRay-Spezifikation bezieht sich auf die FIBEX-Version 1.1.5 und die FlexRay-Spezifikation Version 2.1.

	Slot	ECU	Channel	Cycles									
				0	1	2	3	4	5	6	...	63	
Static Segment	1	Node K	A	b [rep : 1]	b [rep : 1]	b [rep : 1]	b [rep : 1]	b [rep : 1]	b [rep : 1]	b [rep : 1]		b [rep : 1]	
			B	b [rep : 1]	b [rep : 1]	b [rep : 1]	b [rep : 1]	b [rep : 1]	b [rep : 1]	b [rep : 1]		b [rep : 1]	
	2	Node M	A	c [rep : 4]	x [rep : 2]	y [rep : 4]	x [rep : 2]	c [rep : 4]	x [rep : 2]	y [rep : 4]		x [rep : 2]	
			B										
	3	Node L	A	a [rep : 1]	a [rep : 1]	a [rep : 1]	a [rep : 1]	a [rep : 1]	a [rep : 1]	a [rep : 1]		a [rep : 1]	
			B	d [rep : 1]	d [rep : 1]	d [rep : 1]	d [rep : 1]	d [rep : 1]	d [rep : 1]	d [rep : 1]		d [rep : 1]	
Dynamic Segment	4	Node L	A	n [rep : 1]	n [rep : 1]	n [rep : 1]	n [rep : 1]	n [rep : 1]	n [rep : 1]	n [rep : 1]		n [rep : 1]	
			Node O	B	m [rep : 1]	m [rep : 1]	m [rep : 1]	m [rep : 1]	m [rep : 1]	m [rep : 1]	m [rep : 1]		m [rep : 1]
	5	Node N	A	r [rep : 1]	r [rep : 1]	r [rep : 1]	r [rep : 1]	r [rep : 1]	r [rep : 1]	r [rep : 1]		r [rep : 1]	
			B										
	6	Node K	A										
			B	o [rep : 1]	o [rep : 1]	o [rep : 1]	o [rep : 1]	o [rep : 1]	o [rep : 1]	o [rep : 1]		o [rep : 1]	
	7	Node M	A		t [rep : 2]	p [rep : 4]	t [rep : 2]		t [rep : 2]	p [rep : 4]		t [rep : 2]	
			B										
		Node L	A	u [rep : 4]				u [rep : 4]					
			B			v [rep : 8]							
	Node O	A											
		B		w [rep : 4]				w [rep : 4]					

Bild 40: Darstellung einer FlexRay-Kommunikationsmatrix

Infolge der Entwicklung von AUTOSAR-Lösungen wurde ein weiteres Format zur Beschreibung der Bus-Kommunikation definiert: das AUTOSAR Description File, das im XML-Format vorliegt. In der AUTOSAR-4.0-Spezifikation ist der Aspekt der XCP-on-FlexRay-Definition mit berücksichtigt. Zur Zeit der Drucklegung des Buches ist diese Spezifikation jedoch noch nicht endgültig verabschiedet und wird daher nicht weiter betrachtet.

Aufgrund weiterer Eigenschaften des FlexRay-Busses genügt die Angabe der Slot-Nummer als Referenz zum Inhalt alleine nicht aus. Beispielsweise wird ein Multiplexing unterstützt: Wiederholt sich ein Zyklus, muss nicht jedes Mal der gleiche Inhalt gesendet werden. Über ein Multiplexing kann etwa festgelegt werden, dass eine Information immer nur bei jedem zweiten Durchlauf des Slots verschickt wird.

Anstelle der reinen Slot-Nummer werden die sogenannten „FlexRay Data Link Layer Protocol Data Unit Identifier“ (FLX_LPDU_ID) verwendet, die man sich als eine Art verallgemeinerte Slot-ID vorstellen kann. Um eine solche LPDU zu beschreiben, werden vier Angaben benötigt:

- > FlexRay Slot Identifier (FLX_SLOT_ID)
- > Cycle Counter Offset (OFFSET)
- > Cycle Counter Repetition (CYCLE_REPETITION)
- > FlexRay Channel (FLX_CHANNEL)

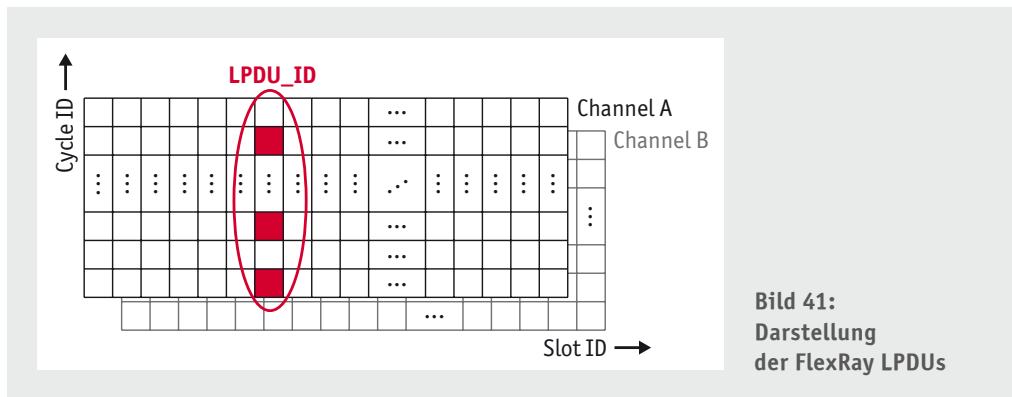


Bild 41:
Darstellung
der FlexRay LPDUs

Das Scheduling hat auch Auswirkungen auf die Nutzung von XCP on FlexRay, da es genau festlegt, was gesendet wird. Bei XCP kann das nicht ohne Weiteres definiert werden. Welche Messdaten versendet werden, legt der Anwender durch die Zusammenstellung der Signale erst zur Laufzeit der Messung fest. Also kann nur geregelt werden, welcher Aspekt der XCP-Kommunikation in welcher LPDU genutzt werden kann: CTO oder DTO vom Master zum Slave oder vom Slave zum Master.

Folgendes Beispiel erläutert den Ablauf: In Slot n darf der XCP-Master ein Kommando (CMD) versenden und im Slot $n + 2$ antwortet Slave A mit der Response (RES). XCP-on-FlexRay-Botschaften werden immer mit Hilfe von LPDUs festgelegt.

Für den Zugriff auf steuergeräteinterne Größen wird die A2L-Beschreibungsdatei benötigt, in der die Objekte mit der entsprechenden Adresse im Steuergerät definiert sind. Darüber hinaus ist die FIBEX-Datei erforderlich, damit der XCP-Master weiß, mit welchen LPDUs er senden darf und auf welchen LPDUs die XCP-Slaves ihre Antworten verschicken. Erst in der Kombination, d. h. der Herstellung der Referenz einer A2L-Datei auf eine FIBEX-Datei, kann die Kommunikation zwischen XCP-Master und XCP-Slave(s) funktionieren.

Auszug aus einer A2L mit XCP-on-FlexRay-Bedeutung:

```
...
/begin XCP_ON_FLX
...
„XCPsim.xml“
„Cluster_1“
...
```

Hierbei ist „XCPsim.xml“ die Referenz aus der A2L- zur FIBEX-Datei.

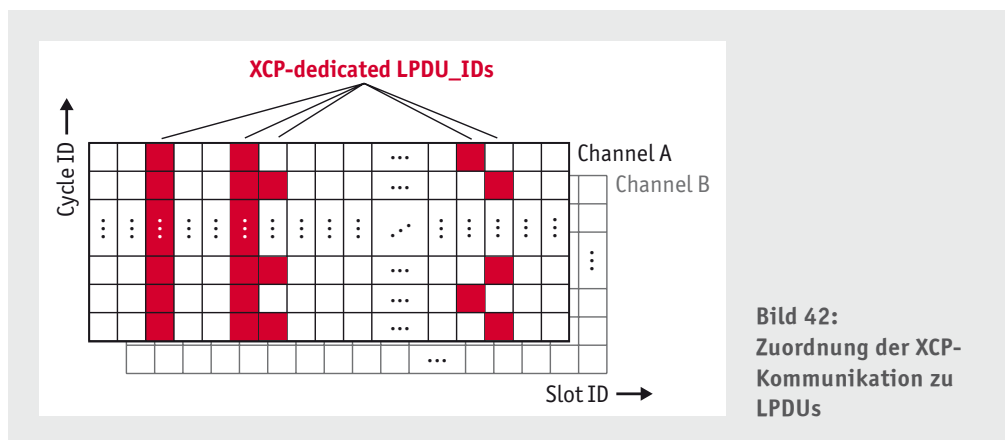


Bild 42:
Zuordnung der XCP-
Kommunikation zu
LPDUs

Weitere Details zu XCP on FlexRay können Sie z. B. in der Online-Hilfe von CANape nachlesen. Mit CANape wird ein sogenannter FIBEX Viewer mitgeliefert, der das komfortable Betrachten des Scheduling erlaubt. Die Zuordnung der XCP-Botschaften zu den LPDUs erfolgt bequem über die Treibereinstellungen des XCP-on-FlexRay-Gerätes in CANape.

Das Protokoll wird ausführlich in ASAM XCP on FlexRay Teil 3 Transport Layer Specification 1.1.0 erläutert. Eine XCP-on-FlexRay-Demo mit dem virtuellen Steuergerät XCPsim finden Sie in CANape. Die Demo setzt eine reale Vector FlexRay-Hardware voraus.

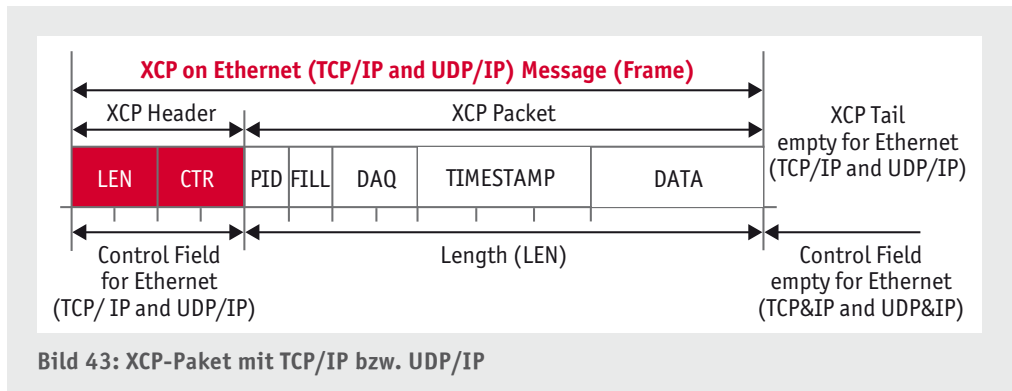
1.4.3 Ethernet

XCP on Ethernet kann wahlweise mit TCP/IP oder mit UDP/IP verwendet werden. TCP ist ein gesichertes Transportprotokoll auf Ethernet, bei dem mittels des Handshake-Verfahrens der Verlust eines Paketes festgestellt wird. TCP organisiert im Falle eines Paketverlusts die Wiederholung des Paketes. UDP bietet diesen Sicherheitsmechanismus nicht. Geht hier ein Paket verloren, bietet UDP auf dieser Protokollebene keine Mechanismen zur erneuten Versendung des verloren gegangenen Paketes.

XCP on Ethernet kann nicht nur mit realen Steuergeräten verwendet werden, sondern auch zum Messen und Verstellen virtueller Steuergeräte. Unter einem virtuellen Steuergerät versteht man die Nutzung des Codes, der sonst im Steuergerät abläuft, als ausführbares Programm (z. B. DLL)

auf dem PC. Dabei stehen ganz andere Ressourcen als in einem Steuergerät zur Verfügung (CPU, Speicher ...).

Doch zunächst zum eigentlichen Protokoll. IP-Pakete beinhalten immer die Adresse des Absenders und des Empfängers. Am einfachsten kann man sich ein IP-Paket als eine Art Brief vorstellen, auf dem die Anschriften des Empfängers und des Absenders stehen. Die Adressen der einzelnen Teilnehmer müssen immer eindeutig sein. Zu einer eindeutigen Adresse gehören IP-Adresse und Port-Nummer.



Der Header besteht aus einem Kontrollfeld mit zwei Words im Intel-Format (= vier Bytes). Sie enthalten die Länge (LEN) und einen Counter (CTR). LEN gibt die Anzahl der Bytes des XCP-Paketes an. Der CTR dient zur Identifizierung des Paketverlusts. UDP/IP ist kein gesichertes Protokoll. Kommt es zu einem Paketverlust, wird dieser nicht von der Protokollschicht abgefangen. Die Kontrolle erfolgt über die Counter-Information. Sendet der Master die erste Botschaft an den Slave, so generiert er eine Counter-Nummer, die bei jedem weiteren Verschicken einer Nachricht inkrementiert wird. Der Slave antwortet mit dem gleichen Muster: Er inkrementiert seinen eigenen Counter mit jeder Botschaft, die er sendet. Der Counter des Slaves und der Counter des Masters arbeiten unabhängig voneinander. Für den Versand von Messdaten ist UDP/IP gut geeignet. Kommt es zum Paketverlust, fehlen die darin befindlichen Messdaten. Es kommt zu einer Messlücke. Geschieht das nur gelegentlich, kann der Verlust vernachlässigt werden. Wird aber auf Basis der Messwerte eine schnelle Regelung realisiert, ist eventuell die Verwendung von TCP/IP zu empfehlen.

Ein Ethernet-Paket kann mehrere XCP-Pakete transportieren, aber ein XCP-Paket darf nie die Grenzen eines UDP/IP-Paketes überschreiten. Im Falle von XCP on Ethernet gibt es keinen „Tail“, also ein leeres Kontrollfeld.

Weitere Informationen zum Protokoll erhalten Sie in ASAM XCP on Ethernet Teil 3 Transport Layer Specification 1.1.0. In CANape finden Sie auch eine XCP on Ethernet Demo mit dem virtuellen Steuergerät XCPsim bzw. mit virtuellen Steuergeräten in Form von DLLs, die mit Hilfe von Simulink-Modellen und dem Simulink Coder umgesetzt wurden.

1.4.4 SxI

SxI steht als Sammelbezeichnung für SPI bzw. SCI. Da es sich hierbei nicht um einen Datenbus, sondern um Controller-Schnittstellen handelt, die nur für Punkt-zu-Punkt-Verbindungen geeignet sind, gibt es bei dieser Übertragung keine Adressierung. Die Kommunikation zwischen beiden Teilnehmern läuft dabei entweder synchron oder asynchron ab.

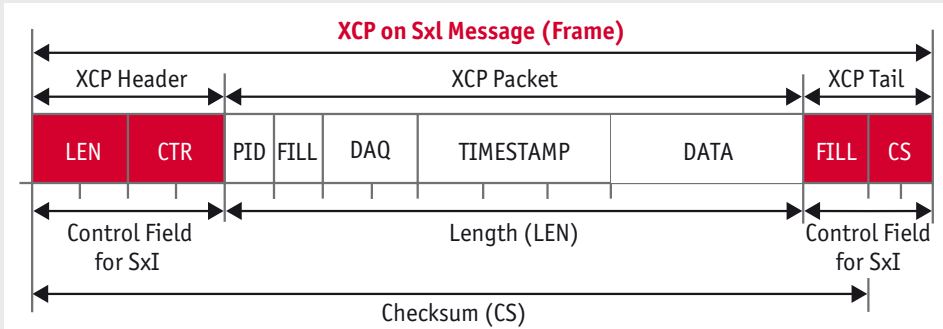


Bild 44: XCP-on-SxI-Paket

Der XCP-Header besteht aus einem Kontrollfeld mit zwei Informationen: der Länge LEN und dem Counter. Die Länge dieser Größen kann Byte oder Word sein (Intel-Format). LEN gibt die Anzahl der Bytes des XCP-Paketes an. Der CTR dient dazu, Paketverlust zu identifizieren. Die Kontrolle erfolgt genauso wie bei XCP on Ethernet über die Counter-Information. Unter gewissen Umständen kann es notwendig sein, das Paket mit Füll-Bytes zu versehen, beispielsweise wenn SPI im WORD- bzw. im DWORD-Modus verwendet wird oder wenn die minimale Paketlänge mit der Botschaft sonst unterschritten wäre. Diese Füll-Bytes werden im Control Field angehängt.

Weitere Informationen zum Protokoll finden Sie in ASAM XCP on SxI Teil 3 Transport Layer Specification 1.1.0.

1.4.5 USB

Derzeit hat XCP on USB keine praktische Bedeutung. Deshalb wird dem Thema hier kein weiterer Platz eingeräumt, sondern nur auf die Dokumente von ASAM verwiesen ASAM XCP on USB Teil 3 Transport Layer Specification 1.1.0, die den Standard beschreiben.

1.4.6 LIN

Aktuell hat ASAM keinen XCP-on-LIN-Standard definiert. Es existiert aber eine Lösung von Vector (XCP-on-LIN-Treiber und CANape als XCP-on-LIN-Master), die weder die LIN- noch die XCP-Spezifikation verletzt und bereits in einigen Kundenprojekten verwendet wird. Für nähere Informationen wenden Sie sich bitte an Vector.

1.5 XCP-Services

In diesem Kapitel erfolgt eine Aufstellung und Erläuterung weiterer Services, die über XCP realisiert werden können. Sie basieren alle auf den bereits beschriebenen Mechanismen der Kommunikation mit Hilfe von CTOs und DTOs. Einige XCP-Services wurden bereits erläutert, z. B. die synchrone Datenerfassung/Stimulation oder der Schreib/Lese-Zugriff in den Gerätespeicher. Die XCP-Spezifikation legt die unterschiedlichen Services zwar eindeutig fest, sagt aber gleichzeitig aus, ob der Service auf jeden Fall implementiert sein muss oder ob er optional ist. Beispielsweise muss ein XCP-Slave ein „Connect“ für den Verbindungsaufbau durch den Master unterstützen. Flashen über XCP ist jedoch nicht zwingend, der XCP-Slave muss es nicht unterstützen. Das hängt einfach von den Anforderungen an das Projekt und die Software ab. Alle in diesem Kapitel aufgeführten Services sind optional.

1.5.1 Speicherseitenumschaltung

Wie bereits bei der Beschreibung der Kalibrierkonzepte erläutert, stehen Parameter normalerweise im Flash-Speicher und werden bei Bedarf ins RAM umkopiert. Manche Kalibrierkonzepte bieten die Möglichkeit der Umschaltung von Speichersegmentseiten von RAM und Flash. XCP beschreibt einen etwas allgemeineren, generischen Ansatz, bei dem ein Speichersegment mehrere umschaltbare Seiten beinhalten kann. Im Normalfall handelt es sich dabei um eine RAM- und eine Flash-Seite. Es sind aber auch mehrere RAM-Seiten oder das Fehlen der Flash-Seite denkbar.

Um die XCP-Befehle zur Seitenumschaltung besser zu verstehen, sollen an dieser Stelle die Begriffe Sektor, Segment und Seite nochmals erläutert werden.

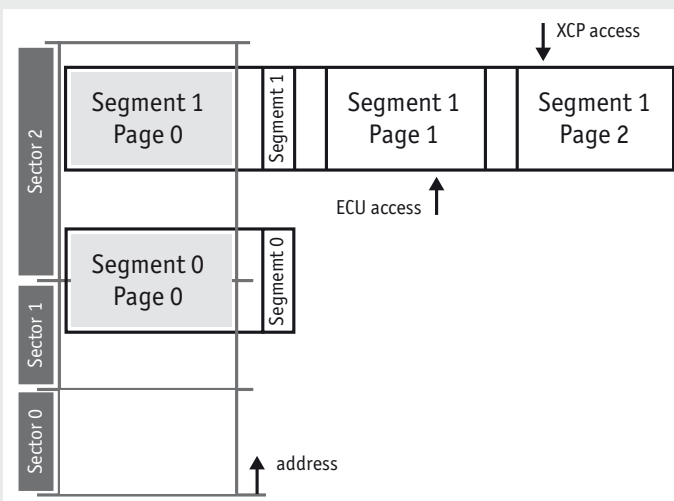


Bild 45:
Speicherdarstellung

Aus XCP-Sicht besteht der Speicher eines Slaves aus einem kontinuierlichen Speicher, der mit einer 40-Bit-Breite adressiert wird. Der physikalische Aufbau eines Speichers erfolgt in Sektoren. Die Kenntnisse über die Flash-Sektoren sind beim Flashen zwingend erforderlich, weil sich die Flash-Speicher nur blockweise löschen lassen.

Der logische Aufbau erfolgt in sogenannten Segmenten, die beschreiben, wo sich Kalibrierdaten im Speicher befinden. Die Startadresse und die Größe eines Segments müssen sich nicht an den Startadressen und Größen der physikalischen Sektoren orientieren. Jedes Segment kann in mehrere Seiten unterteilt sein. Die Seiten eines Segments beschreiben die gleichen Parameter auf den gleichen Adressen. Die Werte dieser Parameter und die Schreib/Lese-Rechte können für jede Seite individuell geregelt sein.

Die Zuordnung des Algorithmus zu einer Seite innerhalb eines Segments muss zu jedem Zeitpunkt eindeutig sein. In einem Segment kann immer nur eine Seite aktiv sein. Diese Seite ist die sogenannte „aktive Seite für das Steuergerät in diesem Segment“. Auf welche Seite das Steuergerät und der XCP-Treiber aktiv zugreifen, kann individuell geschaltet werden. Zwischen diesen Einstellungen besteht keine Abhängigkeit. Angelehnt an die Bezeichnung für das Steuergerät, heißt die aktive Seite für den XCP-Zugriff „aktive Seite für den XCP-Zugriff in diesem Segment“.

Das gilt wiederum individuell für jedes Segment. Segmente müssen in der A2L-Datei angegeben sein und jedes Segment erhält eine Nummer, über die auf das Segment referenziert wird. Innerhalb eines XCP-Slaves muss die SEGMENT_NUMBER immer bei 0 beginnen und dann fortlaufend inkrementiert werden.

Jedes Segment hat mindestens eine Seite. Auch die Seiten werden über Nummern referenziert. Die erste Seite ist die PAGE 0. Für die Nummer steht ein Byte zur Verfügung, sodass pro Segment maximal 255 Seiten definiert werden können.

Der Slave muss alle Seiten für alle Segmente initialisieren. Über das Kommando GET_CAL_PAGE erfragt der Master beim Slave, welche Seite für den Steuergeräte- und welche für den XCP-Zugriff gerade aktiv ist. Dabei kann es durchaus geschehen, dass eine gegenseitige Verriegelung bei den Zugriffen notwendig ist. Beispielsweise darf der XCP-Slave nicht auf eine Seite zugreifen, wenn diese Seite gerade für das Steuergerät aktiv ist. Wie gesagt: Es kann eine Abhängigkeit geben – muss aber nicht. Es ist eine Frage dessen, wie die Slave-Implementierung realisiert ist.

Unterstützt der Slave die optionalen Kommandos GET_CAL_PAGE und SET_CAL_PAGE, dann unterstützt er auch die sogenannte Seitenumschaltung. Über diese beiden Kommandos kann der Master also erfragen, welche Seiten aktuell genutzt werden, und kann sie bei Bedarf für das Steuergerät und den XCP-Zugriff umschalten. Der XCP-Master hat die volle Kontrolle über die Umschaltung der Seiten. Der XCP-Slave kann selbstständig keine Umschaltung auslösen. Aber natürlich muss der Master die Restriktionen respektieren, die sich aus der Implementierung im Slave ergeben.

Worin liegt der Nutzen einer Umschaltung?

Die Umschaltung erlaubt einerseits einen schlagartigen Austausch kompletter Bedatungen – quasi einen Vorher-nachher-Vergleich. Andererseits befindet sich die Regelstrecke in einem stabilen Zustand, während der Applikateur umfangreiche Parameteränderungen auf einer anderen Seite im Steuergerät durchführt. Somit wird verhindert, dass z. B. durch unvollständige Datenstände bei der Bedatung ein kritischer oder instabiler Zustand der Regelstrecke herrscht.

1.5.2 Sicherung von Speicherseiten – Data Page Freezing

Verstellt ein Applikateur Parameter auf einer Seite, gibt es in XCP die konzeptionelle Möglichkeit, die Daten direkt im Steuergerät zu speichern. Dabei werden die Daten einer RAM-Seite auf eine Seite mit nicht flüchtigem Speicher gesichert. Ist der nicht flüchtige Speicher ein Flash, muss berücksichtigt werden, dass die Segment-Startadresse und die Segmentgröße nicht unbedingt mit den Flash-Sektoren übereinstimmen müssen, was beim Löschen und Neubeschreiben des Flash-Speichers ein Problem darstellt (siehe ASAM XCP Teil 2 Protocol Layer Specification 1.1.0).

1.5.3 Flash-Programmierung

Flashen bedeutet, Daten in einen Bereich des Flash-Speichers zu schreiben. Dazu muss genau bekannt sein, wie der Speicher aufgebaut ist. Ein Flash-Speicher unterteilt sich in mehrere Sektoren (physikalische Abschnitte), die mit einer Startadresse und einer Länge beschrieben werden. Um diese zu unterscheiden, erhalten sie eine fortlaufende Identifikationsnummer. Dazu steht ein Byte zur Verfügung, sodass die maximale Anzahl der Sektoren 255 beträgt.

SECTOR_NUMBER [0, 1, 2 ... 255]

Auch die Informationen über die Flash-Sektoren sind Bestandteil der A2L-Bedatung.

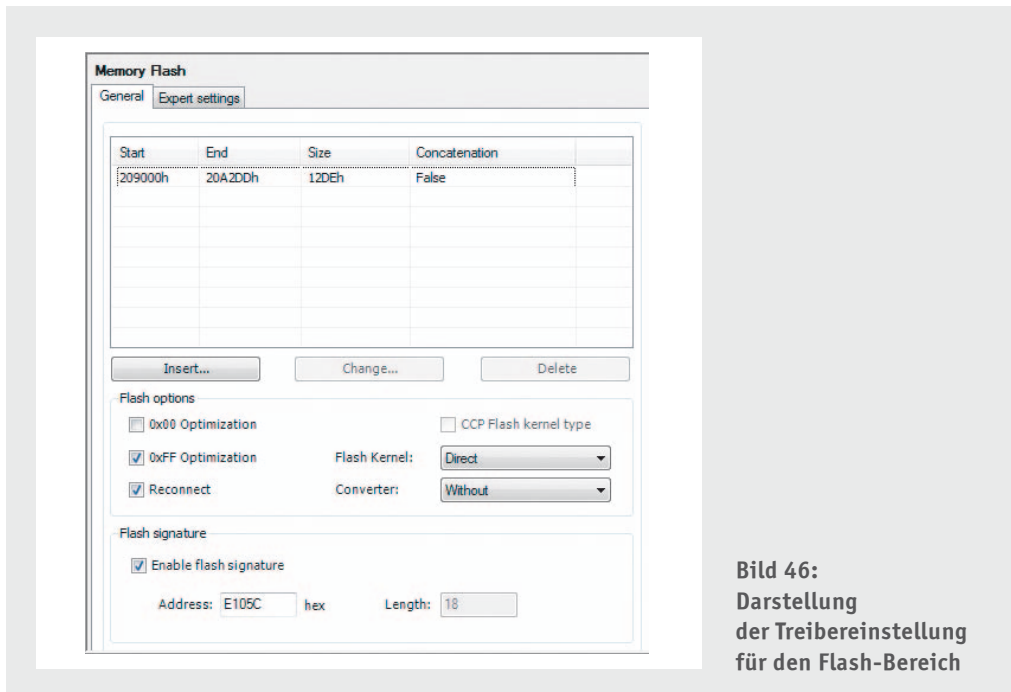


Bild 46:
Darstellung
der Treibereinstellung
für den Flash-Bereich

Das Flashen kann über sogenannte „Flash Kernels“ realisiert werden. Das ist ein ausführbarer Code, der vor dem eigentlichen Flashen in den RAM-Bereich des Slaves gesendet wird und dann die Kommunikation mit dem XCP-Master übernimmt. Darin ist möglicherweise der Algorithmus enthalten, der für das Löschen des Flash-Speichers verantwortlich ist. Aus Sicherheits- und Platzgründen ist dieser sehr häufig nicht dauerhaft im Flash-Speicher des Steuergerätes vorhanden. Unter Umständen kommt ein Konverter zum Einsatz, wenn beispielsweise Checksummen- oder ähnliche Berechnungen durchzuführen sind.

Das Flashen mit XCP unterteilt den gesamten Flash-Vorgang grob in drei Bereiche:

- > Vorbereitung (z. B. zur Versionskontrolle und somit zur Überprüfung, ob der neue Inhalt überhaupt geflasht werden kann)
- > Durchführung (der neue Inhalt wird an das Steuergerät gesendet)
- > Nachbereitung (z. B. Checksummen-Kontrolle etc.)

Im XCP-Standard ist das Hauptaugenmerk auf die eigentliche Durchführung des Flashens gerichtet. Wer diesen Vorgang mit dem Flashen über Diagnoseprotokolle vergleicht, wird feststellen, dass die prozessspezifischen Elemente, wie z. B. Seriennummer-Handling mit Meta-Daten, im XCP eher sparsam unterstützt werden. Das Flashen in der Entwicklungsphase stand bei der Definition eindeutig im Vordergrund und nicht komplexe Prozessschritte, die am Bandende notwendig sind.

In der Vorbereitungsphase geht es deswegen im Wesentlichen darum, festzustellen, ob der neue Inhalt überhaupt zum Steuergerät passt. Spezielle Kommandos zur Versionskontrolle existieren nicht. Vielmehr hat sich die Praxis etabliert, dass projektspezifisch festgelegt wird, welche Kommandos unterstützt werden.

Folgende XCP-Kommandos stehen zur Verfügung:

PROGRAM_START: Beginn des Flash-Ablaufs

Dieses Kommando zeigt den Beginn des Flash-Vorgangs an. Sollte das Steuergerät in einem Zustand sein, der das Flashen nicht erlaubt (z. B. Geschwindigkeit des Fahrzeugs > 0), muss der XCP-Slave mit einem Error quittieren. Erst wenn der PROGRAM_START erfolgreich vom Slave bestätigt wird, darf der eigentliche Flash-Prozess anfangen.

PROGRAM_CLEAR: Aufruf der eigentlichen Flash-Speicher-Löschroutine

Bevor ein Flash-Speicher mit einem neuen Inhalt überschrieben werden kann, muss er zunächst gelöscht werden. Der Aufruf der Löschroutine über dieses Kommando muss im Steuergerät implementiert sein bzw. mit Hilfe des Flash Kernels dem Steuergerät zur Verfügung gestellt werden.

PROGRAM_FORMAT: Wahl des Datenformates der Flash-Daten

Mit diesem Befehl legt der XCP-Master fest, in welchem Format (z. B. komprimiert oder verschlüsselt) die Daten zum Slave übertragen werden. Wird der Befehl nicht versendet, ist die Default-Einstellung die nicht komprimierte und nicht verschlüsselte Übertragung.

PROGRAM: Übertragung der Daten zum XCP-Slave

Für die Anwender, die sich mit dem Flashen über Diagnose auskennen: Dieser Befehl entspricht TRANSFERDATA in der Diagnose. Unter Zuhilfenahme dieses Kommandos werden Daten an den XCP-Slave übertragen, die dann im Flash-Speicher abgelegt werden.

PROGRAM_VERIFY: Anforderung zur Überprüfung des neuen Flash-Inhaltes

Der Master kann den Slave auffordern, eine interne Prüfung – ob der neue Inhalt in Ordnung ist – durchzuführen.

PROGRAM_RESET: Reset-Aufforderung an den Slave

Aufforderung des Masters an den Slave, einen Reset durchzuführen. Danach ist auf jeden Fall die Verbindung zum Slave getrennt und ein neues CONNECT muss gesendet werden.

1.5.4 Automatische Erkennung des Slaves

Das XCP-Protokoll erlaubt dem Master, den Slave über seine protokollspezifischen Eigenschaften zu befragen. Dazu steht eine Reihe von Kommandos zur Verfügung:

GET_COMM_MODE_INFO

Die Antwort auf dieses Kommando bringt dem Master Informationen über verschiedene Kommunikationsmöglichkeiten des Slaves, z. B., ob er den Blocktransfer- oder den Interleaved-Modus unterstützt oder welche zeitlichen Mindestabstände zwischen Requests in diesen Modi durch den Master eingehalten werden müssen.

GET_STATUS

Die Antwort auf diese Anfrage liefert alle aktuellen Statusinformationen des Slaves. Welche Ressourcen (Kalibrieren, Flashen, Messen ...) sind geschützt? Laufen momentan noch irgendwelche Speicheraktivitäten (DAQ-Listen-Konfiguration ...)? Werden gerade DTOs (DAQ, STIM) ausgetauscht?

GET_DAQ_PROCESSOR_INFO

Der Master erhält die allgemeinen Informationen, die er über die Restriktionen des Slaves kennen muss: Anzahl der vordefinierten DAQ-Listen, der verfügbaren DAQ-Listen und Events ...

GET_DAQ_RESOLUTION_INFO

Weitere Informationen über die DAQ-Möglichkeiten des Slaves werden über dieses Kommando ausgetauscht: maximale Größe einer ODT für DAQ und für STIM, Granularität der ODT-Einträge, Anzahl der Bytes bei Zeitstempelübertragung ...

GET_DAQ_EVENT_INFO

Wenn dieses Kommando verwendet wird, erfolgt der Aufruf einmalig pro Event des Steuergerätes. Dabei werden Informationen darüber übermittelt, ob das Event für DAQ, STIM oder DAQ/STIM genutzt werden kann, ob das Event zyklisch auftritt und, wenn ja, welche Zykluszeit es hat ...

1.5.5 Blocktransfer-Modus für Upload, Download und Flashen

Im „normalen“ Kommunikationsmodus wird jedes Kommando des Masters von einer Antwort des Slaves quittiert. In manchen Fällen ist es aber aus Performance-Gründen wünschenswert, den sogenannten Blocktransfer-Modus zu verwenden.

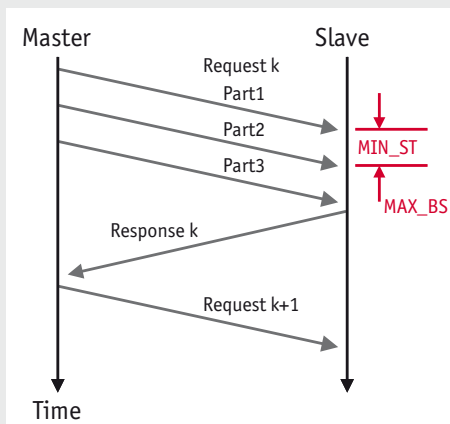


Bild 47:
Darstellung des
Blocktransfer-Modus

Bei der Übertragung größerer Datenmengen (UPLOAD, SHORT_UPLOAD, DOWNLOAD, SHORT_DOWNLOAD und PROGRAM) beschleunigt die Nutzung eines solchen Verfahrens die Abläufe. Ob der Slave das Verfahren unterstützt, erfährt der Master über die Abfrage GET_COMM_MODE_INFO. Mehr dazu finden Sie in ASAM XCP Teil 2 Protocol Layer Specification 1.1.0.

1.5.6 Kaltstartmessung (Start der Messung bei Power-On)

Eine eventgesteuerte Messung, die praktisch schon in der Startphase des Steuergerätes durchgeführt wird, ist mit den bisher beschriebenen Mitteln des XCP nicht möglich. Ursache ist, dass vor der eigentlichen Messung die Konfiguration der Messung erfolgen muss. Wählt man diesen Weg, ist die Startphase des Steuergerätes längst vorbei, bis die ersten Messdaten übertragen werden. Der Ansatz dafür, die Aufgabenstellung dennoch zu meistern, basiert auf einer simplen Idee:

Die Konfiguration und die Messung werden zeitlich voneinander getrennt. Nach der Konfigurationsphase wird nicht direkt die Messung gestartet, sondern das Steuergerät ausgeschaltet. Nach dem erneuten Booten greift der XCP-Slave direkt auf die schon vorhandene Konfiguration zu und beginnt sofort mit dem Versenden der ersten Botschaften. Die damit verbundenen Schwierigkeiten liegen auf der Hand: Die Konfiguration der DAQ-Listen wird im RAM gespeichert und somit ist die Information nach einem Boot nicht mehr vorhanden.

Um den sogenannten RESUME Mode zu nutzen und eine Kaltstartmessung zu ermöglichen, bedarf es im XCP-Slave eines nicht flüchtigen Speichers, der auch ohne Stromversorgung seine Daten nicht verliert. Bei dieser Methode kommt EEPROM zum Einsatz. Ob es sich dabei um ein reales EEPROM oder um ein durch einen Flash-Speicher emuliertes handelt, ist für diese Betrachtung irrelevant.

Mehr Details finden Sie unter ASAM XCP Teil 1 Overview Specification 1.1.0 im Kapitel 1.4.2.2 über die „Advanced Features“.

1.5.7 Schutzmechanismen mit XCP

Ein nicht autorisierter Anwender soll möglichst keine Verbindung zu einem Steuergerät aufbauen können. Zur Überprüfung, ob ein Verbindungsaufbau autorisiert ist, steht das „Seed & Key“-Verfahren zur Verfügung. Per Seed & Key können die drei unterschiedlichen Zugriffsarten geschützt werden: Messen/Stimulation, Verstellen und Flashen.

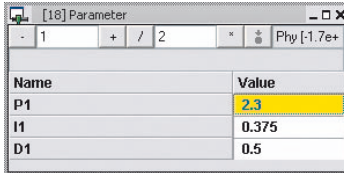
Das „Seed & Key“-Verfahren funktioniert folgendermaßen: Bei der Verbindungsanfrage durch den Master versendet der Slave eine Zufallszahl (= Seed) an den Master. Nun muss der Master über einen Algorithmus eine Antwort (= Key) generieren. Der Key wird an den Slave gesendet. Der Slave kalkuliert ebenfalls die zu erwartende Antwort und vergleicht den Key des Masters mit seinem eigenen Ergebnis. Stimmen beide Ergebnisse überein, haben sowohl Master als auch Slave den gleichen Algorithmus verwendet. Daraufhin akzeptiert der Slave die Verbindung zum Master. Gibt es keine Übereinstimmung, verweigert der Slave die Kommunikation mit dem Master.

Normalerweise steht der Algorithmus im Master als DLL zur Verfügung. Hat ein Anwender also die „Seed & Key“-DLL und die A2L-Datei, so steht dem Zugriff auf den Speicher im Steuergerät nichts mehr im Wege. Nähert sich das Steuergerät dem Serienstart, wird der XCP-Treiber oft deaktiviert. Zur Wiederherstellung des XCP-Zugangs zum Steuergerät wird meist eine individuelle Sequenz von Diagnosekommandos verwendet. Somit steht der XCP-Treiber größtenteils noch in Serienfahrzeugen zur Verfügung, ist aber zum Schutz vor Manipulationen des Steuergerätes im Normalfall deaktiviert (siehe ASAM XCP Teil 2 Protocol Layer Specification 1.1.0).

Ob in einem Projekt Seed & Key oder die Deaktivierung des XCP-Treibers zum Einsatz kommt oder nicht, ist implementierungsspezifisch und unabhängig von der XCP-Spezifikation.

2 Steuergeräte-Beschreibungsdatei A2L

Ein Grund dafür, warum eine A2L-Datei benötigt wird, wurde schon genannt: die Zuordnung von symbolischen Namen zu Adressen. Hat der Software-Entwickler in seiner Anwendung beispielsweise einen PID-Regler realisiert und den Größen Proportional-, Integral- und Differenzialanteil die Namen P1, I1 und D1 gegeben, so sollte der Applikateur auf diese Größen mit ihren symbolischen Namen zugreifen können. Nehmen wir das folgende Bild als Beispiel:



Name	Value
P1	2.3
I1	0.375
D1	0.5

Bild 48:
Parameter in einem
Verstellfenster

Durch die Nutzung der symbolischen Namen kann der Anwender komfortabel die Werte verändern. Ein weiteres Beispiel liefert die Betrachtung von Signalgrößen, die aus dem Steuergerät heraus gemessen werden:

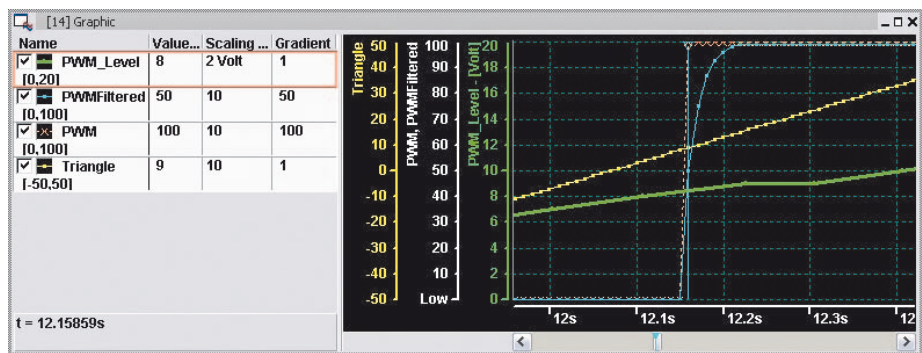


Bild 49: Signaldarstellung über der Zeit

In der Legende kann der Anwender die logischen Namen der Signale ablesen. Auf welcher Adresse letztendlich die Größen einmal im Steuergerät gestanden haben, ist bei der Offline-Betrachtung der Werte sekundär. Für die Anfrage nach den Werten im Steuergerät ist die korrekte Adresse natürlich notwendig, aber der Zahlenwert der Adresse selbst ist für den Anwender ohne Belang. Er nutzt den logischen Namen für die Auswahl und Visualisierung. Der Anwender wählt also das Objekt anhand des Namens aus und der XCP-Master schaut in der A2L nach der zugehörigen Adresse und dem Datentyp.

Als weiteres Attribut einer Größe können etwa ein minimaler und ein maximaler Wert definiert sein. Der Wert des Objektes muss sich innerhalb dieser Grenzen bewegen. Stellen Sie sich vor, dass Sie als Software-Entwickler eine Größe definieren, die direkten Einfluss auf eine Leistungsendstufe hat. Sie müssen nun verhindern, dass der Anwender – aus welchen Gründen auch immer – den Wert so einstellt, dass die Endstufe durchbrennt. Das erledigen Sie, indem Sie die erlaubten Werte mit Hilfe der Definition der minimalen und maximalen Werte in der A2L festlegen.

Regeln für die Umrechnung zwischen physikalischen und Rohwerten werden ebenfalls in der A2L definiert. Als einfaches Beispiel für eine solche Umrechnungsregel können Sie sich einen Sensor mit einem 8-Bit-Wert vorstellen. Die Zahlenwerte, die der Sensor liefert, liegen also zwischen 0 und 255. Sie möchten den Wert aber als prozentuale Angabe sehen. Die Abbildung des Sensorwertes [0 ... 255] in [0 ... 100 %] erfolgt über eine Umrechnungsregel, die wiederum in der A2L hinterlegt ist. Wird ein Objekt gemessen, das als Rohwert im Steuergerät vorliegt und auch so übertragen wird, nutzt das Mess- und Kalibrierwerkzeug die hinterlegte Formel und visualisiert den physikalischen Wert.

Neben skalaren Größen finden auch Kennlinien und Kennfelder häufig Verwendung. Sie nutzen beispielsweise einen Abstandsensor, wie einen Hall-Sensor, der über die Stärke eines Magnetfeldes den Abstand ermittelt, und wollen den Abstandswert in Ihrem Algorithmus verwenden. Das Magnetfeld und der Abstandswert laufen nicht linear zueinander. Diese Nichtlinearität der Werte würde die Formulierung des Algorithmus unnötig erschweren. Mit Hilfe einer Kennlinie können Sie die Werte zunächst linearisieren, bevor Sie die Werte als Eingangsgrößen in Ihren Algorithmus eingeben.

Ein anderes Einsatzgebiet von Kennfeldern ist ihre Nutzung anstelle komplexer Berechnungen. Gibt es beispielsweise einen Zusammenhang $y = f(x)$ und die Funktion f ist nur mit sehr viel Rechenaufwand zu ermitteln, ist es oft einfacher, die Werte über den möglichen Bereich von x zu berechnen und die Ergebnisse in Form einer Tabelle (= Kennlinie) abzulegen. Steht nun im Steuergerät der Wert x fest, muss nicht zur Laufzeit des Controllers der Wert y berechnet werden, sondern das Kennfeld liefert auf die Eingangsgröße x das Ergebnis y . Eventuell muss noch zwischen zwei Werten interpoliert werden, aber das wäre dann schon alles.

Wie erfolgt nun die Ablage dieser Kennlinie im Speicher? Kommen erst alle x -Werte und anschließend alle y -Werte? Oder erfolgt die Ablage nach dem Schema: $x_1, y_1; x_2, y_2; x_3, y_3 \dots$? Da sich verschiedene Optionen eröffnen, wird die Art und Weise der Speicherablage über ein Ablageschema in der A2L definiert.

Der Komfort für den Anwender ergibt sich aus dem symbolischen Umgang mit den Größen, dem direkten Blick auf die physikalischen Werte und dem Zugriff auf komplexe Elemente wie z. B. Kennfelder, ohne sich Gedanken über komplexe Ablageschemata machen zu müssen.

Einen weiteren Vorteil bieten die Kommunikationsparameter. Auch diese sind in der A2L definiert. Bei der Kommunikation zwischen Mess- und Kalibrierwerkzeug und Steuergerät wird die Bedatung aus der A2L genutzt. Somit befindet sich in der A2L alles, was das Mess- und Kalibrierwerkzeug benötigt, um mit dem Steuergerät kommunizieren zu können.

2.1 Aufbau einer A2L-Datei für einen XCP-Slave

Die A2L-Datei ist eine ASCII-lesbare Datei, die mit Hilfe von Schlüsselwörtern

- > die interfacespezifischen Parameter zwischen Mess- und Kalibrierwerkzeug und A2L-Datei (die Beschreibung steht am Anfang der A2L-Datei und befindet sich in einem sogenannten AML-Baum),
 - > die Kommunikation zum Steuergerät,
 - > Ablageschemata für Kennlinien und -felder (Schlüsselwort RECORD_LAYOUT),
 - > Umrechnungsregeln für die Umrechnung von Rohwerten auf physikalische Werte (Schlüsselwort COMPU_METHOD),
 - > Messgrößen (Schlüsselwort MEASUREMENT),
 - > Verstellgrößen (Schlüsselwort CHARACTERISTIC) und
 - > Ereignisse, die zum Auslösen einer Messung geeignet sind (Schlüsselwort EVENT),
- beschreibt. Eine Zusammenfassung von Parametern und Messgrößen erfolgt unter Zuhilfenahme von Gruppen (Schlüsselwort GROUP).

Beispiel einer Messgröße mit dem Namen „Shifter_B3“:

```
/begin MEASUREMENT Shifter_B3 „Single bit signal (bit from a byte shifting)“
  UBYTE HighLow 0 0 0 1
  READ_WRITE
  BIT_MASK 0x8
  BYTE_ORDER MSB_LAST
  ECU_ADDRESS 0x124C02
  ECU_ADDRESS_EXTENSION 0x0
  FORMAT „%.3“
/end IF_DATA CANAPE_EXT
  100
  LINK_MAP „byteShift“ 0x124C02 0x0 0 0x0 1 0x87 0x0
  DISPLAY 0 0 20
/end IF_DATA
/end MEASUREMENT
```

Beispiel eines Parameter-Kennfeldes mit dem Namen KF1:

```
/begin CHARACTERISTIC KF1 „8*8 BYTE no axis“
  MAP 0xE0338 __UBYTE_Z 0 Factor100 0 2.55
  ECU_ADDRESS_EXTENSION 0x0
  EXTENDED_LIMITS 0 2.55
  BYTE_ORDER MSB_LAST
  BIT_MASK 0xFF
/end IF_DATA AXIS_DESCR
  FIX_AXIS NO_INPUT_QUANTITY BitSlice.CONVERSION 8 0 7
  EXTENDED_LIMITS 0 7
  READ_ONLY
```

```

    BYTE_ORDER MSB_LAST
    FORMAT „%.0“
    FIX_AXIS_PAR_DIST 0 1 8
/end AXIS_DESCR
/begin AXIS_DESCR
    FIX_AXIS NO_INPUT_QUANTITY BitSlice.CONVERSION 8 0 7
    EXTENDED_LIMITS 0 7
    READ_ONLY
    BYTE_ORDER MSB_LAST
    FORMAT „%.0“
    FIX_AXIS_PAR_DIST 0 1 8
/end AXIS_DESCR
/begin IF_DATA CANAPE_EXT
    100
    LINK_MAP „map3_8_8_uc“ 0xE0338 0x0 0 0x0 1 0x87 0x0
    DISPLAY 0 0 255
/end IF_DATA
    FORMAT „%.3“
/end CHARACTERISTIC

```

Der ASCII-Text ist nicht leicht zu verstehen. Eine Beschreibung des Aufbaus finden Sie in ASAM XCP Teil 2 Protocol Layer Specification 1.1.0 in Kapitel 2.

Im Folgenden wird demonstriert, wie eine A2L erstellt wird. Konzentrieren wir uns auf die eigentlichen Inhalte einer A2L und deren Bedeutung und überlassen die Details der A2L-Beschreibungssprache einem Editor. Im Weiteren kommt dabei der mit CANape ausgelieferte A2L-Editor zum Einsatz.

2.2 Manuelle Erstellung einer A2L-Datei

Die A2L beschreibt im Wesentlichen den Inhalt des Speichers des XCP-Slaves. Der wiederum hängt von der darin befindlichen Anwendung ab, die als C-Code entwickelt wurde. Nach dem Compiler/Linker-Lauf des Anwendungs-Codes liegen bereits wichtige Elemente einer A2L-Datei in der Linker-Map-Datei vor: der Name der Objekte, die Datentypen und die Adressen im Speicher. Auf jeden Fall fehlen die Kommunikationsparameter zwischen XCP-Master und -Slave. In aller Regel werden zusätzlich die minimalen und maximalen Werte der Parameter, Umrechnungsregeln, Ablageschemata für Kennfelder usw. benötigt.

Beginnen wir mit der Erstellung einer leeren A2L und den Kommunikationsparametern: Möchten Sie z. B. eine A2L anfertigen, die ein Steuergerät mit XCP-on-CAN-Schnittstelle beschreibt, legen Sie in CANape ein neues Gerät an und selektieren XCP on CAN als Schnittstelle. Dann können Sie weitere kommunikationsspezifische Informationen (z. B. CAN-Identifizier) ergänzen. Nach dem Abspeichern liegt Ihnen eine A2L vor, die den gesamten Kommunikationsanteil der A2L beinhaltet. Nun fehlt noch die Definition der eigentlichen Mess- und Verstellgrößen.

Im A2L-Editor (Teil von CANape oder als separates Werkzeug erhältlich) erfolgt die Zuordnung der Linker-Map-Datei zur A2L. Über einen Auswahl-Dialog kann der Anwender nun aus der Map-Datei die Größen selektieren, die er in der A2L benötigt: skalare Mess- und Verstellgrößen, Kennlinien und -felder. Damit können nun nach und nach die gewünschten Größen in die A2L eingefügt und gruppiert werden. Weitere objektspezifische Informationen werden ebenfalls über den Editor hinzugefügt.

Was ist zu tun, wenn Sie Ihren Code ändern, neu kompilieren und linken? Die Adressen der Objekte werden sich mit einer an Sicherheit grenzenden Wahrscheinlichkeit ändern. Dafür muss grundsätzlich keine neue A2L generiert werden. Möchten Sie neu hinzugekommene Objekte des Codes auch in der A2L verfügbar haben, müssen Sie diese in der A2L natürlich ergänzen. Auf jeden Fall aber ist eine Adressaktualisierung in der A2L notwendig. Das erfolgt über den Editor, der aufgrund des Namens des A2L-Objektes den entsprechenden Eintrag im Linker-Map-File sucht, die Adresse ausliest und in der A2L aktualisiert.

Ändert sich Ihre Anwendung sehr dynamisch, werden Objekte umbenannt, Datentypen angepasst, Parameter gelöscht und andere hinzugenommen, dann ist der manuelle Arbeitsablauf nicht praktikabel. Um aus einem C-Code eine A2L zu generieren, stehen weitere Werkzeuge für einen automatischen Ablauf zur Verfügung.

Auf der Vector Homepage finden Sie Informationen über das „ASAP2 Tool-Set“, mit dessen Hilfe Sie automatisiert A2Ls aus dem Source Code in einem Batch-Prozess generieren können.

2.3 A2L-Inhalt versus ECU-Implementierung

Liest ein XCP-Master-Tool eine A2L ein, die nicht vollständig zum Steuergerät passt, treten eventuell Missverständnisse in der Kommunikation auf. Beispielsweise kann in der A2L-Datei ein anderer Wert bzgl. der Zeitstempelauflösung stehen, als im Steuergerät realisiert worden ist. Ist das der Fall, muss das Problem erkannt und gelöst werden. Unterstützung erhält der Anwender vom Master, der über das Protokoll den Slave fragen kann, was im Slave wirklich umgesetzt worden ist.

XCP bietet eine Reihe von Funktionen, die für die automatische Erkennung des Slaves entwickelt wurden. Dies setzt natürlich voraus, dass die automatische Erkennung im Slave implementiert ist. Fragt der Master den Slave und stimmen die Antworten des Slaves nicht mit der Bedatung der A2L-Beschreibungsdatei überein, muss der Master sich entscheiden, welche Einstellungen er verwenden wird. Bei CANape haben die Informationen, die aus dem Slave ausgelesen werden, eine höhere Priorität als die Informationen aus der A2L.

Hier eine Übersicht über die möglichen Kommandos, um etwas über die XCP-Implementierung im Slave zu erfahren:

GET_DAQ_PROCESSOR_INFO

Liefert allgemeine Informationen zu den DAQ-Listen zurück: MAX_DAQ, MAX_EVENT_CHANNEL, MIN_DAQ ...

GET_DAQ_RESOLUTION_INFO

Maximale Größe eines ODT-Eintrages für DAQ/STIM, Zeitrasterinformationen

GET_DAQ_EVENT_INFO (Event_channel_number)

Gibt Informationen zu einem bestimmten Zeitraster zurück: Name und Auflösung des Zeitrasters, Anzahl der DAQ-Listen, die diesem Raster zugeordnet werden dürfen ...

GET_DAQ_LIST_INFO (DAQ_List_Number)

Gibt Informationen zu der ausgewählten DAQ-Liste zurück: MAX_ODT, MAX_ODT_ENTRIES, liegen als vordefinierte DAQ-Listen vor ...

3 Kalibrierkonzepte

Steuergeräte-Parameter sind konstante Größen, die während der Entwicklung des Steuergerätes oder einer Steuergeräte-Variante angepasst und optimiert werden. Dies ist ein iterativer Prozess, in dem durch wiederholte Messungen und Veränderungen der optimale Wert eines Parameters gefunden wird.

Das Kalibrierkonzept beantwortet die Frage, wie Parameter im Steuergerät während der Entwicklungs- und Kalibrierphase des Steuergerätes verändert werden können. Es existiert nicht nur ein Kalibrierkonzept, sondern mehrere. Welches Konzept in Frage kommt, hängt meist stark von den Möglichkeiten und Ressourcen des verwendeten Mikrocontrollers ab.

Im Normalfall werden Parameter im Flash-Speicher des Seriensteuergerätes abgelegt. Die zugrunde liegenden Programmvariablen sind in der Software als Konstanten definiert. Um Parameter während der Entwicklung des Steuergerätes zur Laufzeit veränderbar zu machen, benötigt man zusätzlichen RAM-Speicher.

Ein Kalibrierkonzept beschäftigt sich unter anderem mit folgenden Fragestellungen: Wie kommen die Parameter initial vom Flash ins RAM? Wie wird der Zugriff des Mikrocontrollers auf das RAM umgeleitet? Wie sieht die Lösung aus, wenn es mehr Parameter gibt, als im RAM gleichzeitig untergebracht werden können? Wie kommen die Parameter zurück ins Flash? Sind Veränderungen an den Parametern persistent, d. h., bleiben sie beim Ausschalten des Steuergerätes erhalten?

Man unterscheidet zwischen transparenten und nicht transparenten Kalibrierkonzepten. Transparent bedeutet, dass das Kalibrier-Tool sich nicht um die obigen Fragen kümmern muss, da alle notwendigen Mechanismen durch die Steuergeräte-Implementierung realisiert werden.

Im Folgenden werden einige Verfahren kurz vorgestellt.

3.1 Parameter im Flash

Der Software-Entwickler legt im Source Code fest, ob eine Größe eine Variable oder eine Konstante ist, d. h., ob ein Parameter im Flash- oder im RAM-Speicher liegt.

C-Code-Beispiel:

```
const float factor = 0.5;
```

Die Größe „factor“ stellt eine Konstante mit dem Wert 0.5 dar. Beim Kompilieren und Linken des Codes wird für das Objekt „factor“ ein Platz im Flash vorgesehen. Das Objekt wird also einer Adresse zugeordnet, die im Datenbereich des Flash-Speichers liegt. Der Wert 0.5 wird sich an der entsprechenden Adresse in der Hex-Datei wiederfinden und die Adresse erscheint in der Linker-Map-Datei.

Das einfachste denkbare Kalibrierkonzept besteht darin, den Wert im C-Code anzupassen, ein neues Hex-File zu erzeugen und zu flashen. Die Methode ist allerdings sehr umständlich, da jede Wertänderung nur im Code erfolgen kann und einen Compiler/Linker-Lauf mit anschließendem Flashen zur Folge hat. Alternativ kann nur der Wert in der Hex-Datei modifiziert und dieser dann erneut geflasht werden. Jedes Kalibrier-Tool ist dazu in der Lage. Man spricht hier vom „Offline-Kalibrieren“ der Hex-Datei, einem durchaus gebräuchlichen Verfahren.

Unter Umständen kann es bei bestimmten Compilern nötig sein, explizit dafür zu sorgen, dass Parameter in jedem Fall auch im Flash-Speicher abgelegt und nicht etwa in den Code integriert werden und somit gar nicht in der Linker-Map-Datei auftauchen. Auch möchte man es meist nicht dem Zufall überlassen, wo im Flash-Speicher eine Konstante angelegt wird. Die dazu nötigen Mittel sind fast immer compilerspezifische Pragma-Anweisungen. Um den Compiler am Einbetten in den Code zu hindern, genügt meistens das „volatile“ Attribut für konstante Größen. Eine typische Definition einer Flash-Konstanten sieht z. B. so aus:

C-Code-Beispiel:

```
#pragma section „FLASH_Parameter“  
volatile const float factor = 0.5;
```

Ein Kalibrieren von Parametern im Flash ist online normalerweise nicht realisierbar. Zwar sind die meisten Mikrocontroller in der Lage, ihr Flash selbst zu programmieren, was zum Zwecke der Re-Programmierung im Feld benötigt wird. Allerdings hat der Flash-Speicher immer die Eigenschaft, in größeren Blöcken (Sektoren), die sich nur am Stück löschen lassen, organisiert zu sein. Ein gezieltes Flashen von einzelnen Parametern scheidet aus, da das Steuergerät normalerweise nicht die Ressourcen hat, den Rest des Sektors zwischenspeichern und wieder neu zu programmieren. Zudem würde dieser Vorgang zu viel Zeit in Anspruch nehmen.

Manche Steuergeräte verfügen über die Möglichkeit, Daten in einem sogenannten EEPROM-Speicher abzulegen. EEPROM-Speicher können im Gegensatz zu Flash-Speichern jede Speicherzelle einzeln löschen und programmieren. Der verfügbare EEPROM-Speicher ist immer deutlich kleiner als der verfügbare Flash-Speicher, meist ist er auf wenige Kilobytes beschränkt. Ein EEPROM-Speicher wird oft verwendet, um beispielsweise in der Werkstatt programmierbare Parameter abzulegen oder um im Steuergerät einen Persistenzmechanismus, wie für den Kilometerstand, zu realisieren. Hier wäre eine Online-Kalibrierung denkbar, sie wird jedoch selten angewandt, da der Zugriff auf EEPROM-Zellen relativ langsam ist und EEPROM-Parameter meist beim Booten in RAM-Speicher umkopiert werden, wo man direkt auf sie zugreifen kann.

Steuergeräte, die keinen EEPROM-Speicher besitzen, realisieren oft eine sogenannte EEPROM-Emulation. Dabei werden mehrere kleine Flash-Sektoren alternierend verwendet, um Parameteränderungen aufzuzeichnen, sodass immer der letzte gültige Wert ermittelt werden kann. Damit wäre ebenfalls eine Online-Kalibrierung denkbar.

In beiden Fällen würden dann in der Softwarekomponente des XCP-Treibers die betreffenden Speicherzugriffe abgefangen und mit den Softwareroutinen des EEPROM oder der EEPROM-Emulation realisiert. Der Vector XCP Professional Treiber bietet die dafür nötigen Software-Hooks.

3.2 Parameter im RAM

Die häufigste Vorgehensweise zur Änderung von Parametern zur Laufzeit („Online-Kalibrieren“) ist, die Parameter im zur Verfügung stehenden RAM-Speicher anzulegen.

C-Code-Beispiel:

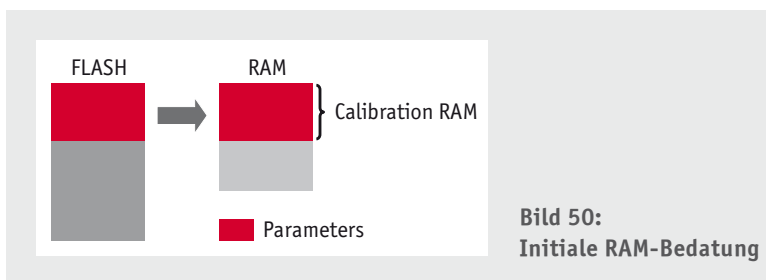
```
#pragma section „RAM_Parameter“  
volatile float factor = 0.5;
```

Hier wird festgelegt, dass die Größe „factor“ eine RAM-Variable mit dem Initialwert 0.5 ist. Beim Kompilieren und Linken des Codes wird für das Objekt „factor“ ein Platz im RAM belegt und die entsprechende RAM-Adresse wird in der Linker-Map-Datei erscheinen. Der Initialwert 0.5 wird im Flash-Speicher abgelegt.

Während des Bootens des Steuergerätes werden alle RAM-Variablen einmalig mit ihren Initialwerten aus dem Flash-Speicher initialisiert. Dies wird meistens bereits im Start-up-Code des Compiler-Herstellers ausgeführt, der Anwendungsprogrammierer muss sich darum nicht kümmern. Die Anwendung nutzt die Werte der Parameter, die im RAM stehen, und diese können über normale XCP-Speicherzugriffe modifiziert werden.

Kalibrierparameter im RAM sind aus der Sicht der Steuergeräte-Software immer noch unveränderlich, d. h., die Anwendung selbst ändert diese nicht. Viele Compiler entdecken diese Tatsache durch Code-Analyse und optimieren den benötigten RAM-Speicherplatz einfach weg. Im Normalfall ist es deshalb auch hier nötig, den Compiler mit dem Attribut „volatile“ am Optimieren zu hindern.

Der RAM-Bereich, in dem die Parameter stehen, wird aus der Sicht des Kalibrier-Tools als Calibration RAM (kalibrierbarer Speicher) bezeichnet.



Das Calibration RAM muss nicht aus einem vollständig zusammenhängenden RAM-Bereich bestehen. Es kann auch auf mehrere Bereiche oder gar beliebig verteilt sein. Allerdings bietet es signifikante Vorteile, die Parameter in einigen wenigen, zusammenhängenden RAM-Bereichen anzuordnen und von anderen RAM-Größen, wie z. B. durch die Applikations-Software veränderlichen Zustandsgrößen und Zwischenergebnissen, zu trennen. Dies ist insbesondere dann wichtig, wenn auch eine Offline-Kalibrierung des Calibration RAMs mit einer Hex-Datei möglich sein soll. Das Kalibrier-Tool muss auf Anwenderwunsch beim Übergang vom Offline- zum Online-Kalibrieren die offline veränderten Größen in das Steuergerät laden können.

Dieser Fall tritt sehr häufig auf. Wenn ein Applikateur beispielsweise am nächsten Arbeitstag sich wieder mit dem Steuergerät verbindet, möchte er an dem Punkt weiterarbeiten, an dem er am Abend zuvor aufgehört hat. Durch das Booten des Steuergerätes ist aber der geflashte Inhalt als Initialbedatung in das RAM kopiert worden. Damit der Anwender die Arbeit vom Vortag fortsetzen kann, muss er seine am Vorabend gespeicherte Parametersatzdatei in das RAM des Steuergerätes laden. Dieser Ladevorgang kann zeitlich optimiert werden, indem die Anzahl der notwendigen Übertragungen auf ein Minimum beschränkt wird. Hierzu ist es von Vorteil, wenn das Tool durch Prüfsummenbildung über größere zusammenhängende Bereiche schnell und sicher ermitteln kann, ob Unterschiede bestehen. Liegen keine Unterschiede zwischen dem Kalibrier-RAM-Inhalt im Steuergerät und der durch das Tool veränderten Datei vor, muss dieser Bereich nicht übertragen werden. Ist der Speicherbereich mit den Kalibriergrößen nicht klar definiert oder befinden sich darin Größen, die durch die Steuergeräte-Software verändert werden, ergibt eine Prüfsummenbildung immer einen Unterschied und die Parameterwerte werden übertragen, entweder aus dem Steuergerät zum XCP-Master oder in umgekehrter Richtung. Je nach Übertragungsgeschwindigkeit und Menge der Daten dauert die Übertragung mehrere Minuten.

Ein weiterer Vorteil klar definierter Speichersegmente besteht in der Nutzung des Speicherbereiches der Initialwerte im Flash-Speicher zur Offline-Kalibrierung. Der Inhalt des Flash-Speichers wird mit Hilfe von flashbaren Hex-Dateien definiert. Kennt das Kalibrier-Tool die Lage der Parameter in der Hex-Datei, kann es deren Wert verändern und durch das Flashen der geänderten Hex-Datei neue Initialwerte im Steuergerät realisieren.

Das Kalibrier-Tool muss nicht nur die Lage der Parameter im RAM, sondern auch die der Initialwerte im Flash kennen. Voraussetzung ist, dass das RAM-Speicher-Segment aus einem identisch aufgebauten Speicher-Segment im Flash durch Kopieren initialisiert wird, wie es bei den meisten Compilern/Linkern umgesetzt ist. Stehen in der A2L-Datei die Adressen der Größen im RAM, muss dem Tool nur noch bekannt sein, welcher Offset auf die Startadresse des Kalibrier-RAMs dazugerechnet werden muss, um auf die Startadresse des entsprechenden Flash-Bereichs zu gelangen. Dieser Offset gilt dann für jede einzelne Größe in der A2L.

Das Kalibrier-Tool kann anschließend entweder selbst flashbare Hex-Dateien dieses Bereichs erzeugen oder es setzt direkt auf den originalen Hex-Dateien des Linkers auf, um die Initialwerte der Parameter in der Hex-Datei zu ändern.

3.3 Flash-Overlay

Viele Mikrocontroller bieten Möglichkeiten an, Speicherbereiche im Flash mit internem oder externem RAM zu überlagern. Dieser Vorgang wird als Flash-Emulation oder Flash-Overlay bezeichnet. Von der Verwendung einer Memory Management Unit bis hin zu dedizierten Mechanismen, die genau diesem Zweck dienen, ist vieles möglich. Die Parameter werden in diesem Fall wie beim Kalibrierkonzept 1 als Parameter im Flash angelegt. Diese Methode hat enorme Vorteile gegenüber dem beschriebenen Kalibrierkonzept 2 „Parameter im RAM“:

- > Flash- und RAM-Adressen werden nicht unterschieden. In der A2L-Datei, der Hex-Datei und der Linker-Map-Datei stehen immer die Flash-Adressen. Das sorgt für klare Verhältnisse, die Hex-Datei ist direkt flashbar und die A2L-Datei passt unmittelbar dazu.
- > Das Overlay lässt sich im Ganzen ein- und ausschalten, wodurch ein blitzschnelles Umschalten zwischen den Werten im Flash und denen im RAM möglich wird. Man spricht hier von der RAM- und der Flash-Seite (Page) eines Speicherbereichs (Segment). XCP unterstützt die Steuerung der Speicherseiten-Umschaltung mit eigenen Kommandos.
- > Die Speicherseiten lassen sich ggf. für den XCP-Zugriff und den Steuergeräte-Zugriff getrennt umschalten, d. h., XCP kann auf eine Speicherseite zugreifen, während die Steuergeräte-Software mit der anderen arbeitet. Dadurch wird z. B. der Download der Offline-Kalibrierdaten ins RAM ermöglicht, während das Steuergerät noch mit den Flash-Daten arbeitet, und mögliche Inkonsistenzen, die bei einem laufenden Steuergerät problematisch sein können, werden vermieden.
- > Die Überlagerung mit RAM muss nicht vollständig erfolgen und kann an den Anwendungsfall angepasst werden. Man vermag mit weniger RAM als mit Flash operieren. Doch dazu später mehr.

Ein typischer Ablauf beim Herstellen der Verbindung vom Kalibrier-Tool zum Steuergerät mit dem anschließenden Download von offline kalibrierten Werten sieht folgendermaßen aus:

Verbindungsaufbau mit ECU	CONNECT
Verbindung XCP-Master zur RAM Page	SET_CAL_PAGE XCP to RAM
Checksummen-Berechnung	CALC_CHECKSUM

Wenn bei der Checksummen-Berechnung über das RAM ein Unterschied erkannt worden ist, erfolgt normalerweise zuerst eine Nachfrage an den Anwender, wie er weiter vorgehen möchte. Soll der Inhalt des Steuergeräte-RAMs an den Master oder der Inhalt einer Datei auf der Master-Seite in das RAM der ECU gesendet werden? Wenn der Anwender sich dafür entscheidet, die Offline-Änderungen ins Steuergerät zu schreiben, sieht der weitere Ablauf wie folgt aus:

ECU soll die Bedatung der Flash Page nutzen	SET_CAL_PAGE ECU to FLASH
Datei vom Master in die RAM Page kopieren	DOWNLOAD ...
ECU soll Bedatung der RAM Page nutzen	SET_CAL_PAGE ECU to RAM

Abschließend wird die Speicherseite immer auf RAM umgestellt, damit Parameter verändert werden können. Der Anwender kann aber auch explizit bestimmen, welche Speicherseite im Steuergerät aktiv sein soll. So kann beispielsweise das Verhalten des RAM-Parametersatzes mit dem Flash-Parametersatz verglichen werden oder im Notfall blitzschnell auf einen erprobten Parametersatz im Flash zurückgeschaltet werden.

3.4 Dynamic Flash-Overlay Allocation

Die bisher beschriebenen Konzepte mit Kalibrier-RAM sind dann unproblematisch, wenn genügend RAM für alle Parameter bereitsteht. Was ist aber, wenn die Gesamtmenge der Parameter nicht in den zur Verfügung stehenden RAM-Bereich passt?

Hier bietet es sich an, das Überlagern von Flash mit RAM dynamisch durchzuführen und erst beim tatsächlichen Schreibzugriff auf einen Parameter den betroffenen Flash-Speicherbereich mit RAM zu überlagern. Dieser Vorgang kann mit einer bestimmten Granularität stattfinden und ist – abhängig von der Implementierung – für das Kalibrier-Tool aus XCP-Sicht transparent. Wenn der XCP-Treiber im Steuergerät einen Schreibzugriff auf das Flash detektiert, der zu einer Änderung führen würde, wird ein Teil des Kalibrier-RAMs dazu verwendet, den entsprechenden Teil des Flashs umzukopieren und den Overlay-Mechanismus für diesen Teil zu aktivieren. Dazu wird das RAM allokiert, also fest zugeordnet und als belegt gekennzeichnet. Die Ressourcen des Kalibrier-RAMs sind allerdings begrenzt. Während des Kalibriervorgangs wird ein einmal allozierter RAM-Bereich nicht mehr freigegeben, sodass das zur Verfügung stehende Kalibrier-RAM mit weiteren Anfragen zur Neige geht. Ist die RAM-Ressource aufgebraucht und wird eine erneute Allokierung angefordert, erhält der Anwender einen Hinweis auf die erschöpfte RAM-Ressource. Ihm wird dann die Möglichkeit angeboten, die bisher gemachten Änderungen zu flashen oder zu speichern. So wird der allokierte RAM-Bereich wieder frei und der Anwender kann erneut kalibrieren. Die Variante, dass das Steuergerät die bisher veränderten Parameter selbsttätig flasht, scheidet aus den bereits beim Kalibrierkonzept „Parameter im Flash“ genannten Gründen meistens aus.

In einigen Fällen kann es passieren, dass der Download eines offline erstellten Parametersatzes aufgrund zu geringer RAM-Ressourcen undurchführbar ist. Hier bleibt nur, diesen zu flashen. Der Anwender kann vom Tool aus jederzeit die Änderungen zurücknehmen, wodurch die allokierten RAM-Blöcke wieder frei werden.

Eine Seitenumschaltung zwischen RAM und Flash Page ist auch mit diesem Konzept ohne Einschränkungen möglich.

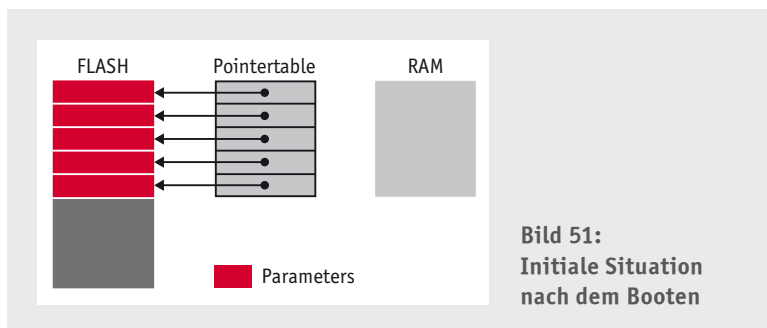
Die Parameter sollten im Flash funktional zusammengehörig angeordnet werden, damit die zur Verfügung stehenden RAM-Blöcke möglichst effizient ausgenutzt werden. Der Softwareentwickler legt dann fest, dass die Größen, die thematisch zusammengehören, auch in einem zusammenhängenden Speicherbereich liegen. Nach dem Kopieren in das RAM stehen die für die Abstimmung dieser Funktion notwendigen Parameter komplett bereit.

3.5 RAM-Pointer-basiertes Kalibrierkonzept nach AUTOSAR

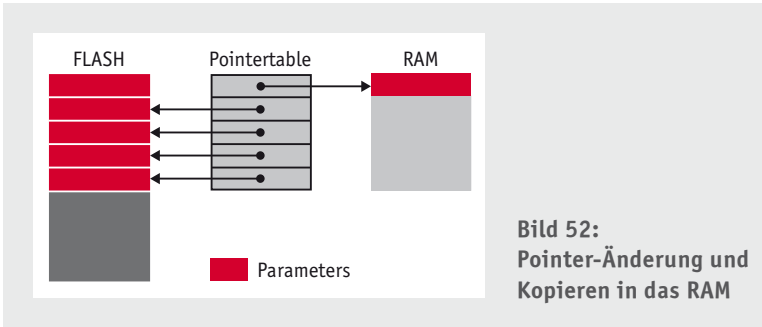
Das Konzept setzt nicht unbedingt die Nutzung eines AUTOSAR-Betriebssystems voraus, auch in einem anderen Umfeld – z. B. ohne Betriebssystem – kann es zum Einsatz kommen. Das Konzept weist eine prinzipielle Ähnlichkeit mit dem vorherigen auf. Der wesentliche Unterschied besteht darin, dass das Ersetzen von Flash mit RAM nicht durch Hardware-Mechanismen, sondern durch Softwaremechanismen realisiert wird. Die Kalibrierparameter werden von der Steuergeräte-Software immer über Pointer referenziert. Der Zugriff auf Flash- oder RAM-Inhalte geschieht durch Änderung dieser Pointer. Die zu ändernden Flash-Parameter werden in einen festgelegten Block mit freiem RAM kopiert. Dieses Verfahren kann, genauso wie die vorherigen, aus XCP-Sicht völlig transparent implementiert werden. Alternativ wählt der Anwender des Kalibrier-Tools explizit die zu verändernden Größen aus, indem etwa eine Vorauswahl der gewünschten Parameter stattfindet. Das hat den Vorteil, dass die Ressourcenbelegung und die Auslastung für den Anwender sichtbar ist und er nicht mitten in der Arbeit von der Situation, dass Speicher fehlt, überrascht wird.

3.5.1 Single-Pointer-Konzept

Die Pointer-Tabelle steht im RAM. Beim Booten des Steuergerätes zeigen alle Pointer auf die Parameterwerte im Flash. Das Kalibrier-RAM ist zwar in seiner Lage und Größe bekannt, jedoch stehen nach dem Booten noch keine Parameterwerte darin. Die Anwendung arbeitet zunächst komplett aus dem Flash heraus.



Wählt ein Anwender zum ersten Mal nach dem Booten einen Parameter aus der A2L-Datei aus und möchte schreibend darauf zugreifen, wird damit zunächst ein Kopiervorgang innerhalb des Steuergerätes ausgelöst. Der XCP-Slave stellt fest, dass die Adresse, auf die der Zugriff erfolgen soll, im Flash-Bereich liegt, und kopiert den Wert des Parameters in das Kalibrier-RAM. Damit nun die Anwendung den Parameterwert nicht weiterhin aus dem Flash-Bereich holt, sondern aus dem RAM-Bereich, erfolgt zusätzlich eine Änderung in der Pointer-Tabelle:



Die Anwendung bezieht nach wie vor über die Pointer-Tabelle den Wert des Parameters. Da nun aber der Pointer auf die RAM-Adresse zeigt, wird der Wert von dort geholt. Somit kann der Anwender per XCP den Wert des Parameters ändern und über die Messung beobachten, wie sich die Änderung auswirkt. Der Nachteil dieses Verfahrens ist, dass für jeden Parameter ein Eintrag in einer Pointer-Tabelle zur Verfügung stehen muss, und der ist wiederum mit einem erheblichen zusätzlichen RAM-Speicher-Bedarf für die Pointer-Tabelle verbunden.

Die nächste Abbildung verdeutlicht das Problem. Drei Parameter eines PID-Reglers (P, I und D) sind im Flash-Bereich eines Steuergerätes enthalten. In der Pointer-Tabelle stehen schon die RAM-Adressen und die Parameterwerte im RAM sind auch verändert.

Parameter	Flash		Pointertable		RAM	
	Addr.	Content	Addr.		Addr.	Content
P	0x0000100A	0x11	0x000A100A	→	0x000A100A	0x44
I	0x000012BC	0x22	0x000A100B	→	0x000A100B	0x55
D	0x00007234	0x33	0x000A100C	→	0x000A100C	0x66

Bild 53: Pointer-Tabelle für einzelne Parameter

Kalibrierkonzepte sind deshalb so wichtig, weil die RAM-Ressourcen knapp sind. Durch große RAM-Pointer-Tabellen wird das ad absurdum geführt.

Damit nicht für jeden einzelnen Parameter ein Pointer angelegt werden muss und das Verfahren als solches zum Einsatz kommt, können die Parameter zu Strukturen zusammengefasst werden. Dadurch wird pro Struktur nur ein Pointer benötigt. Mit der Auswahl eines Parameters durch den Anwender wird nicht nur dieser Parameter, sondern die gesamte damit verbundene Struktur ins RAM kopiert. Der Granularität der Strukturen kommt auf diese Weise eine besondere Bedeutung zu. Wählt man große Strukturen, sind nur wenige Pointer notwendig. Das bedeutet wiederum, dass mit der Entscheidung für einen bestimmten Parameter eine entsprechend große Struktur in den RAM-Bereich kopiert wird und somit die Begrenzung des Kalibrier-RAMs rasch erreicht werden kann.

Beispiel:

Das Kalibrier-RAM soll 400 Bytes groß sein. In der Software sind vier Strukturen mit folgenden Größen definiert:

Struktur A: 250 Bytes

Struktur B: 180 Bytes

Struktur C: 120 Bytes

Struktur D: 100 Bytes

Wählt der Anwender einen Parameter aus der Struktur A aus, so werden die 250 Bytes aus dem Flash in das Kalibrier-RAM kopiert und der Anwender hat auf alle Parameter, die sich in Struktur A befinden, per XCP Zugriff. Beschränkt sich die Kalibrieraufgabe auf die Parameter dieser Struktur, reicht das Kalibrier-RAM völlig aus. Wählt der Anwender nun einen weiteren Parameter aus, der sich jedoch in der Struktur C befindet, müssen auch diese 120 Bytes in das Kalibrier-RAM kopiert werden. Da die 400 Bytes im Kalibrier-RAM genügen, kann der Anwender auf alle Parameter der Strukturen A und C gleichzeitig zugreifen.

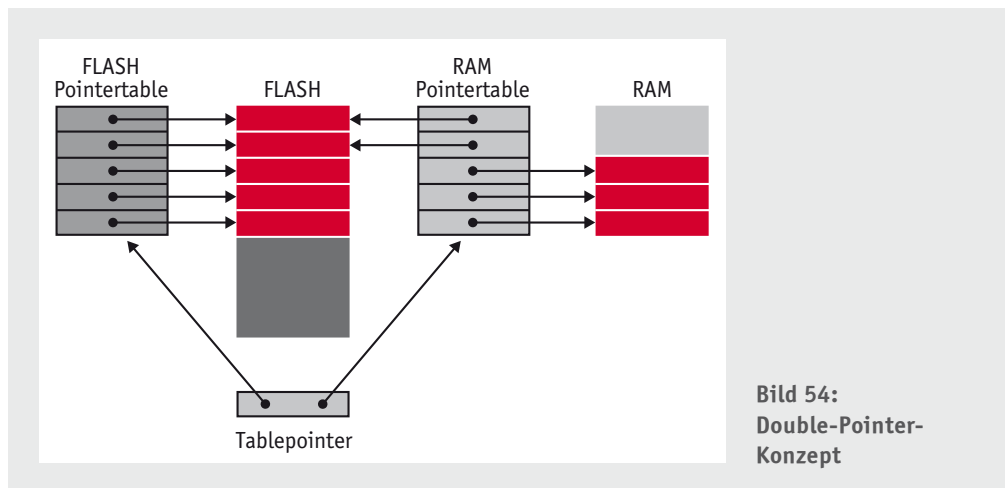
Befindet sich ein weiterer ausgewählter Parameter statt in der Struktur C in B, müssten zu den 250 Bytes der Struktur A noch die 180 Bytes der Struktur B ins RAM kopiert werden. Da der Platz im RAM dafür nicht ausreicht, hat der Anwender zwar Zugriff auf die Parameter der Struktur A, aber nicht auf die Daten der Struktur B, weil das Steuergerät den Kopierbefehl nicht ausführen kann.

3.5.2 Double-Pointer-Konzept

Ein Nachteil des Single-Pointer-Konzepts ist, dass eine Speicherseitenumschaltung nicht einfach realisierbar ist. Das Kalibrier-Tool könnte zwar zur Seitenumschaltung einfach die Pointer-Tabelle komplett beschreiben, dies ist jedoch nicht in beliebig kurzer Zeit machbar, sodass sich daraus temporäre Inkonsistenzen und Nebeneffekte ergeben können. Eine tooltransparente Implementierung würde den Speicherplatzbedarf für die Pointer-Tabelle verdoppeln, da bei der Umschaltung der Speicherseite ins Flash eine Kopie der vorherigen Pointer-Tabelle mit den RAM-Pointern angelegt werden müsste.

Für Anwendungen mit großen Pointer-Tabellen, einer transparenten Implementierung oder einer völlig konsistenten Umschaltung besteht die Möglichkeit der Erweiterung auf ein Double-Pointer-Konzept. Um das zu erläutern, kehren wir nochmals zur initialen Bedatung des RAMs zurück.

Bild 50 stellt die Pointer-Tabelle dar. Sie liegt im RAM. Wie bereits erwähnt, muss diese Tabelle aus dem Flash in das RAM kopiert werden. Also liegt diese Tabelle im Flash-Speicher vor. Verwendet man nun einen weiteren Pointer (einen Tabellen-Pointer), der entweder auf die Pointer-Tabelle im RAM oder im Flash zeigt, kommt man zur Double-Pointer-Lösung.



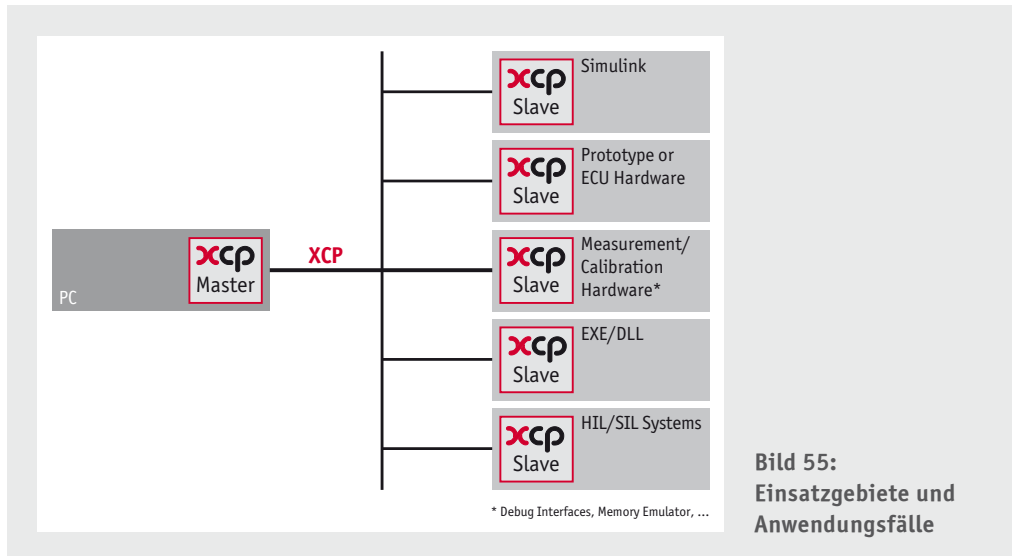
Der Zugriff auf die Parameterwerte erfolgt zunächst über den Tabellen-Pointer. Deutet der Tabellen-Pointer auf die Pointer-Tabelle im RAM, greift die Anwendung letztendlich auf die eigentlichen Parameter über den Inhalt der RAM-Pointer-Tabelle zu. Die geringe Zugriffsgeschwindigkeit und die Entstehung von mehr Programm-Code sind die Nachteile dieser Lösung.

3.6 Flash-Pointer-basiertes Kalibrierkonzept

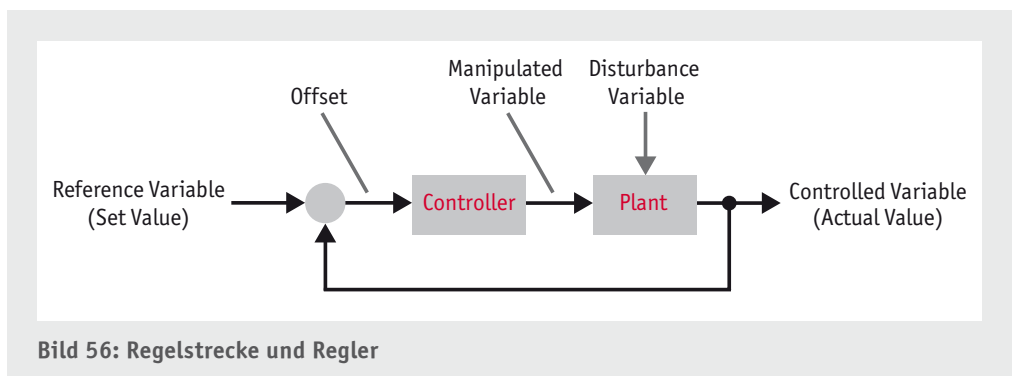
Dieses Verfahren wurde vor einigen Jahren von ZF Friedrichshafen unter dem Namen „InCircuit2“ patentiert und hat starke Ähnlichkeiten mit dem Pointer-basierten Konzept nach AUTOSAR. Auch hier erfolgt der Zugriff der Steuergeräteanwendung auf die Parameterdaten mit Hilfe einer Pointer-Tabelle. Allerdings liegt diese Pointer-Tabelle nicht im RAM, sondern im Flash. Änderungen an der Pointer-Tabelle können daher nur durch Flash-Programmierung vorgenommen werden. Eine tooltransparente Implementierung ist nicht möglich. Der Vorteil besteht im eingesparten RAM-Speicher für die Pointer-Tabelle.

4 Einsatzgebiete für XCP

Wenn Steuergeräte-Applikateure sich über den Einsatz von XCP Gedanken machen, sind sie meist auf die Nutzung des Protokolls im Steuergerät fixiert.



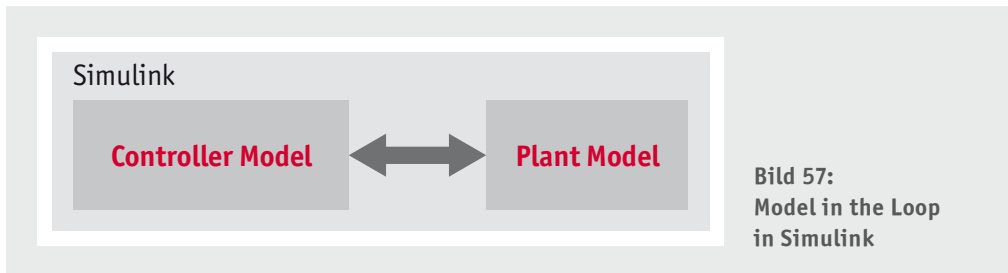
Bewegt man sich entlang des Entwicklungsprozesses, stößt man auf die unterschiedlichsten Lösungsansätze für die Entwicklung der Elektronik und Software. HIL, SIL und Rapid Prototyping sind Stichwörter, die verschiedene Szenarien beschreiben. Diese haben immer eine Regelstrecke (englisch „plant“) und einen Regler (englisch „controller“) gemeinsam.



Im Sinne einer Automotive-Entwicklung entspricht der Regler dem Steuergerät und die Regelstrecke ist das physikalische System, das es zu regeln gilt, wie Getriebe, Motor, Seitenspiegel ...

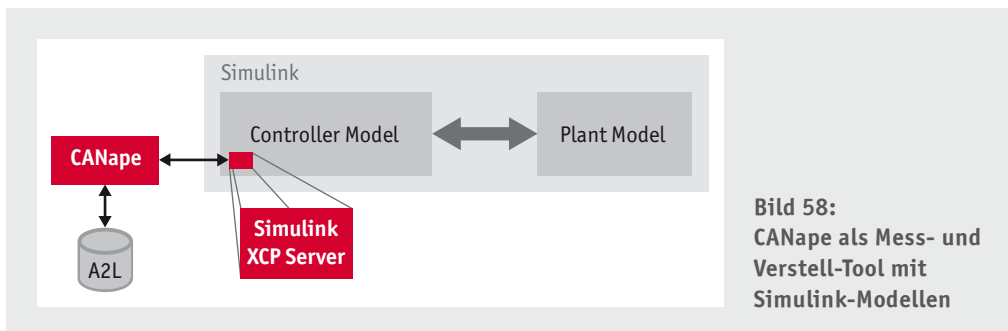
Die grobe Unterteilung der Entwicklungsansätze erfolgt danach, ob der Regler oder die Regelstrecke real oder simuliert ablaufen. Dabei ergibt sich eine Kombinatorik, die mit den Begriffen SIL, HIL etc. näher beschrieben wird.

4.1 MIL: Model in the Loop



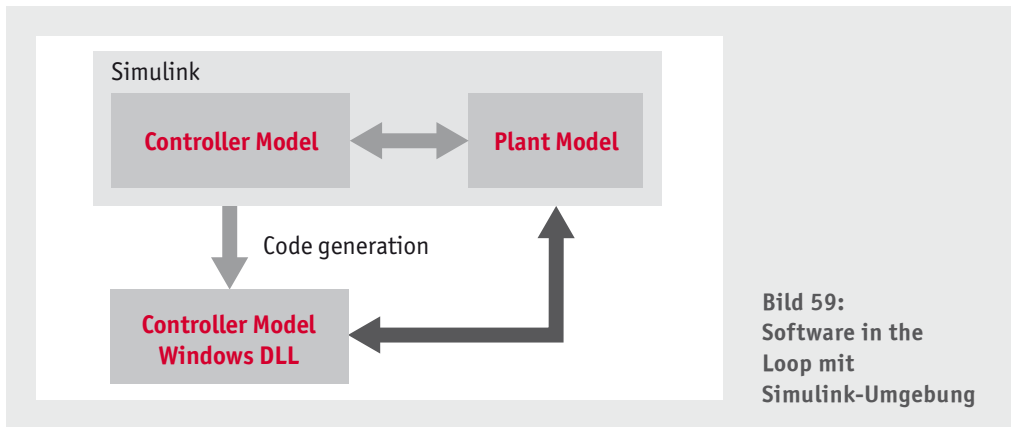
In dieser Entwicklungsumgebung werden sowohl der Regler als auch die Regelstrecke als Modell simuliert. Im gezeigten Beispiel laufen beide Modelle in Simulink als Ablaufumgebung ab. Zur Analyse des Verhaltens stehen Ihnen hierzu die Möglichkeiten der Simulink-Ablaufumgebung zur Verfügung.

Um schon in einer frühen Entwicklungsphase den Komfort eines Mess- und Verstell-Tools wie CANape zu nutzen, kann ein XCP-Slave in das Controller-Modell integriert werden. Der Slave generiert in einem Autorenschritt die zum Modell passende A2L und schon hat der Anwender den kompletten Bedienkomfort mit Visualisierung der Abläufe in Grafikfenstern, dem Zugriff auf Kennlinien und -felder und vieles mehr.



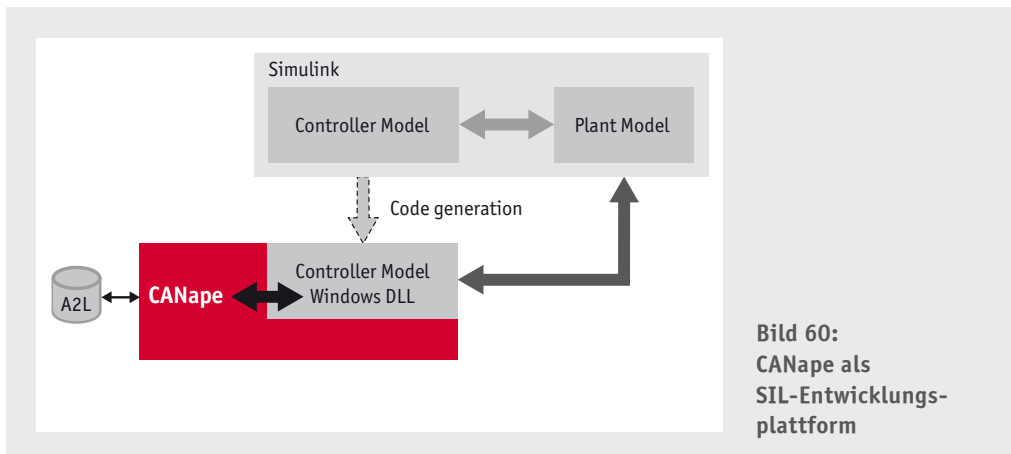
Dazu ist weder ein Code-Generierungsschritt noch eine Instrumentierung des Modells notwendig. Über das XCP werden auch Zeitstempel mit versendet. CANape passt sich dabei komplett dem zeitlichen Verhalten der Simulink-Ablaufumgebung an. Ob das Modell schneller oder langsamer als in Echtzeit abläuft, ist ohne Bedeutung. Nutzt beispielsweise der Funktionsentwickler im Modell den Simulink-Debugger, um durch das Modell zu steppen, so nimmt CANape dennoch die durch XCP übertragene Zeit als Referenzzeit.

4.2 SIL: Software in the Loop



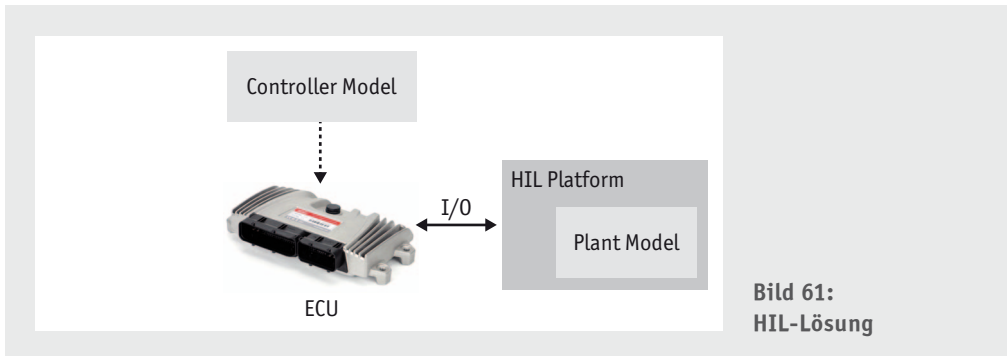
In diesem Entwicklungsschritt wird aus dem Modell des Reglers Code generiert, der dann in einer PC-basierten Ablaufumgebung zum Einsatz kommt. Natürlich kann der Regler auch ohne jeglichen modellbasierten Ansatz entwickelt worden sein. Die Regelstrecke ist nach wie vor simuliert. Mit Hilfe von XCP kann der Regler gemessen und verstellt werden. Stammt der Controller aus einem Simulink-Modell, wird über einen Code-Generierungsschritt (Simulink Coder mit dem Target „CANape“) C-Code für eine DLL und die dazugehörige A2L erzeugt. Erfolgt die Controller-Entwicklung auf der Basis von handgeschriebenem Code, wird dieser in ein C++-Projekt eingebettet, das mit CANape ausgeliefert wird.

Nach dem Kompilieren und Linken wird die DLL im Kontext von CANape genutzt. Die Algorithmen in der DLL können mit Unterstützung der XCP-Anbindung genauso gemessen und verstellt werden, als wäre die Anwendung bereits in einem Steuergerät integriert.



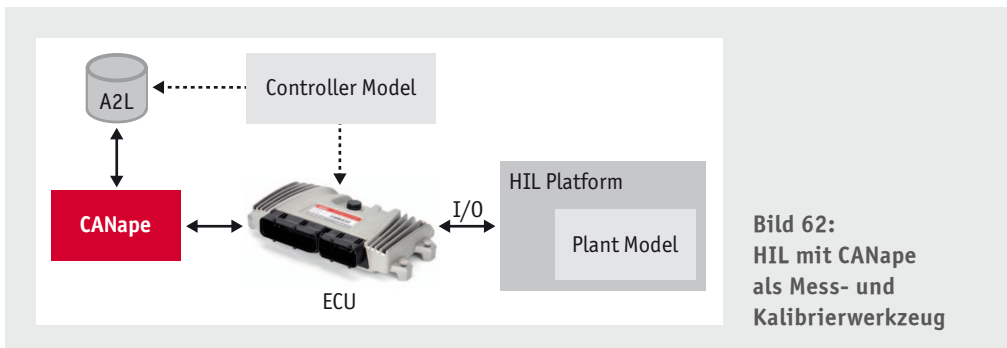
4.3 HIL: Hardware in the Loop

Sogenannte HIL-Systeme gibt es in unterschiedlichen Ausprägungen. Sie reichen von sehr einfachen, kostengünstigen Systemen bis hin zu sehr großen und teuren Ausbaustufen. Das nachfolgende Bild zeigt das Grobkonzept:



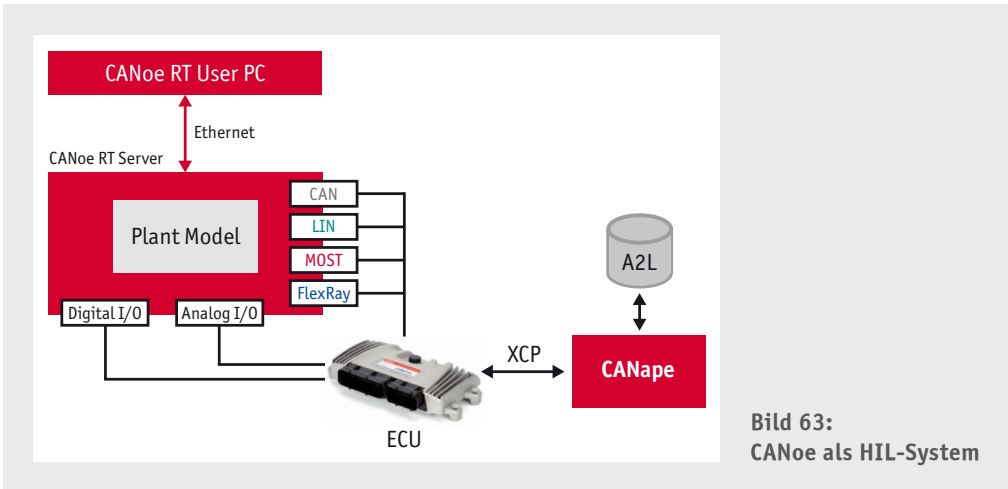
Der Controller-Algorithmus läuft in einer Mikrocontroller-Plattform (z. B. dem Steuergerät), die Regelstrecke wird nach wie vor simuliert. Je nach Größe und Komplexität der Regelstrecke und der notwendigen I/Os steigen die Anforderungen an die HIL-Plattform und damit auch die Kosten. Da das Steuergerät in Echtzeit läuft, muss auch das Modell der Regelstrecke in Echtzeit gerechnet werden.

Hier nun XCP zur Optimierung einzuführen erscheint trivial, da es sich um ein weiteres Steuergerät handelt. Das Ganze sieht dann so aus:

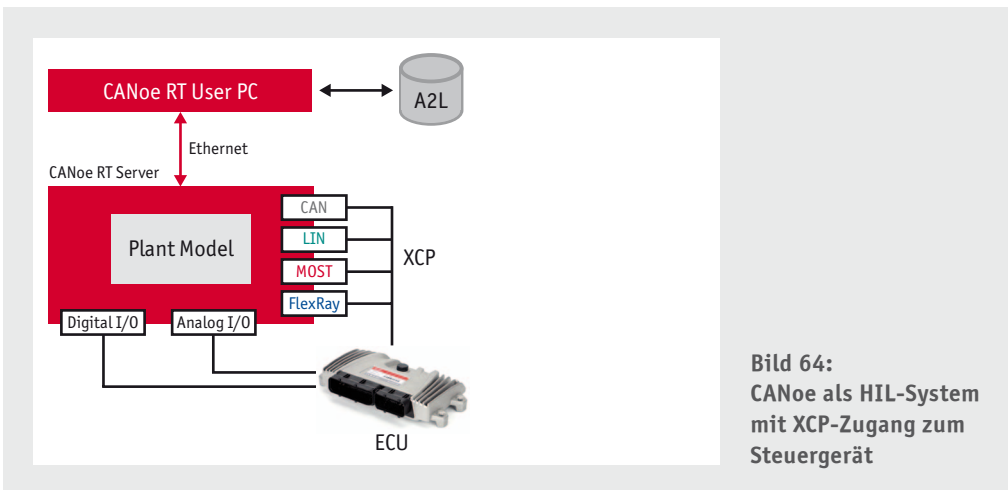


Über XCP hat der Anwender aus CANape heraus Zugriff auf die Algorithmen im Steuergerät.

Auch das Vector Tool CANoe wird von vielen Kunden als HIL-System genutzt. Mit CANoe schaut ein HIL-System z. B. folgendermaßen aus:



Durch die Möglichkeit, für Testzwecke direkt aus CANoe heraus auf XCP-Daten zuzugreifen, ergibt sich auch folgende Variante:



Hier läuft das Modell der Regelstrecke auf dem CANoe Echtzeit-Server. Gleichzeitig erfolgt der XCP-Zugriff auf das Steuergerät ebenfalls aus CANoe heraus. Einem Werkzeug wird damit der gleichzeitige Zugriff auf die Regelstrecke und den Controller erlaubt.

Um das Bild abzurunden, soll noch eine weitere Option einer HIL-Lösung skizziert werden. Die Regelstrecke kann z. B. auch als DLL in CANape ablaufen. Somit hat der Anwender über XCP vollen Zugriff auf die Regelstrecke und auf den Regler.

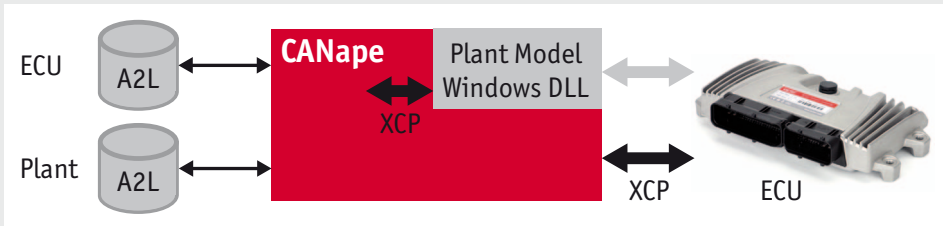


Bild 65: CANape als HIL-Lösung

4.4 RCP: Rapid Control Prototyping

In dieser Entwicklungsphase läuft der Regler-Algorithmus auf einer Echtzeit-Hardware anstelle eines Steuergerätes. Diese Situation ergibt sich häufig dann, wenn die dafür notwendige Steuergeräte-Hardware noch nicht zur Verfügung steht. Als passende Hardware kommen mehrere Plattformen in Frage: von einfachen Evaluation Boards bis hin zu speziellen automotivetauglichen Hardware-Lösungen, je nachdem, welche weiteren Anforderungen erfüllt werden müssen. Auch hier hilft die Integration mit XCP, eine herstellerunabhängige Werkzeugkette aufzubauen.

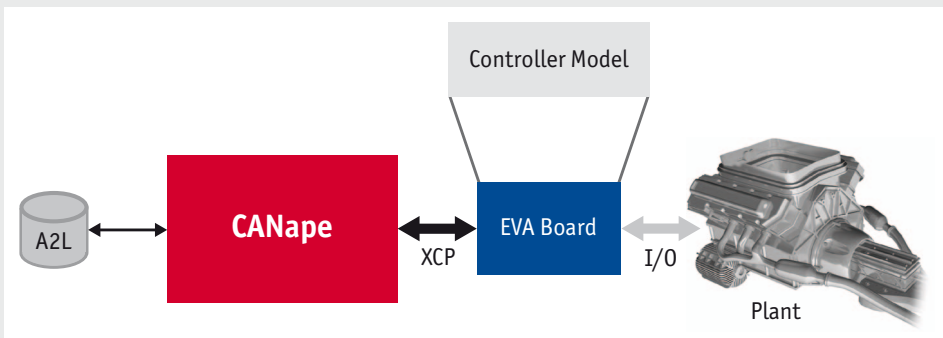


Bild 66: RCP-Lösung

Die Begriffe „Rapid“ und „Prototyping“ umreißen die Aufgabenstellung sehr gut. Es geht darum, möglichst schnell einen Funktionsprototyp zu entwickeln, ihn auf der Ablaufumgebung zu nutzen und zu testen. Das setzt einfache Arbeitsschritte über den ganzen Prozess hinweg voraus.

In der Literatur wird der RCP-Ansatz häufig in zwei Bereiche unterteilt: Fullpassing und Bypassing

Wie in Bild 66 dargestellt, läuft der gesamte Regler auf einer separaten Echtzeit-Hardware. Dieses Verfahren wird Fullpassing genannt, weil der gesamte Regler auf der Controller-Hardware läuft. Sie muss über die notwendigen I/Os verfügen, um sich mit der Regelstrecke verbinden zu können. Die technischen Anforderungen an die I/Os sind sehr häufig nur mit einer entsprechenden Leistungselektronik realisierbar.

Nicht nur die I/Os stellen eine Herausforderung dar, sondern oftmals werden auch Funktionselemente der Steuergerätesoftware (z. B. Netzwerkmanagement) benötigt, um das Funktionieren in einem komplexeren Verbund zu ermöglichen. Nutzt man aber ein komplettes Steuergerät für das Rapid Control Prototyping anstelle einer allgemeinen Controller-Plattform, so stehen die Komplexität des Flash-Vorgangs, der Umfang der Gesamtsoftware etc. der Anforderung „Rapid“ entgegen.

Zusammengefasst: Die Nutzung eines kompletten Steuergerätes als Ablaufumgebung des Reglers bietet den Vorteil, dass die notwendige Hard- und Softwareinfrastruktur für die Regelstrecke vorhanden ist. Der Nachteil liegt jedoch in der hohen Komplexität.

Das Konzept des Bypassings wurde entwickelt, um die Vorteile der Steuergeräte-Infrastruktur zu nutzen, ohne die Nachteile der hohen Komplexität in Kauf nehmen zu müssen.

4.5 Bypassing

Im Bild 67 ist das Steuergerät an die Regelstrecke angeschlossen. Die notwendigen I/Os und Softwarekomponenten stehen im Steuergerät zur Verfügung. In der Bypassing-Hardware läuft ein Algorithmus A1, der in der ECU in der Version A vorkommt. A1 ist eine neue Variante des Algorithmus und soll nun an der realen Regelstrecke erprobt werden.

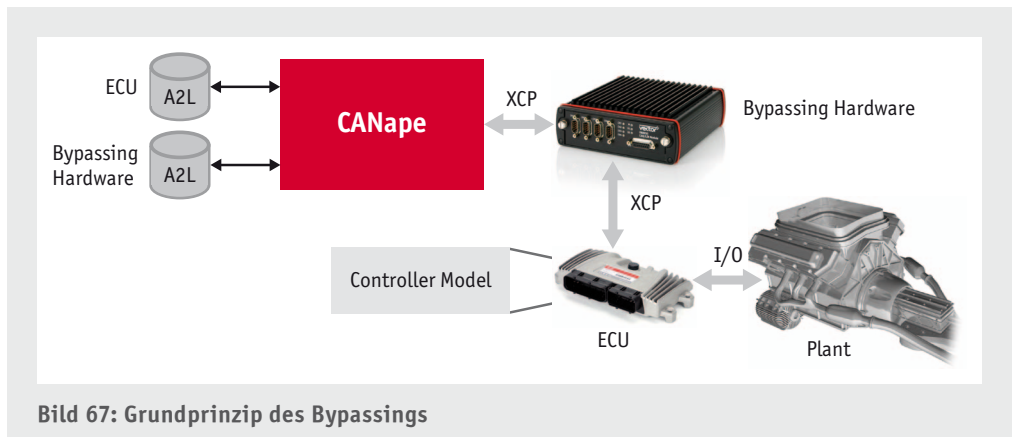
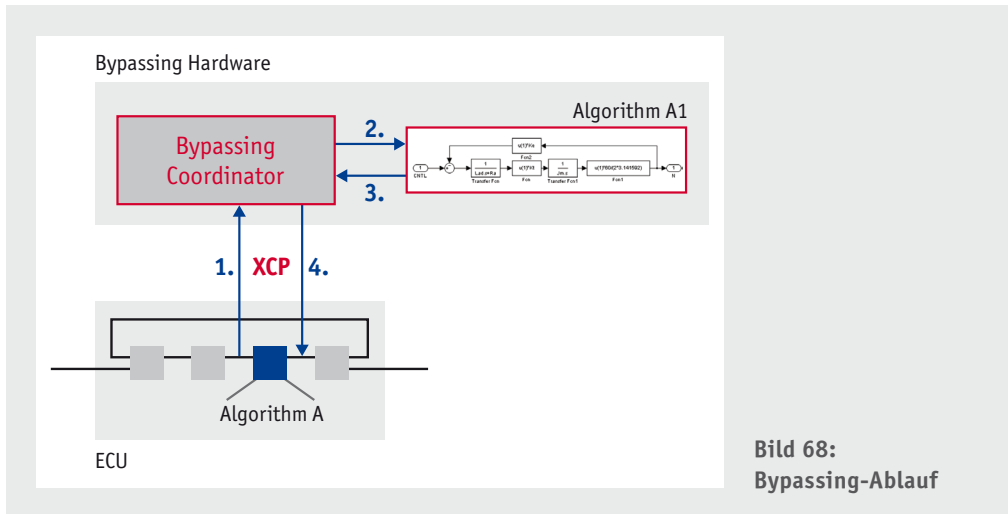


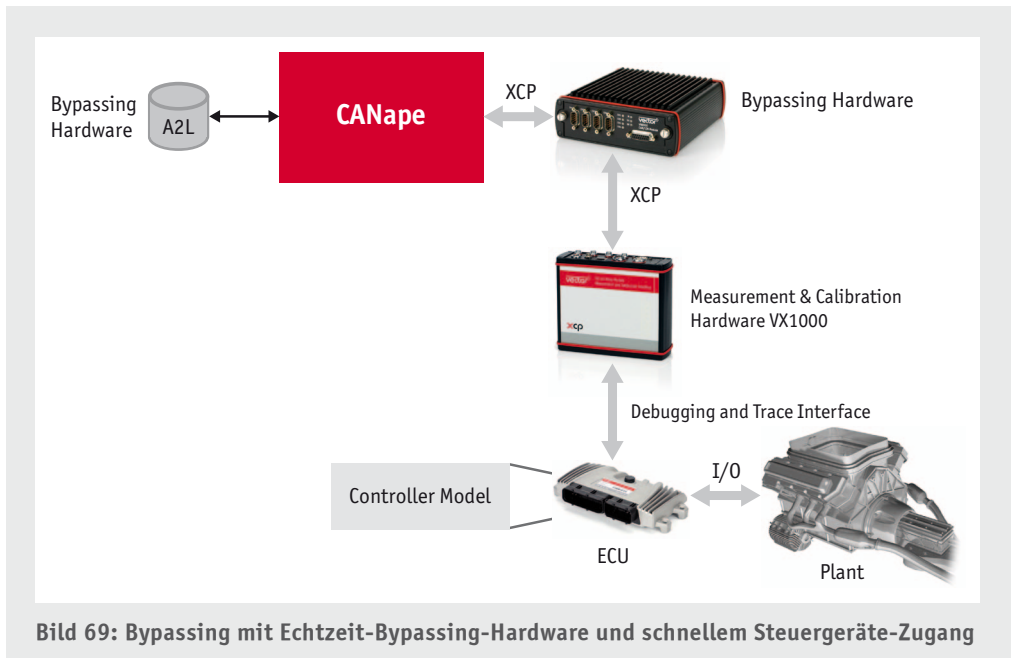
Bild 67: Grundprinzip des Bypassings

Die Bypassing-Hardware (im Bild ein VN8900) und die ECU sind miteinander über XCP verbunden. Dabei geht es zum einen darum, die für den Algorithmus A1 notwendigen Daten per DAQ aus dem Steuergerät zu holen, und zum anderen darum, die Ergebnisse von A1 wieder in das Steuergerät hineinzustimulieren. Das folgende Bild demonstriert den schematischen Ablauf:



In der ECU ist ein blauer Funktionsblock abgebildet, in dem der Algorithmus A abläuft. Damit nun A1 zum Einsatz kommen kann, werden die Daten, die als Eingangsgröße in den Algorithmus A hineingelangen, per DAQ aus dem Steuergerät herausgemessen. Der Bypassing-Koordinator übernimmt in Schritt 1 die Daten und übergibt sie im Schritt 2 dem Algorithmus A1. A1 wird auf der Bypassing-Hardware berechnet und das Ergebnis wird im Schritt 3 wieder dem Bypassing-Koordinator übergeben und in Schritt 4 per STIM an die ECU übertragen. Die Daten werden an die „Stelle“ geschrieben, an der der nachfolgende Funktionsblock im Slave seine Eingangsgrößen erwartet. Somit wurde die Berechnung des Algorithmus A1 und nicht von A im Gesamtablauf des Steuergerätes verwendet. Diese Methode erlaubt die Kombination des schnellen („rapid“) Austauschs der Algorithmen auf der Bypassing-Hardware mit der Nutzung der I/Os und der Basis-Software des Steuergerätes.

Die Performance-Grenzen eines XCP-on-CAN-Treibers wirken sich natürlich auch auf das Bypassing aus. Werden kurze Bypassing-Zeiten benötigt, so kann der Zugriff per DAQ und STIM in das Steuergerät auch über Debugging- bzw. Trace-Schnittstellen des Controllers erfolgen. Die Vector VX1000 Mess- und Kalibrierhardware setzt die Daten aus der Controller-Schnittstelle in einen XCP-on-Ethernet-Datenstrom um. Damit können Daten von bis zu ein Megabyte pro Sekunde in das Steuergerät transportiert werden.



4.6 Verkürzung der Iterationszyklen mit virtuellen Steuergeräten

Um den Algorithmus im Steuergerät mit Hilfe von XCP zu optimieren, ist eine Stimulation mit Daten notwendig. Der Vorgang kann im Rahmen von Testfahrten im Steuergerät erfolgen. Mit XCP steht aber auch eine andere Lösung zur Verfügung, bei der der Algorithmus nicht auf einem Steuergerät läuft, sondern in Form eines ausführbaren Codes auf dem PC oder als Modell in Simulink in Form eines „virtuellen Steuergerätes“. Dieses virtuelle Steuergerät muss nicht in Echtzeit ablaufen, da in diesem Fall keine Kopplung an ein reales System existiert. Es kann – je Rechenleistung des PCs – wesentlich schneller laufen.

Die Stimulation des Algorithmus erfolgt durch eine vorher aufgezeichnete Messdatei, in der alle Signale enthalten sind, die als Eingangssignale in den Algorithmus benötigt werden. Über XCP wird die Verbindung zu CANape aufgebaut. Der Anwender kann die Parametrierung und die Messkonfiguration durchführen. Anschließend startet er die Ausführung. Dabei werden die Daten aus der Testfahrt als Stimulanz in den Algorithmus eingespeist und gleichzeitig die gewünschten Messgrößen aus der Anwendung herausgemessen und wiederum in einer Messdatei gespeichert.

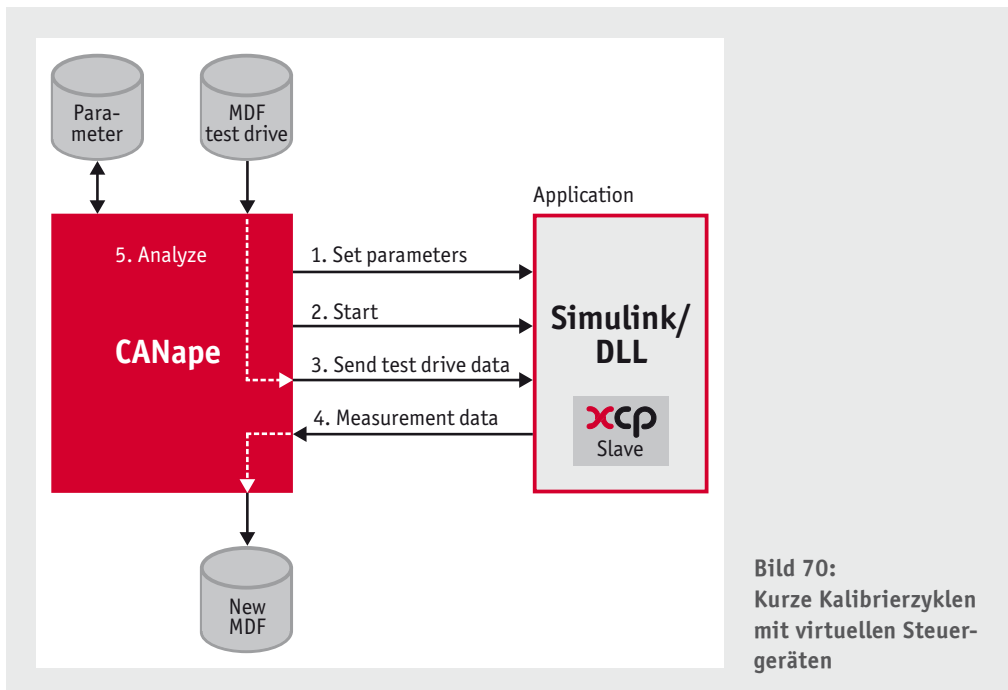
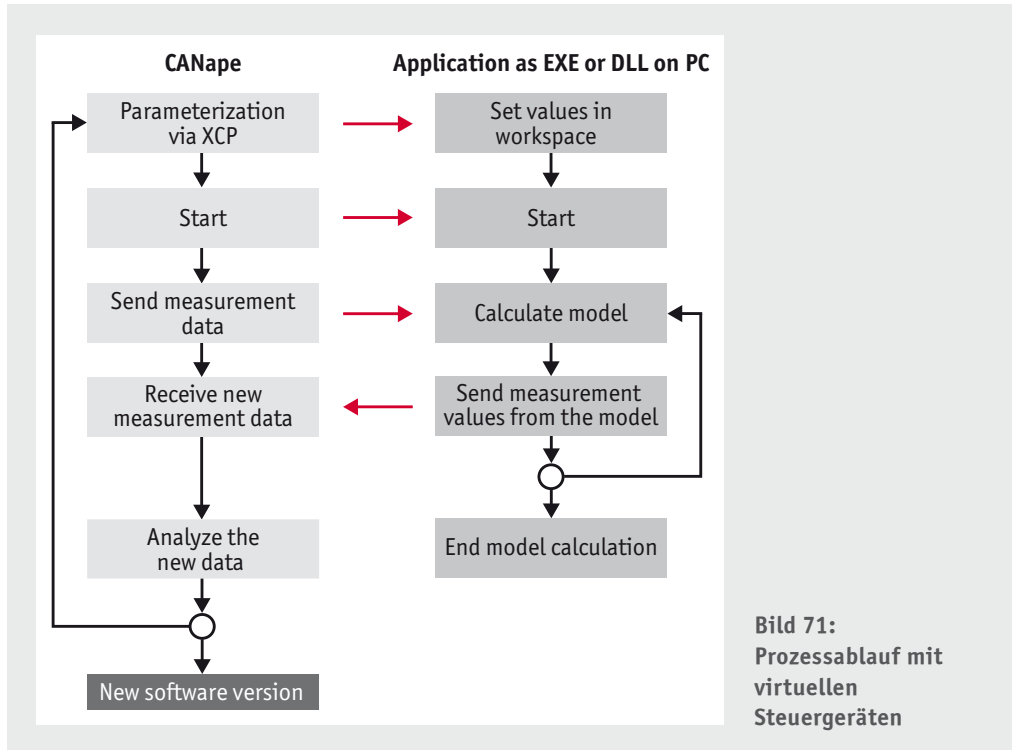


Bild 70:
Kurze Kalibrierzyklen
mit virtuellen Steuer-
geräten

Nach dem Ende der Berechnung steht dem Anwender eine neue Messdatei zur Analyse des Verhaltens des Steuergerätes zur Verfügung. Die zeitliche Länge der neuen Messdatei entspricht genau der Länge der Eingangsmessdatei. Liegt die Dauer einer Testfahrt bei einer Stunde, so rechnet der Algorithmus auf dem PC z. B. in wenigen Sekunden die gesamte Testfahrt durch. Dann liegt ein Messergebnis vor, das einem Test von einer Stunde Dauer entspricht. Aus der Analyse der Daten trifft der Anwender Entscheidungen für die Parametrierung und der Iterationszyklus beginnt von vorne.



Zu der Verkürzung der Iterationszyklen kommt noch die Stimulierung des Algorithmus mit immer den gleichen Daten. Das macht die Ergebnisse mit unterschiedlichen Parametern deutlich vergleichbarer, da die Ergebnisse nur durch die unterschiedlichen Parameter beeinflusst werden.

Dieser Prozess kann selbstverständlich automatisiert werden. Durch die integrierte Skriptsprache von CANape erfolgt eine Analyse der Messergebnisse, aus denen Parametervstellungen abgeleitet und automatisiert ausgeführt werden. Auch die Steuerung durch ein externes Optimierungswerkzeug wie z. B. MATLAB ist über die CANape Automatisierungs-Schnittstelle möglich.

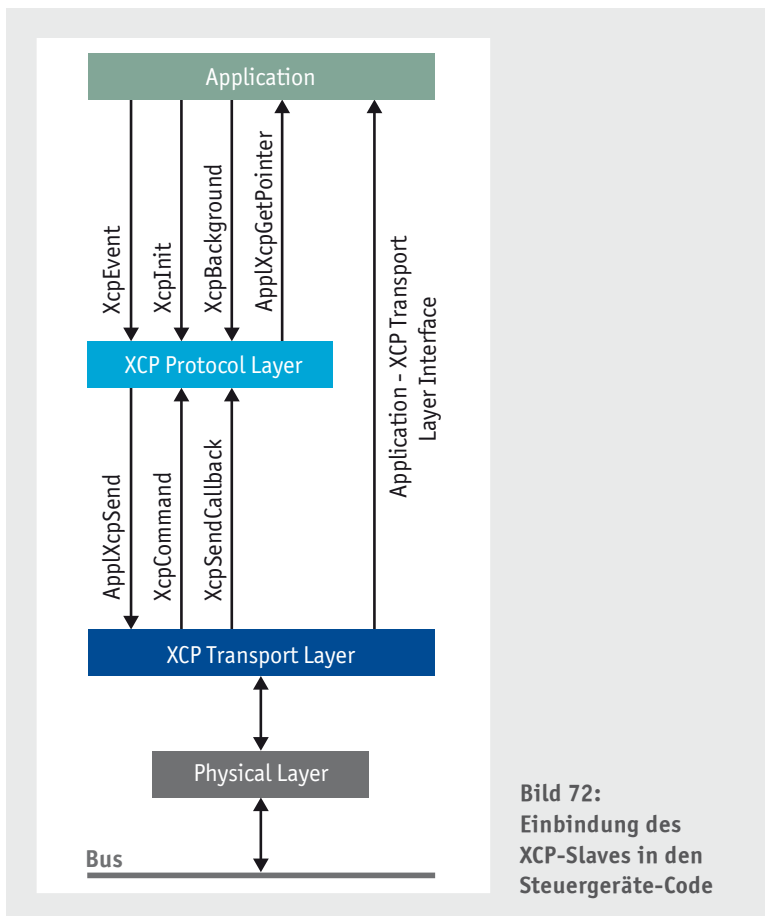
5 Beispiel einer XCP-Implementierung

Damit ein Steuergerät über XCP kommunizieren kann, bedarf es der Integration eines XCP-Treibers in die Anwendung des Steuergerätes. Als Beispiel wird im Folgenden der kostenlose XCP-Treiber, den Sie im Download-Center der Vector Homepage (www.vector.com/xcp-treiber) herunterladen können, beschrieben. In diesem Paket sind auch einige Musterimplementierungen für verschiedene Transport-Layer und Zielplattformen enthalten. Der Treiber umfasst den Protokoll-Layer mit der Basisfunktionalität, die zum Messen und Verstellen notwendig ist. Features wie Kaltstartmessung, Stimulation oder Flashen sind darin nicht enthalten. Eine vollständige Implementierung, integriert in die Vector CANbedded- oder AUTOSAR-Umgebung, können Sie als Produkt erwerben.

Die XCP-Protokollschicht setzt auf der XCP-Transportschicht auf, die wiederum auf der eigentlichen Buskommunikation basiert. Die Implementierung der XCP-Protokollschicht umfasst nur eine einzige C-Datei und einige H-Dateien (`xcpBasix.c`, `xcpBasic.h`, `xcp_def.h` und `xcp_cfg.h`). In den Beispielen finden sich Implementierungen für verschiedene Transport-Layer, z. B. Ethernet und RS232. Im Falle von CAN ist die Transportschicht im Normalfall besonders einfach, die verschiedenen XCP-Botschafts-Typen werden direkt auf CAN-Botschaften abgebildet. Es gibt dann je einen festen CAN-Identifizier für die Sende- und die Empfangsrichtung.

Die Softwareschnittstelle zwischen der Transport- und der Protokollschicht ist sehr einfach. Sie umfasst nur wenige Funktionen:

- > Empfängt der Slave über den Bus eine XCP-Botschaft, kommt diese zunächst im Kommunikationstreiber an, der die Nachricht an die XCP-Transportschicht weiterleitet. Über den Funktionsaufruf `XcpCommand()` informiert die Transportschicht die Protokollschicht über die Botschaft.
- > Möchte die XCP-Protokollschicht eine Botschaft (z. B. eine Antwort auf ein XCP-Kommando des Masters oder eine DAQ-Botschaft) versenden, wird die Botschaft per Aufruf an die Funktion `ApplXcpSend()` an die Transportschicht weitergeleitet.
- > Die Transportschicht teilt der Protokollschicht durch den Funktionsaufruf `XcpSendCallBack()` mit, dass die Botschaft erfolgreich versendet worden ist.



Auch das Interface zwischen der Anwendung und der Protokollschicht lässt sich über nur vier Funktionen realisieren:

- > Die Anwendung aktiviert den XCP-Treiber mit Hilfe von `XcpInit()`. Dieser Aufruf erfolgt einmalig beim Startvorgang.
- > Mit `XcpEvent()` informiert die Anwendung den XCP-Treiber darüber, dass ein bestimmtes Ereignis eingetreten ist (z. B. „das Ende eines Berechnungszyklus ist erreicht“).
- > Der Aufruf `XcpBackground()` ermöglicht es dem XCP-Treiber, bestimmte Tätigkeiten im Hintergrund auszuführen (z. B. die Kalkulation einer Checksumme).
- > Da in A2L-Files die Adressen immer als 40-Bit-Wert definiert werden (32-Bit-Adresse, 8-Bit-Adress-Extension), nutzt der XCP-Treiber die Funktion `ApplXcpGetPointer()`, um aus einer A2L-konformen Adresse einen Pointer zu erhalten.

Diese Schnittstellen reichen aus, um die grundlegenden Funktionalitäten zum Messen und Verstellen zu integrieren. Erst für erweiterte Funktionen, wie Seitenumschaltung, Identifikation oder Seed & Key, werden weitere Schnittstellen benötigt. Sie sind in der Dokumentation des Treibers ausführlich beschrieben.

5.1 Beschreibung der Funktionen

void XcpInit (void)

Aufgabe:

Initialisierung des XCP-Treibers

Beschreibung:

Mit XcpInit() aktiviert die Anwendung den XCP-Treiber. Dieser Befehl muss genau einmal ausgeführt werden, bevor irgendeine XCP-Treiberfunktion aufgerufen werden darf.

void XcpEvent (BYTE event)

Aufgabe:

Die Anwendung informiert den XCP-Treiber darüber, welches Ereignis eingetreten ist. Dabei wird jedem Ereignis eine eindeutige Eventnummer zugeordnet.

Beschreibung:

Bei der Zusammenstellung der Messkonfiguration im Mess- und Kalibrierwerkzeug selektiert der Anwender, mit welchem Event welche Messwerte synchron erfasst werden sollen. Die Information über Messwerte und Events stammt aus der A2L. Die gewünschte Messkonfiguration wird dem XCP-Treiber in Form von DAQ-Listen mitgeteilt.

Beispiel einer Event-Definition eines Motorsteuergerätes:

```
XcpEvent (1);    // Event 1 steht für die 10-ms-Task
XcpEvent (2);    // Event 2 steht für die 100-ms-Task
XcpEvent (5);    // Event 5 steht für die 1-ms-Task
XcpEvent (8);    // Event 8 wird für zündwinkelsynchrone Messungen genutzt
```

BYTE XcpBackground (void)

Aufgabe:

Führt Hintergrundtätigkeiten des XCP-Treibers aus.

Beschreibung:

Die Funktion sollte periodisch in einer Hintergrund- oder Idle-Task aufgerufen werden. Sie wird vom XCP-Treiber beispielsweise zur Prüfsummenberechnung verwendet, da die Berechnung einer längeren Prüfsumme in XcpCommand() unakzeptabel lange dauern könnte. Bei jedem Aufruf an XcpBackground() wird eine Teilprüfsumme von 256 Bytes berechnet. Die Dauer einer Prüfsummenberechnung hängt daher von der Aufrufhäufigkeit von XcpBackground() ab. Sonst bestehen keine Anforderungen an die Aufrufhäufigkeit und die Periodizität. Der Return-Wert 1 zeigt an, dass aktuell eine Prüfsummenberechnung abläuft.

void XcpCommand (DWORD* pCommand)

Aufgabe:

Interpretation eines XCP-Kommandos.

Beschreibung:

Jedes Mal, wenn die Transportschicht eine XCP-Nachricht erhält, muss diese Funktion aufgerufen werden. Der Parameter ist ein Pointer auf die Nachricht.

void ApplXcpSend (BYTE len, BYTE *msg)

Aufgabe:

Übergabe einer zu sendenden Nachricht an den Transport-Layer.

Beschreibung:

Die Protokollschicht sendet über diesen Aufruf eine Botschaft an die Transportschicht zum Versenden an den Master. Über den Aufruf XcpSendCallBack wird ein Handshake-Verfahren zwischen Protokoll- und Transportschicht realisiert.

BYTE XcpSendCallBack (void)

Aufgabe:

Mit dem Callback informiert die Transportschicht die Protokollschicht über das erfolgreiche Versenden der letzten Nachricht, die an ApplXcpSend() übergeben worden ist.

Beschreibung:

Die Protokollschicht wird so lange keinen ApplXcpSend()-Befehl aufrufen, bis über XcpSendCallBack() angezeigt wird, dass die vorangegangene Botschaft erfolgreich versendet worden ist. XcpSendCallBack() liefert den Wert 0 (FALSE) zurück, wenn der XCP-Treiber im Leerlauf ist. Falls weitere zu sendende Nachrichten anstehen, wird ApplXcpSend() direkt aus XcpSendCallBack() heraus aufgerufen.

BYTE *ApplXcpGetPointer (BYTE addr_ext, DWORD addr)

Aufgabe:

Umwandlung einer A2L-konformen Adresse in einen Pointer.

Beschreibung:

Die Funktion bildet die 40-Bit-A2L-konforme Adressierung (32-Bit-Adresse + 8-Bit-Adress-Extension), die vom XCP-Master versendet wird, auf einem gültigen Pointer ab. Die Adress-Extension kann z. B. zur Unterscheidung unterschiedlicher Adressbereiche oder Speicherarten verwendet werden.

5.2 Parametrierung des Treibers

Der XCP-Treiber ist in vielerlei Hinsicht skalierbar und parametrierbar, um den unterschiedlichsten Funktionsumfängen, Transportprotokollen und Zielplattformen gerecht zu werden. Alle Einstellungen erfolgen in der Parameterdatei `xcp_cfg.h`. Im einfachsten Fall sieht diese folgendermaßen aus:

```
/* Define protocol parameters */
#define kXcpMaxCTO    8    /* Maximum CTO Message Length */
#define kXcpMaxDTO    8    /* Maximum DTO Message Length */
#define C_CPUYPE_BIGENDIAN /* Byte order Motorola */

/* Enable memory checksum */
#define XCP_ENABLE_CHECKSUM
#define kXcpChecksumMethod XCP_CHECKSUM_TYPE_ADD14

/* Enable calibration */
#define XCP_ENABLE_CALIBRATION
#define XCP_ENABLE_SHORT_UPLOAD

/* Enable data acquisition */
#define XCP_ENABLE_DAQ
#define kXcpDaqMemSize (512) /* Memory space reserved for DAQ */
#define XCP_ENABLE_SEND_QUEUE
```

Für einen CAN-Transport-Layer wird die passende CTO- und DTO-Größe von acht Bytes eingestellt. Der Treiber muss wissen, ob er auf einer Plattform mit Motorola- oder Intel-Byteorder läuft, in diesem Fall einer Motorola-CPU (Big-Endian). Die restlichen Parameter aktivieren die Funktionalitäten: Messen, Verstellen und Prüfsummenberechnung. Für die Prüfsummenberechnung wird der Algorithmus eingestellt (hier Summierung aller Bytes in ein DWORD) und zum Messen wird die Größe des verfügbaren Speichers angegeben (hier 512 Bytes). Der Speicher wird hauptsächlich zur Ablage der DAQ-Listen und zur Pufferung der Daten während der Messung benötigt. Die Größe bestimmt daher die maximal mögliche Anzahl von Messsignalen. In der Dokumentation des Treibers finden Sie genauere Angaben zur Abschätzung der benötigten Größe.

Abkürzungsverzeichnis

A2L	File Extension for an ASAM 2MC Language File
AML	ASAM 2 Meta Language
ASAM	Association for Standardisation of Automation and Measuring Systems
BYP	Bypassing
CAL	Calibration
CAN	Controller Area Network
CCP	CAN Calibration Protocol
CMD	Command
CS	Checksum
CTO	Command Transfer Object
CTR	Counter
DAQ	Data Acquisition, Data Acquisition Packet
DTO	Data Transfer Object
ECU	Electronic Control Unit
ERR	Error Packet
EV	Event Packet
FIBEX	Field Bus Exchange Format
LEN	Length
MCD	Measurement Calibration and Diagnostics
MTA	Memory Transfer Address
ODT	Object Descriptor Table
PAG	Paging
PGM	Programming
PID	Packet Identifier
RES	Command Response Packet
SERV	Service Request Packet
SPI	Serial Peripheral Interface
STD	Standard
STIM	Data Stimulation Packet
TCP/IP	Transfer Control Protocol/Internet Protocol
TS	Time Stamp
UDP/IP	Unified Data Protocol/Internet Protocol
USB	Universal Serial Bus
XCP	Universal Measurement and Calibration Protocol
Download	Versand der Daten vom Master zum Slave
Upload	Versand der Daten vom Slave zum Master

Literaturverzeichnis

Die Spezifikation von XCP erfolgt durch ASAM (Association for Standardisation of Automation and Measuring Systems).

Sie finden Details zum Protokoll und zu ASAM unter **www.asam.net**

Web-Adressen

Standardisierungsgremien:

> ASAM, XCP-Protokollspezifische Dokumente, A2L-Spezifikation, **www.asam.net**

Zulieferer für Entwicklungssoftware:

> MathWorks, Informationen zu MATLAB, Simulink und Simulink Coder, **www.mathworks.com**

> Vector Informatik GmbH, Demoversion CANape, kostenloser und frei verfügbarer XCP-Treiber (Basisversion), umfassende Informationen zu den Themen Steuergeräte-Kalibrierung, Test und Simulation, **www.vector.com**

Abbildungsverzeichnis

Bild 1: Grundlegende Kommunikation mit einer Ablaufumgebung	8
Bild 2: Das Interface-Modell von ASAM	9
Bild 3: Ein XCP-Master kann gleichzeitig mit mehreren Slaves kommunizieren	10
Bild 4: Aufteilung des XCP-Protokolls in Protokoll- und Transportschicht	14
Bild 5: Slaves können in vielfältigen Ablaufumgebungen genutzt werden	15
Bild 6: XCP-Paket	19
Bild 7: Überblick über die XCP Packet Identifier (PID)	19
Bild 8: XCP-Kommunikationsmodell mit CTO/DTO	20
Bild 9: Die Botschafts-Identifikation	21
Bild 10: Zeitstempel	21
Bild 11: Datenfeld im XCP-Paket	22
Bild 12: Die drei Modi des XCP-Protokolls: Standard, Block und Interleaved Mode	24
Bild 13: Übersicht über den Aufbau des CTO-Pakets	25
Bild 14: Trace-Darstellung eines Verstellvorgangs	30
Bild 15: Übertragung einer Parametersatzdatei in das RAM des Steuergerätes	31
Bild 16: Hex-Fenster	32
Bild 17: Adressinformation der Größe „Triangle“ aus der A2L-Datei	33
Bild 18: Polling-Kommunikation im Trace-Fenster von CANape	34
Bild 19: Ereignisse im Steuergerät	35
Bild 20: Eventdefinition in einer A2L	35
Bild 21: Zuordnung von „Triangle“ zu den möglichen Events in der A2L	36
Bild 22: Auswahl der Events (Measurement mode) pro Messgröße	36
Bild 23: Auszug aus dem CANape Trace-Fenster einer DAQ-Messung	37
Bild 24: ODT: Zuordnung RAM-Adressen zu DAQ DTO	38
Bild 25: DAQ-Liste mit drei ODTs	39
Bild 26: Statische DAQ-Listen	39
Bild 27: Dynamische DAQ-Listen	40
Bild 28: Event für DAQ und STIM	41
Bild 29: Aufbau XCP-Paket bei DTO-Übertragungen	42
Bild 30: Identifikationsfeld mit absoluten ODT-Nummern	43
Bild 31: ID-Feld mit relativen ODT- und absoluten DAQ-Nummern (ein Byte)	43
Bild 32: ID-Feld mit relativen ODT- und absoluten DAQ-Nummern (zwei Bytes)	43
Bild 33: ID-Feld mit relativen ODT- und absoluten DAQ-Nummern sowie Füll-Byte (gesamt vier Bytes)	44
Bild 34: Festlegung, welcher Busteilnehmer welche Botschaft sendet	45
Bild 35: Darstellung eines CAN-Netzwerks	46
Bild 36: Beispiel einer XCP-on-CAN-Kommunikation	47
Bild 37: Darstellung einer XCP-on-CAN-Botschaft	47
Bild 38: Die Knoten K und L sind redundant miteinander verbunden	48
Bild 39: Kommunikation per Slot-Definition	48
Bild 40: Darstellung einer FlexRay-Kommunikationsmatrix	49
Bild 41: Darstellung der FlexRay LPDUs	50
Bild 42: Zuordnung der XCP-Kommunikation zu LPDUs	51
Bild 43: XCP-Paket mit TCP/IP bzw. UDP/IP	52
Bild 44: XCP-on-SxI-Paket	53

Bild 45: Speicherdarstellung	54
Bild 46: Darstellung der Treibereinstellung für den Flash-Bereich	56
Bild 47: Darstellung des Blocktransfer-Modus	59
Bild 48: Parameter in einem Verstellfenster	64
Bild 49: Signaldarstellung über der Zeit	64
Bild 50: Initiale RAM-Bedatung	74
Bild 51: Initiale Situation nach dem Booten	78
Bild 52: Pointer-Änderung und Kopieren in das RAM	79
Bild 53: Pointer-Tabelle für einzelne Parameter	79
Bild 54: Double-Pointer-Konzept	81
Bild 55: Einsatzgebiete und Anwendungsfälle	84
Bild 56: Regelstrecke und Regler	84
Bild 57: Model in the Loop in Simulink	85
Bild 58: CANape als Mess- und Verstell-Tool mit Simulink-Modellen	85
Bild 59: Software in the Loop mit Simulink-Umgebung	86
Bild 60: CANape als SIL-Entwicklungsplattform	86
Bild 61: HIL-Lösung	87
Bild 62: HIL mit CANape als Mess- und Kalibrierwerkzeug	87
Bild 63: CANoe als HIL-System	88
Bild 64: CANoe als HIL-System mit XCP-Zugang zum Steuergerät	88
Bild 65: CANape als HIL-Lösung	89
Bild 66: RCP-Lösung	89
Bild 67: Grundprinzip des Bypassings	90
Bild 68: Bypassing-Ablauf	91
Bild 69: Bypassing mit Echtzeit-Bypassing-Hardware und schnellem Steuergeräte-Zugang	92
Bild 70: Kurze Kalibrierzyklen mit virtuellen Steuergeräten	93
Bild 71: Prozessablauf mit virtuellen Steuergeräten	94
Bild 72: Einbindung des XCP-Slaves in den Steuergeräte-Code	99

Anhang – XCP-Lösungen bei Vector

Der XCP-Standard wurde maßgeblich von Vector mitgeprägt. Das umfangreiche Know-how und viele Erfahrungen flossen in die umfassende XCP-Unterstützung ein:

Tools

- > Der primäre Einsatzbereich von **CANape** ist die optimale Parametrierung (Kalibrierung) von elektronischen Steuergeräten. Während der Laufzeit des Systems verstellen Sie Parameterwerte und erfassen gleichzeitig Messsignale. Die physikalische Anbindung zwischen CANape und Steuergerät erfolgt über XCP (für alle standardisierten Transportprotokolle) oder CCP.
- > Komplette Werkzeugkette zur Generierung und Verwaltung der notwendigen A2L-Beschreibungsdateien (**ASAP2 Tool-Set** sowie **CANape** mit dem auch als Stand-alone-Tool erhältlichen **ASAP2 Editor**).
- > Mit **CANoe.XCP** greifen Sie für Test- und Analyseaufgaben auf steuergeräteinterne Werte zu.

Embedded Software

Kommunikationsmodule mit getrennten Transportschichten für CAN, FlexRay und Ethernet:

- > **XCP Basic** – kostenlos zum Download unter www.vector.com/xcp-treiber, enthält nur XCP-Basisfunktionen. Die Konfiguration des XCP-Protokolls und die Anpassung der Transportschicht erfolgt manuell im Source Code. Die Integration von XCP Basic in Ihr Projekt müssen Sie selber vornehmen.
- > **XCP Professional** – enthält nützliche Erweiterungen der ASAM-Spezifikation und erlaubt eine Werkzeug-basierte Konfiguration. Erhältlich für die Vector Basissoftware CANbedded.
- > **MICROSAR XCP** – enthält den Funktionsumfang von XCP Professional und basiert auf den AUTOSAR-Spezifikationen. Erhältlich für die Vector Basissoftware MICROSAR.

Dienstleistungen

- > **Beratung** zum Einsatz von XCP in Ihren Projekten
- > **Integration** von XCP in Ihrem Steuergerät

Schulungen

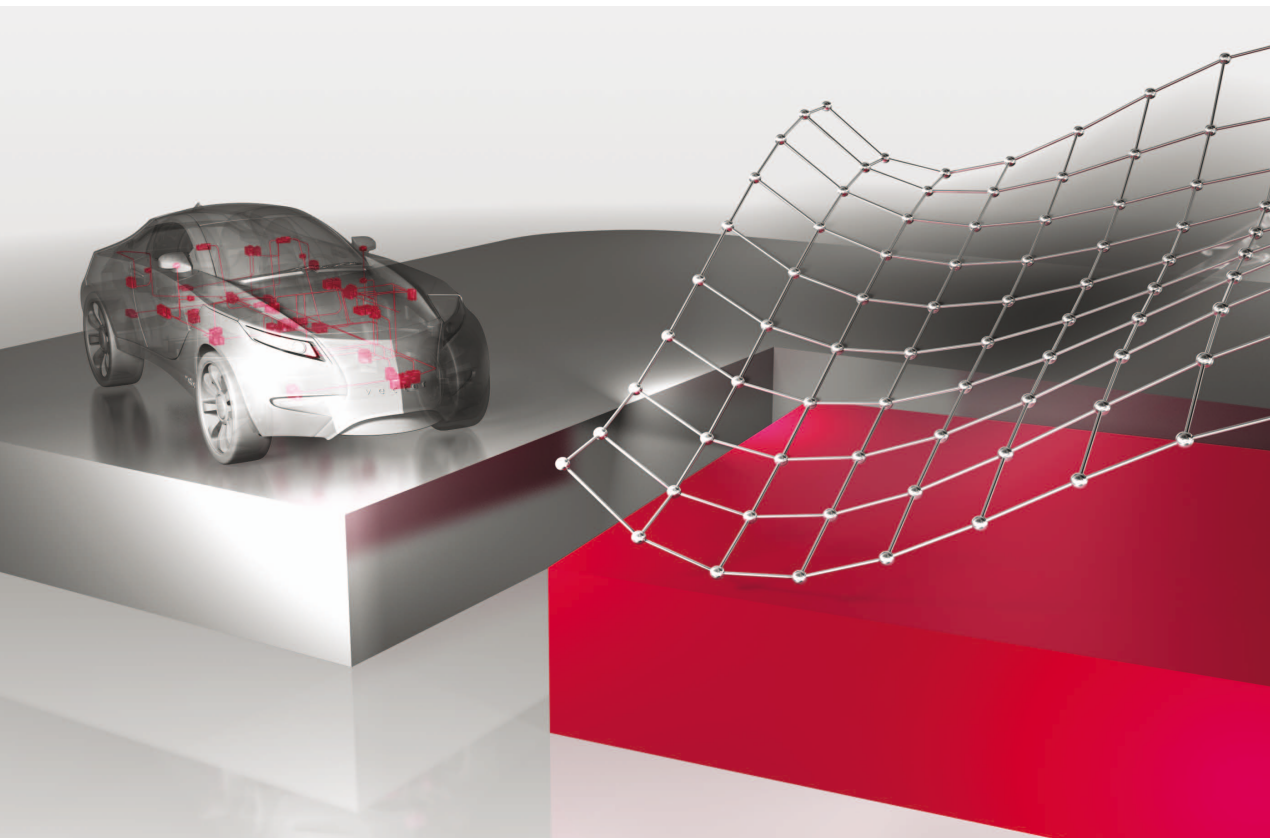
- > Die zugrunde liegenden Mechanismen und Modelle des Protokolls lernen Sie im „**XCP Seminar**“ kennen.
- > Im Workshop „**CANape mit XCP on FlexRay**“ werden Ihnen die FlexRay-Grundlagen und die Besonderheiten bei XCP on FlexRay, insbesondere die dynamische Bandbreitenverwaltung, erläutert

Spezielle XCP-Unterstützung von CANape

CANape unterstützte als erstes MCD-Tool die Spezifikation XCP 1.0 und war auch der erste XCP on FlexRay Master auf dem Markt.

Eine technische Besonderheit von XCP on FlexRay ist die dynamische Bandbreitenverwaltung. Dabei identifiziert CANape die für XCP im FlexRay-Cluster vorgesehene freie Bandbreite und weist diese dem aktuellen Applikationsdatenverkehr dynamisch und sehr effizient zu. Die freie Bandbreite wird somit optimal für die XCP-Kommunikation genutzt.

Des Weiteren verfügt CANape über eine DLL-Schnittstelle. Diese ermöglicht die Unterstützung von XCP auf jeder beliebigen (anwenderdefinierten) Transportschicht. Damit integrieren Sie beliebige Messtechnik und proprietäre Protokolle in CANape. Ein Code Generator unterstützt Sie bei der Erstellung des XCP-spezifischen Anteils eines solchen Treibers.



Sachwortverzeichnis

A

A2L 10, 18, 25, 33, 35, 39–41, 50,
51, 55, 56, 61, 63–68, 79, 85,
86, 99–101, 104
Adress-Extension 29, 33, 38, 99,
101
AML 25, 66, 104
ASAM 7, 53, 104
ASAP2 Tool-Set 68

B

Bandbreitenoptimierung 34
Buslast 34
BYP 104
Bypassing 44, 90–92

C

CAN 7, 14, 24, 29, 33, 38, 45–47,
49, 67, 92, 104
CCP 7, 39, 45, 104
CMD 20, 25, 50, 104
CTO 20–22, 25, 44, 104
CTR 52, 53, 104
CYCLE_REPETITION 50

D

DAQ 21, 22, 32–44, 58, 60, 69,
91, 92, 98, 100, 104
DBC 45
DOWNLOAD 30, 31, 59
DTO 20–22, 33, 37, 38, 42, 104

E

ECU 66, 88, 90, 91, 104
EEPROM 16, 31, 60
ERR 20, 25, 28, 104
EV 29, 104
Event 35, 39–41, 58, 60, 69, 100

F

FIBEX 49, 50, 51
Flash-Speicher 16, 17, 54–57, 60
FLX_CHANNEL 50
FLX_LPDU_ID 50
FLX_SLOT_ID 50

Fullpassing 90

G

GET_CAL_PAGE 25, 55
GET_DAQ_PROCESSOR_INFO 44, 58, 69

H

HIL 84, 87–89

K

Kompilieren 68, 72, 74, 86

L

Linken 68, 72, 74, 86
LPDU 50

M

MIL 85
MTA 30, 104

N

Nutz-Bytes 29, 38, 45, 47

O

ODT 37–40, 42–44, 58, 69, 104
OFFSET 50

P

PAG 104
PGM 104
PID 8, 19, 21, 25, 42, 64, 104
Polling 33, 34, 36

R

RAM 16–18, 30–32, 37–39, 56, 57, 60,
72, 74, 77–80
Reboot 32
Reifegrad 31
RES 20, 21, 28, 50, 104

S

Segment	54–56
SEGMENT_NUMBER	55
Seite	37, 54–56, 59, 61
Seitenumschaltung	55
Sektor	54
SERV	29, 104
SET_CAL_PAGE	25, 55
SHORT_UPLOAD	30, 33, 59
SIL	84, 86
STIM	22, 33, 41, 42, 44, 58, 69, 91, 92, 104
Stimulation	29, 54, 61, 93

T

Task	100
TCP/IP	51, 52, 104

U

UDP/IP	51, 52, 104
USB	53, 104

V

Virtuelles Steuergerät	93
Volatile	72, 73
VX1000	92

Mehr Informationen



Besuchen Sie unsere Internetseite für:

- > News
- > Produkte
- > Demo-Software
- > Support
- > Trainings und Workshops
- > Kontaktadressen

www.vector.com

