Masterarbeit

KONZEPT ZUR ECHTZEIT-KOPPLUNG AUTOMOBILER STEUERGERÄTE AN EIN PRÜFSTANDS-AUTOMATISIERUNGSSYSTEM

ausgeführt am

CAMPUS 02 GRAZ
FACHHOCHSCHULE DER WIRTSCHAFT

Fachhochschul-Masterstudiengang
Automatisierungstechnik-Wirtschaft

von
Ing. Georg Fuchs, BSc
1510322001

betreut und begutachtet von
Dipl.-Ing. Peter Priller

Graz, im Jänner 2017

.....
Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

DANKSAGUNG

Zu Beginn möchte ich mich bei allen Beteiligten und Außenstehenden dieser Masterarbeit für ihren Rat, ihre Zeit und ihre Motivation bedanken.

Besonderer Dank gilt meinem Betreuer Dipl.-Ing. Peter Priller, der die Umsetzung der Masterarbeit ermöglicht hat.

Des Weiteren bin ich meinem Betreuer im Unternehmen Ing. Markus Hechtl, der mich im Namen der Magna Steyr Engineering AG & Co KG bei dieser Arbeit unterstützt hat, sehr dankbar. Auch meinem Arbeitskollegen Ing. Patrick Robier, der mir dieses Thema nahegelegt hat, gilt besonderer Dank.

Zu guter Letzt will ich noch meiner Lebensgefährtin, meiner gesamten Familie und meinen Freunden für ihre moralische Unterstützung, ihre investierte Zeit und ihre aufbauenden Worte danken.

KURZFASSUNG

Die vorliegende Masterarbeit beschäftigt sich mit der Erstellung eines Konzepts zur Verbesserung der Bedienung bzw. Flexibilität einer vorhandenen Prüfstandsanlage für Antriebsstrangkomponenten durch den Einsatz einer Schnittstelle zur automatisierten Bedatung von Steuergeräten sowie des neuentwickelten CAN FD-Bussystems. Im Zuge dessen wird unter anderem den Fragen nachgegangen inwiefern die automatisierte Bedatung eines Steuergeräts umgesetzt werden kann bzw. welche Erneuerungen und Möglichkeiten die Entwicklung des CAN FD-Protokolls in Bezug auf die moderne Prüfstandstechnik mit sich gebracht hat.

Im ersten Teil der Arbeit werden die Grundlagen der digitalen Bustechnik, die wesentlichen Bussysteme in der Fahrzeugtechnik, die vollständige ASAM-Standardreihe sowie die unterschiedlichen Automatisierungsschnittstellen erläutert. Anhand der erarbeiten Informationen zu diesen Schnittstellen wird im Anschluss die Einsatztauglichkeit beurteilt und damit der Ausgangspunkt für die weiteren Abschnitte der Arbeit geschaffen.

Im zweiten Teil werden die gesetzten Ziele der Arbeit, gemäß den erarbeiteten Themen, behandelt. Dazu wird die zuvor gewählte Schnittstelle in Betrieb genommen, darauffolgend eine LabVIEW-Clientanwendung erstellt und mittels dieser Anwendung verschiedene Funktionstests, die der Beurteilung des Interfaces zugrunde liegen, durchgeführt. Darüber hinaus werden die Eigenschaften der Bussysteme CAN FD, CAN und FlexRay verglichen und dementsprechend die Unterschiede wie auch die Einsatzmöglichkeiten dieser Protokolle abgeleitet. Daneben werden die benötigten Hardwarekomponenten für einen möglichen Einsatz von CAN FD im Prüfstandsbetrieb betrachtet und untersucht.

Schlussendlich konnte aufgrund der erarbeiteten Themen bzw. durchgeführten Funktionstests festgestellt werden, dass die automatisierte Bedatung automobiler Steuergeräte via LabVIEW anhand der von Vector spezifizierten COM-Schnittstelle möglich ist und, dass sich die Flexibilität der Prüfstandsanlage durch den Einsatz von CAN FD anstatt des herkömmlichen CAN-Protokolls wesentlich erhöht. Somit wurde das Ziel der Arbeit, ein Konzept, welches die Grundlage für die zukünftige Optimierung der Anlage bildet und den gesetzten Anforderungen entspricht, erreicht.

ABSTRACT

The present master thesis deals with the development of a concept to improve the operation and flexibility of an existing test rig system for powertrain components by means of an interface for the automated data supply of ECUs as well as the newly developed CAN FD bus system. This thesis investigates questions in what way the automated data supply of an ECU can be implemented and which changes and possibilities the development of CAN FD has induced concerning modern test rig technology.

In the first part of the thesis, the basics of digital bus technology, the essential bus systems in vehicle technology, the complete ASAM standard series as well as the different automation interfaces are explained. Based on this information the suitability for its use is assessed, providing a starting point for subsequent sections of the work.

In the second part, the objectives of the thesis are discussed according to the results obtained. For this purpose, the previously selected interface is put into operation, subsequently a LabVIEW client application is created and various function tests based on the assessment of the interface are carried out using the application. In addition, the characteristics of the CAN FD, CAN and FlexRay bus systems are compared and the differences as well as the application possibilities of these protocols are discussed. Additionally, the required hardware components for a possible use of CAN FD in test bench operation are evaluated and examined.

Based on thorough investigation it was possible to determine that the automated data supply of ECUs via LabVIEW is possible by using the Vector COM interface and that the flexibility of the test rig system is significantly increased by transforming the bus system from CAN to CAN FD. Thus, the aim of the work, a concept which forms the basis for the future optimization of the test bench system and fulfils the set requirements, was reached.

INHALTSVERZEICHNIS

1 EINLEITUNG

Diese Arbeit beschäftigt sich mit der Erstellung eines Konzepts zur Echtzeit-Kopplung automobiler Steuergeräte an ein Prüfstands-Automatisierungssystem. Im Zuge dessen wird untersucht, inwiefern die automatisierte Bedatung von Steuergeräten durchgeführt werden kann und welche Möglichkeiten sich durch das moderne CAN FD-Protokoll ergeben. Dazu werden die diesbezüglichen Grundlagen erarbeitet und anhand von Funktionstests sowie Gegenüberstellungen geprüft, ob die gewählte Schnittstelle sowie das CAN FD-Protokoll den gesetzten Anforderungen in Bezug auf Performance bzw. Flexibilität entsprechen und eine etwaige Implementierung in das vorhandene Prüfstandssystem in Frage kommt.

Im Folgenden werden die Ausgangssituation sowie das Unternehmen beschrieben. Nachdem es sich bei der vorliegenden Arbeit um eine Adaptierung eines vorhandenen Prüfstandssystems handelt, werden dazu zunächst dessen grundlegenden Daten erläutert. Im Anschluss daran werden der Ist- und Sollzustand der Prüfstandsanlage erläutert.

1.1 Ausgangssituation

Den Ausgangspunkt dieser Arbeit bildete ein Prüfstandssystem ohne Schnittstelle zur automatisierten Bedatung von Steuergeräten. Da aufgrund dessen bei bestimmten Funktionsmessungen manuelles Eingreifen erforderlich ist, kann kein vollständig automatisierter Betrieb der Prüfanlage gewährleistet werden. Daneben ergaben sich durch den Einsatz des gewöhnlichen CAN-Protokolls Einschränkungen in Bezug auf die Flexibilität des Prüfstandsystems. Diese Umstände sollen in dieser Arbeit behandelt werden, indem eine Schnittstelle ausgewählt wird, die eine automatisierte Bedatung von Steuergeräten erlaubt und der mögliche Einsatz des CAN FD-Bussystems eingeschätzt bzw. beurteilt wird.

1.1.1 Prüfstand

Die Prüfstandsanlage dient zur Validierung von Antriebsstrangkomponenten und besteht grundsätzlich aus drei Elektromotoren inklusive Frequenzumrichter, einem Prüfstands-Automatisierungs-PC mit I/O-System, einer Regelungseinheit, einem Applikations-System-PC mit Bus-Interface sowie einem Messsystem. Zusätzlich besteht die Möglichkeit verschiedene Prüfeinheiten bei Bedarf in den Prüfstand zu integrieren. Allerdings sind diese Komponenten, wie auch der sogenannte Schnittstellenrechner zur Datensichtung, für diese Arbeit nicht von Bedeutung und werden folglich nicht näher erwähnt.

Die Elektromotoren mit einer Leistung von 330kW (z.B. Eintrieb) bzw. 220kW (zwei Stück z.B. Abtrieb links und rechts) können dabei je nach Aufbauart und Prüfmethode als Antriebs- oder Abtriebsmaschinen genutzt werden. Demzufolge können entsprechend der Anordnung verschiedene Prüfszenarien durchgeführt und die Anlage als 1-, 2- oder 3-Maschinen Prüfstand genutzt werden. Aufgrund dieser Flexibilität im Aufbau ergeben sich eine Vielzahl an durchführbaren Prüfmöglichkeiten. Mit dem Einsatz von Übersetzungsgetrieben kann eine noch höhere Flexibilität erreicht werden.

In der folgenden Abbildung (Abb. 1) ist ein 2-Maschinen-Aufbau (2M-Aufbau) mit Verlagerungsgetrieben ersichtlich. Im linken unteren Vordergrund ist ein Prüfgetriebe inklusive dem dazugehörigen Steuergerät erkennbar.

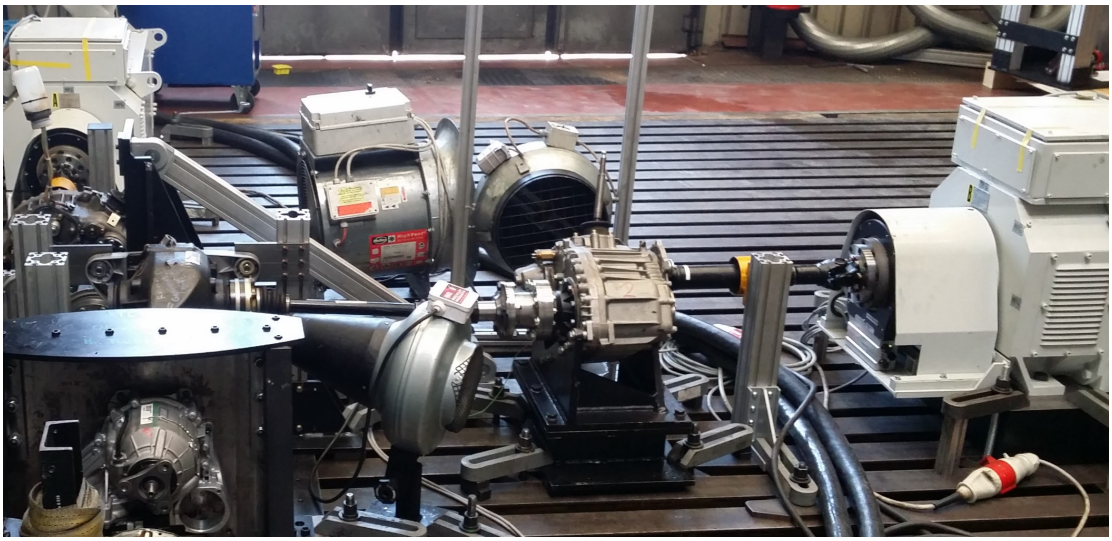


Abb. 1: Prüfstand 2M-Aufbau, Quelle: Eigene Darstellung.

Am Prüfstands-Automatisierungs-PC, auch als Vorgaberechner bezeichnet, werden die Prüfprogramme ausgeführt und damit der Prüfstandsbetrieb sichergestellt. Die Regelungseinheit ist mit den Frequenzumrichtern (kurz FU) der Elektromotoren verbunden, überwacht die Signale der FUs und regelt die Drehzahlen und Drehmomente gemäß der Vorgabe des Prüfstands-Automatisierungs-PCs über das I/O-System. Der Applikations-System-PC mit Bus-Interface wird unabhängig vom Vorgaberechner zur Bedatung von Steuergeräten genutzt. Das Messsystem dient der Datenverarbeitung, Überwachung und Aufzeichnung aller am Prüfstand anfallenden Messsignale (außer steuergerätinterne Daten). Durch die Messsignalüberwachung wird der Prüfstand im Fehlerfall abgestellt. Folglich können Schäden an den Prüfkomponenten oder am Aufbau vermieden werden. Die angeführten Systeme sind untereinander über Ethernet und CAN verbunden und können mit einander kommunizieren. Die detaillierte Beschreibung der Vernetzung und Kommunikation der einzelnen Geräte bzw. Einheiten wird im Praxisteil der Arbeit erläutert.

1.1.2 Istzustand

Die Bedatung der Steuergeräte wurde seit Bestehen der Prüfstandsanlage ohne automatisierte Schnittstelle durchgeführt. Darüber hinaus wurde der gewöhnliche CAN-Bus anstatt des modernen CAN FD-Bussystems als primäre Kommunikationstechnik verwendet. Durch das erforderliche manuelle Eingreifen im Prüfstandsbetrieb, aufgrund der fehlenden Automatisierungsschnittstelle, kann kein vollständig automatisierter Testablauf durchgeführt werden. Des Weiteren begrenzt das herkömmliche CAN-Protokoll die Flexibilität der Prüfstandsanlage. Folglich entstehen hohe Zeitaufwände (z.B. 1 - 5 Minuten für die Parametrierung eines Steuergeräts bevor eine Funktionsmessung aufgenommen werden kann, die mehrmals an einem Tag durchgeführt werden muss) und eine verhältnismäßig schlechte Auslastung der Prüfanlage, da kein 24h-Betrieb möglich ist. Aus diesem Grund ergibt sich das Untersuchungsinteresse die Zeitaufwände zu minimieren und gleichzeitig die Auslastung der Prüfanlage zu steigern. Die Problemstellung dabei liegt darin, dass im Umfang dieser Arbeit geprüft werden muss, ob die unterschiedlichen Automatisierungsschnittstellen sowie der moderne CAN FD-Bus überhaupt für den geforderten Prüfstandsbetrieb ausgelegt sind und eingesetzt werden können.

1.1.3 Sollzustand

Das Ergebnis dieser Masterarbeit soll ein Konzept zur Echtzeit-Kopplung automobiler Steuergeräte an das vorhandene Prüfstands-Automatisierungssystem sein, das als Basis für die zukünftige Optimierung der Anlage dient, um Testläufe flexibler und effizienter durchführen zu können. Dementsprechend ist eine Schnittstelle zwischen Vorgaberechner und Applikations-System-PC auszuwählen, welche die automatisierte Bedatung von automobilen Steuergeräten im Prüfstandsbetrieb mit ausreichender Performance und Flexibilität ermöglicht. Des Weiteren soll aus dieser Arbeit hervorgehen, welche Möglichkeiten sich durch einen Umstieg von CAN auf CAN FD ergeben und inwiefern dieses Bussystem für den Einsatz am Prüfstand geeignet ist.

Die anschließende Darstellung zeigt den Ist- bzw. Sollzustand der Prüfstandsanlage in vereinfachter Weise. Dazu sind die erwähnten Komponenten wie Prüfstands-Automatisierungs-PC (LabVIEW, abgebildet ohne I/O-System), Regelungseinheit, Applikations-System-PC (CANape) mit Bus-Interface (Vector) und Messsystem ersichtlich. Die Elektromaschinen inklusive der übrigen Prüfstandskomponenten wurden als Prüfstandsanlage zusammengefasst. Die Automatisierungsschnittstelle, die im Zuge dieser Arbeit betrachtet werden soll, ist in Rot (strichliert) gehalten. Mit einer derartigen Schnittstelle soll es möglich sein über den Vorgaberechner CANape fernzusteuern und somit eine automatisierte Bedatung von Steuergeräten sicherzustellen.

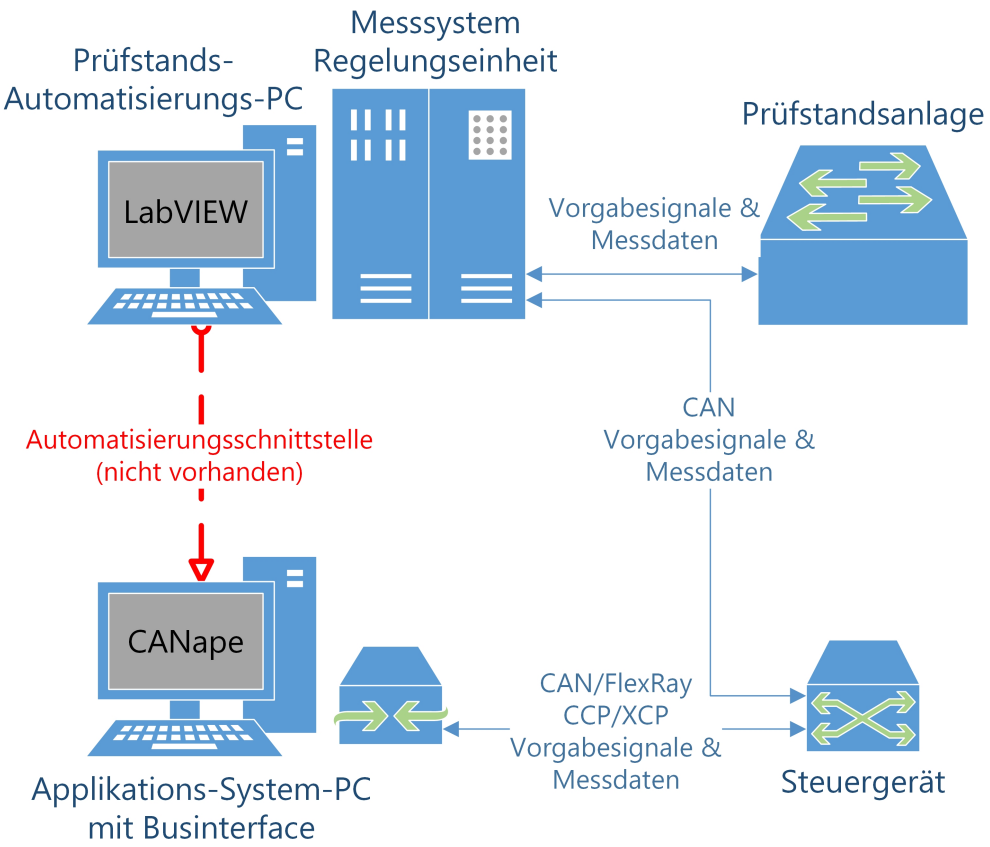


Abb. 2: Prüfstandsanlage vereinfacht, Quelle: Eigene Darstellung.

1.2 Unternehmen

Diese Arbeit wurde bei der Magna Steyr Engineering AG & Co KG am Standort Graz in der Abteilung Validation Chassis & Powertrain durchgeführt. Das Unternehmen ist im weltweit agierenden Magna Konzern eingegliedert, ist in der Automobilbranche als Forschungs- und Entwicklungsunternehmen tätig, beschäftigt über 950 Mitarbeiter und ist in mehrere Fachbereiche unterteilt. Graz ist der größte Magna Engineering Standort weltweit.[1]

Die anschließende Tabelle dient dem Überblick zu den wesentlichen Daten des genannten Unternehmens.

Unternehmen	Magna Steyr Engineering Austria
Sitz	Graz, Liebenauer Hauptstraße 317
Rechtsform	Kommanditgesellschaft (AG & Co KG)
Leitung	Martin Peter, General Manager
Mitarbeiter	960, Stand: April 2016
Tätigkeitsfeld	F&E in der Automobilbranche
Kompetenzen	Styling, Rohbau Exterior/Interior, Fahrwerk/Antriebsstrang, Elektrik/Elektronik, Prototypenbau, Gesamtfahrzeug, Integrierte Validierung, Hybrid/E-Fahrzeug, Innovationen, Technologien, Projekt Management

Tab. 1: Übersicht MSE, Quelle: Magna Steyr Engineering (2016), Online-Quelle [9.Mai.2016].

In der nachkommenden Abbildung ist das Logo des Magna Konzerns ersichtlich. Dieses wird weltweit für alle Sparten des Automobilzulieferers verwendet.

Abb. 3: Magna Logo, Quelle: Magna International Inc. (2016), Online-Quelle [9.Mai.2016].

Die Abteilung Validation Chassis & Powertrain ist in vier Gruppen unterteilt und beschäftigt 38 Mitarbeiter (Stand: Mai 2016). Die Tätigkeitsfelder der Abteilung liegen im Absicherungsprozess der Module/Systeme des Fachbereichs Fahrwerk und Antrieb vom Konzept über die virtuelle Simulation und physische Erprobung bis hin zur Freigabeempfehlung und Serienbetreuung.

[1] Vgl. Magna Steyr Engineering (2016), Online-Quelle [9.Mai.2016].

Des Weiteren beschäftigt man sich mit dem Entwerfen von Design Validation Plans (DVP) und der Durchführung von Teststrecken Analysen mit anschließendem Ausarbeiten von Prüfprogrammen für physische Erprobungen am Hydraulikprüffeld.[2]

Die folgende Abbildung zeigt das Entwicklungsgebäude in Graz. Dieses stellt das Magna Steyr Engineering-Hauptgebäude dar.

Abb. 4: Hauptgebäude MSE, Quelle: Magna Steyr Engineering (2016), Online-Quelle [9.Mai.2016].

[2] Vgl. Magna Steyr Engineering (2016), Online-Quelle [9.Mai.2016].

2 BUSTECHNIK IN DER FAHRZEUGTECHNIK

Digitale Bussysteme gelten als wesentlicher Bestandteil in Fahrzeugsystemen. Die Aufgabe dieser Systeme besteht darin, die stetig steigende funktionale Ausrichtung in der Fahrzeugtechnik durch Bereitstellung des entsprechenden Kommunikationssystems zu gewährleisten.[3]

Dieses Kapitel soll einen Überblick über die Bustechnik in Fahrzeugsystemen geben. Dazu werden die Grundlagen der digitalen Bustechnik und die Kfz-typischen Bussysteme erläutert. Anschließend wird das Kapitel mit den Steuergerätesystemen, die seit Jahren wesentlich in der Automobilbranche sind, und den dazugehörigen Diagnosetechniken abgeschlossen. Diese Informationen stellen den Ausgangspunkt zur Durchführung dieser Arbeit dar.

2.1 Grundlagen der digitalen Bustechnik

Der folgende Abschnitt dient zur Darstellung der zentralen Grundlagen digitaler Bussysteme und soll die notwendigen Ausgangsinformationen zur Abarbeitung dieser Arbeit liefern. Zu Beginn werden die Grundbegriffe der Bustechnik beschrieben. Darauf folgen das ISO/OSI-Schichtenmodell, die unterschiedlichen Netzwerktopologien, die Organisation und Adressierung von Busteilnehmern, die Kommunikationsprinzipien in einem Bussystem, die verschiedenen Übertragungsarten, der Einsatz von Protokollen, die unterschiedlichen Buszugriffsverfahren, der grundlegende Botschaftsaufbau und die Kopplung von Bussystemen.

2.1.1 Grundbegriffe

Im Folgenden werden die Grundbegriffe der Bustechnik erarbeitet. Diese dienen als Basis für die weiteren Abschnitte und Kapitel. Auf die erläuterten Begriffe und Auslegungen wird im Laufe dieser Arbeit mehrmals zurückgegriffen.

Der Begriff Bussystem beschreibt eine spezielle Art von Kommunikationsnetzwerken. Speziell deshalb, da von Bussystemen eine hohe Zuverlässigkeit, zeitliche Präzision der Datenübermittlung und geringe Leitungs- und Knotenkosten gefordert werden. Zusätzlich dürfen elektromagnetische Störquellen auch in direkter Umgebung der Busleitung keinen Einfluss haben.[4]

Eine Definition, die bei Bussystemen häufig verwendet wird, ist das Protokoll. Unter diesem Begriff versteht man ein eindeutiges und komplettes Reglement für Kommunikationen von zumindest zwei Teilnehmern. Die Regelungen in einem Protokoll beziehen sich auf Zeitvorgaben, Semantik, Pragmatik und Syntax. Mit der Syntax eines Protokolls werden die erlaubten Zeichen- oder Wortfolgen der Kommunikation bestimmt. Die Semantik hingegen beschreibt die entsprechende Verwendung syntaktisch korrekter Objekte.[5]

[3] Vgl. Reif (2014), S. 1.

[4] Vgl. Paulweber/Lebert (2014), S. 253.

[5] Vgl. Reif (2014), S. 2.

Ein weiterer wesentlicher Begriff bei Bussystemen ist die Latenzzeit. Dieses Merkmal dient zur Bestimmung der Performance.[6] Die Latenzzeit, auch als Verzögerungszeit, Reaktionszeit oder Verweilzeit bezeichnet, beschreibt den Zeitrahmen zwischen einem Ereignis und dem Eintritt der daraus resultierenden Reaktion.[7]

Der nächste wichtige Parameter ist die Übertragungsrate. Diese gibt die Anzahl der Bytes an, die pro angeführter Zeiteinheit übermittelt werden. Vorrangig wird die Übertragungsrate in kByte/s oder MByte/s angegeben. Dabei wird oftmals die Bruttoübertragungsrate genannt, welche die gesamten Botschaftsinhalte beinhaltet. Diese bestehen aus Nutz- und Zusatzdaten. Um ein Bussystem besser beurteilen zu können, wird daher die Nettoübertragungsrate, auch Nutzdatenrate genannt, benötigt. Diese bezieht sich lediglich auf die Nutzdaten und kann demzufolge Aufschluss über die Einsatzmöglichkeiten geben.[8]

Ein weiterer Parameter der Auskunft über die Performance eines Protokolls geben kann, ist die Dateneffizienz. Bei Nachrichten mit nur wenigen Bits ist die Dateneffizienz wesentlich. Diese erhält man indem man die Nutzdaten zu den Bruttodaten in ein Verhältnis setzt. Ist die Dateneffizienz hoch kann eine bestmögliche Ausnutzung der verfügbaren Busbandbreite erreicht werden.[9]

In der folgenden Abbildung ist ein Beispiel für die Dateneffizienz drei bekannter Kommunikationssysteme bei der Übertragung von 8 Bit Nutzdaten (ND) unter den angeführten Bedingungen dargestellt.

Datenbus	**Dateneffizienz für ND = 8 Bit**	**Anmerkung**
Basic CAN	15 %	11 Bit Identifier; Stuffing Bits vernachlässigt; inkl. EOF
Profibus-DP	7 %	SD2 Telegramm, 11 Bit pro Zeichen
TCP/IP Ethernet	1 %	

Abb. 5: Dateneffizienz, Quelle: Paulweber/Lebert (2014), S. 255.

Zum Abschluss der Grundbegriffe werden die drei unterschiedlichen Kommunikationsformen, welche in einem Bussystem vorkommen können, erläutert. Dabei wird entsprechend der beteiligten Teilnehmer unterschieden. Von einer Punkt-zu-Punkt-Verbindung, auch Unicast-Verbindung genannt, spricht man wenn lediglich zwei Teilnehmer, also ein Sender und ein Empfänger, vorhanden sind. Bei einer Broadcast-Verbindung übermittelt ein Sender eine Botschaft an eine Vielzahl von Teilnehmern. Die Botschaft wird dabei nicht notwendigerweise von allen Teilnehmern verarbeitet. Bei der Multicast-Verbindung wird ebenfalls eine Botschaft an mehrere Teilnehmer gesendet, jedoch wird diese auch von allen Empfängern verwertet.[10]

[6] Vgl. Reif (2014), S. 2.

[7] Vgl. Streichert/Traub (2012), S. 149.

[8] Vgl. Reif (2014), S. 2.

[9] Vgl. Paulweber/Lebert (2014), S. 255.

[10] Vgl. Reif (2014), S. 3.

Ein weiterer Unterschied zwischen einer Broadcast- und einer Unicast-Verbindung liegt in der Bestätigung des Empfängers. Bei der Broadcast-Kommunikation erhält der entsprechende Sender für gewöhnlich keine Bestätigung des Empfängers und wird aus diesem Grund auch als Unacknowledged Communication bezeichnet. Bei einer Unicast-Verbindung, auch Acknowledged Communication genannt, hingegen, erwartet der Sender häufig eine positive Bestätigung, sofern die Nachricht fehlerfrei empfangen wurde. Wird auf diese Bestätigung verzichtet, wird jedenfalls eine Fehlermeldung, Not Acknowledged, erwartet, falls ein Übermittlungsfehler vom Empfänger detektiert wurde. Daraufhin übermittelt der Sender die Botschaft im Normalfall erneut.[11]

In der folgenden Darstellung sind die beschriebenen Kommunikationsformen und deren Erläuterung angeführt.

Bezeichnung	Erläuterung
Punkt-zu-Punkt-Verbindung	Ein Sender, ein Empfänger
Broadcast-Verbindung	Ein Sender, potentiell viele Empfänger
Multicast-Verbindung	Ein Sender, tatsächlich viele Empfänger

Abb. 6: Kommunikationsformen, Quelle: Reif (2014), S. 3.

2.1.2 ISO/OSI-Referenzmodell

Das ISO/OSI-Referenzmodell, entwickelt von der ISO (International Organization for Standardization), dient als Basis zur Definition von unterschiedlichen Kommunikationsprotokollen und deren Vergleich. Dabei werden Datenkommunikationssysteme in mehreren Schichten, auch als Ebenen oder Layer bezeichnet, angeführt. Folglich können die Teile der Datenkommunikation in Funktionsbereichen übersichtlich und in vereinfachter Form dargestellt werden. Für bestimmte Anwendungen (z.B. Kfz-Bussysteme wie CAN) ist es möglich unterschiedliche Ebenen zur einfacheren Darstellung zusammenzufassen.[12]

Um die wesentlichen Eigenschaften des ISO/OSI-Modells zu verdeutlichen, ist in der nachkommenden Abbildung die Kommunikation der Systeme A und B in einem Bussystem dargestellt. Dabei wird die Informationsverarbeitung anhand der sieben aufeinander aufbauenden Schichten durchgeführt. Die unterschiedlichen Schichten übernehmen dabei spezifische Aufgaben. Der darüber liegenden Ebene werden jene Dienste, welche die entsprechende Schicht enthält, zur Verfügung gestellt. Um auf waagrechter Ebene mit der jeweils selben Schicht kommunizieren zu können, werden entsprechende Protokolle, wie in der folgenden Abbildung zu sehen, eingesetzt.[13]

[11] Vgl. Zimmermann/Schmidgall (2014), S. 22.

[12] Vgl. Reif (Hrsg.) (2011), S. 76.

[13] Vgl. Reif (2014), S. 3f.

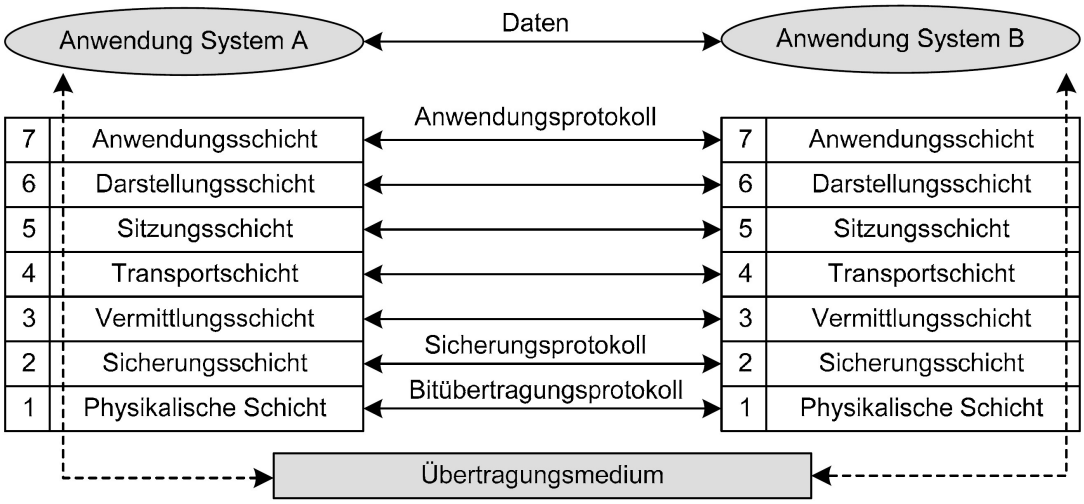


Abb. 7: ISO/OSI-Referenzmodell, Quelle: Reif (2014), S. 3.

Die physikalische Schicht umfasst die Bitübertragung und hat die Bereiche elektrische Signalpegel, Bitcodierung und die Spezifizierung von Hardware, wie Stecker und Kabel, als Aufgabe. Schicht 2 ist für Flusskontrolle, Fehlersicherung, Buszugriff und Botschaftsaufbau zuständig. Die Vermittlungsschicht dient der Adressvergabe, dem Routing und der Teilnehmerüberwachung bzw. -erkennung. Für den Datentransport ist Schicht 4 zuständig. Ihre Aufgaben liegen in der Aufteilung und Zusammensetzung der Datensätze mehrerer Nachrichten, auch Segmentierung genannt. In den Schichten 5 und 6 wird die Sitzungssteuerung und Darstellung abgehandelt. Schicht 7 ist für Anwendungen und Dienste zuständig.[14]

Bei Feldbussystemen werden üblicherweise nur die Ebenen eins (physical layer), zwei (data link layer) und vereinzelt sieben (application layer) integriert.[15] Im Kfz-Bereich wird dementsprechend auf eine vereinfachte Darstellung mit diesen drei Schichten zurückgegriffen.[16]

Die physikalische Schicht (Ebene 1, physical layer) umfasst die Festlegung prozeduraler und elektrischer Parameter der physikalischen Verbindung unter den Teilnehmern in einem Netzwerk.[17] Für diese Ebene ist eine weitere Unterteilung in drei Teilschichten vorhanden. Diese Teilschichten werden als MDI, PMA und PLS bezeichnet. MDI steht dabei für Medium Dependant Interface, PMA für Physical Medium Attachment und PLS für Physical Signaling.[18] In den Umfang der physikalischen Ebene fällt die Definition mechanischer und funktionaler Parameter, wie zum Beispiel der Typ und die Art von Kabeln.[19] Dies wird von der ersten Teilschicht, der medienabhängigen Schnittstelle MDI, abgedeckt. Ein weiteres Beispiel für diesen Teil der physikalischen Ebene ist die Spezifikation eines Steckverbinders zur Busankopplung.[20]

[14] Vgl. Zimmermann/Schmidgall (2014), S. 6.

[15] Vgl. Paulweber/Lebert (2014), S. 254.

[16] Vgl. Reif (2014), S. 4.

[17] Vgl. Reif (Hrsg.) (2011), S. 76.

[18] Vgl. Borgeest (2014), S. 93.

[19] Vgl. Reif (2014), S. 4.

[20] Vgl. Borgeest (2014), S. 93.

Darüber hinaus hat die Ebene 1 die Festlegung des Signalpegels als Aufgabe. Nachdem Daten in der Digitaltechnik durch die Aufeinanderfolge der binären Zustände 0 und 1 übermittelt werden, muss festgesetzt werden wie diese beiden Zustände auf dem eingesetzten Übertragungsmedium (z.B. Zweidrahtleitung) dargestellt werden. Dabei ist es wesentlich, dass es auf der Busleitung zu keinem Kurzschluss kommt, falls ein Busteilnehmer den Zustand 0 und ein anderer den Zustand 1 übermittelt. Die Darstellung dieser Binärzustände kann auf unterschiedliche Art und Weise erfolgen (z.B. CAN-B 0 und 5 Volt). Sollte es die entsprechende Kodierung vorsehen, dass ein Pegel einen anderen überschreiben kann, spricht man beim Überschreibenden von dominant und beim Nachgebenden von rezessiv.[21] Diese Umfänge, sprich die elektrische Aufbereitung von ausgehenden und ankommenden Signalen, fallen in den Aufgabebereich des physikalischen Medienzugangs PMA.[22] Nachdem alle Funktionen für Betrieb und Synchronisation der Verbindung und Kommunikation in der physikalischen Schicht festgelegt werden, zählt auch die Bestimmung des Bitcodierungsverfahrens hinzu. Zwei bekannte Verfahren sind die Non-Return-to-Zero-Codierung, auch NRZ-Codierung genannt, und die Manchester-Codierung. Diese Verfahren unterscheiden sich in der Anzahl der Zeitblöcke, welche für die Darstellung eines Bits genutzt werden. Bei der NRZ-Codierung kann der Fall eintreten, dass der gleiche Signalpegel über mehrere Zeitblöcke aufrecht bleibt. Um die Synchronisation sicherstellen zu können, sind daher Zusatzmaßnahmen erforderlich.[23] Eine Maßnahme nennt sich Bit-Stuffing. Bei diesem Verfahren wird nach einer definierten Anzahl von Bits einer Botschaft, bei CAN beispielsweise 5 Bit, stets dann ein so genanntes Stuffbit eingefügt, wenn zu viele Bits mit dem gleichen Zustand aufeinanderfolgen. Ansonsten wäre im Signal keine Signalflanke für die Nachsynchronisation des Bittaktgenerators verfügbar.[24] Diese genannten Aufgaben, wie die Darstellung wann ein Bit gesetzt oder entfernt wird sowie die Überprüfung ob das gesendete Signal auch empfangen wurde, werden in der Teilschicht PLS bearbeitet.[25] Die nächsthöhere Schicht im Referenzmodell ist die Sicherungsschicht und ist für die Datenübermittlung zwischen zwei Teilnehmern, welche sich im selben Netz befinden zuständig. Geregelt wird dies durch den Einsatz eines Zugriffsverfahren und die Steuerung des Datenflusses. Außerdem sichert diese Ebene die Kommunikation durch Korrektur- und Fehlererkennungsverfahren.[26] Aufgeteilt wird die Sicherungsschicht in die Teile MAC und LLC. MAC ist die Abkürzung für Medium Access Control (Medienzugriffssteuerung). Die Medienzugriffsteuerung bestimmt welcher Teilnehmer zur welchen Zeit auf das Bussystem zugreifen darf. Zusätzlich definiert die MAC die Bitfolge der zu übertragenden Botschaften und die vom eingesetzten Protokoll übermittelten Zusatzdaten, auch Hilfsdaten genannt. LLC steht für Logic Link Control und beschreibt die logische Verbindungsabsicherung. Diese sorgt dafür, dass die Teilnehmer im System die für sie bestimmten Daten erhalten.[27]

[21] Vgl. Reif (Hrsg.) (2011), S. 76f.

[22] Vgl. Borgeest (2014), S. 93.

[23] Vgl. Reif (2014), S. 4.

[24] Vgl. Zimmermann/Schmidgall (2014), S. 27.

[25] Vgl. Borgeest (2014), S. 93.

[26] Vgl. Reif (2014), S. 5.

[27] Vgl. Borgeest (2014), S. 94.

Als oberste Ebene des ISO/OSI-Schichtenmodells ist die Anwenderschicht angeführt. Diese besteht aus der Applikation, welche entsprechende Informationen bereitstellt und verarbeitet. Die Ebene 7 ist folglich die einzige Protokollebene des Referenzmodells, an welche Sensor- und Anwendereingaben, wie beispielsweise Anforderungen bzw. Vorgaben, übermittelt werden können.[28]

In der nachkommenden Darstellung ist ein für Kfz-Bussysteme abgewandeltes OSI-Modell abgebildet. Auf diesem sind die für automobile Bussysteme relevanten und vorhin näher erläuterten Ebenen und deren Teilschichten hervorgehoben. Die weniger bedeutenden Schichten sind lediglich angedeutet (z.B. Schicht 3).

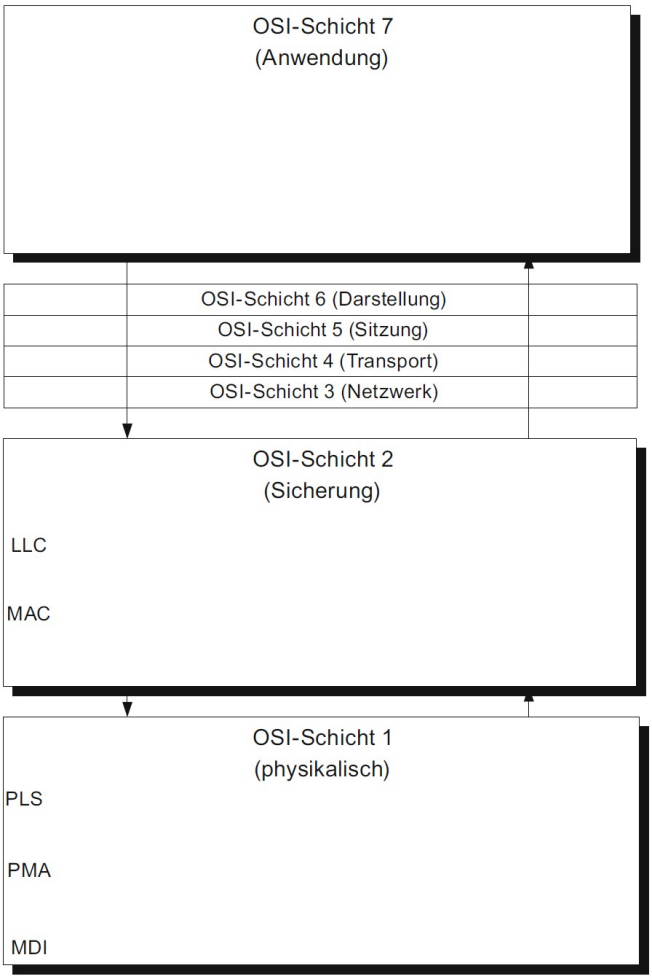


Abb. 8: OSI-Modell für automobile Bussysteme, Quelle: Borgeest (2014), S. 93.

Die gängigen Kfz-Bussysteme wie CAN, LIN und FlexRay legen in der Regel nur Schicht 1 und 2 fest. Meist wird selbst davon nur ein Ausschnitt eindeutig spezifiziert. Für die höheren Ebenen sind erst seit kurzer Zeit Standardisierungen verfügbar.[29]

[28] Vgl. Reif (Hrsg.) (2011), S. 78.

[29] Vgl. Zimmermann/Schmidgall (2014), S. 7.

2.1.3 Netzwerktopologie

Die Topologie eines Netzwerks (z.B. Bussystem in einem Fahrzeug) stellt die Art und Weise dar, wie die Verknüpfung zwischen den unterschiedlichen Teilnehmern aufgebaut ist. Für gewöhnlich bestehen Beziehungen oder Zwangsverknüpfungen zwischen dem gewählten Protokoll, den Realisierungsaufwendungen und der entsprechenden Topologie.[30]

Eine Netzwerktopologie gibt demnach einen Überblick über die Struktur und die Verbindungen innerhalb eines Systems. Genauere Details, wie die Kabellänge zwischen den einzelnen Teilnehmern, werden nicht angeführt. Entsprechend der Anwendung gibt es unterschiedliche Anforderungen an die verwendete Topologie, die diverse Eigenschaften des Gesamtnetzwerks festlegt. Alle Topologietypen setzen auf den vier Grundtopologien Sterntopologie, Bustopologie, Ringtopologie und Maschentopologie auf. Darüber hinaus gibt es die Möglichkeit Hybridtopologien einzusetzen. Dabei werden durch Kombination der genannten Grundtopologien weitere Strukturen möglich.[31]

Die wesentliche Eigenschaft der Sterntopologie ist der zentrale Knoten, welcher in der Regel Sternkoppler genannt wird. Die übrigen Teilnehmer des Bussystems sind über eine Punkt-zu-Punkt-Verbindung mit diesem Sternkoppler verknüpft. Der Vorteil dieser Topologie liegt in der darstellbaren Übertragungsrate durch die exklusive Ankopplung der Teilnehmer. Als Nachteil sind die erhöhte Komplexität in Bezug auf die Verbindungstechnik sowie das Systemverhalten bei einem möglichen Ausfall des Sternkopplers anzuführen.[32] Für die Sterntopologie gilt demnach: Sollte ein Netzwerkteilnehmer oder eine Verbindung zum zentralen Teilnehmer ausfallen, bleibt das Netzwerk für die übrigen Teilnehmer in Betrieb. Fällt indes der Sternkoppler aus, ist das gesamte System nicht mehr funktionsfähig. Die Kommunikation unter den Netzwerkteilnehmer ist ab dem Ausfall somit nicht mehr möglich.[33]

Bei der Bustopologie, auch Linientopologie genannt, werden alle Netzwerkteilnehmer durch ein gemeinsam genutztes Übertragungsmedium, die Busleitung, verknüpft. Die sich darauf befindlichen Daten sind für alle Busteilnehmer verfügbar. Fällt ein aktiver Teilnehmer aus, entstehen bei dieser Topologie keine gravierenden Auswirkungen in Bezug auf das gesamte Systemverhalten. Demgegenüber sind bei der Bustopologie Mechanismen zur Kollisionsvermeidung oder -erkennung erforderlich, da prinzipiell mehrere Busteilnehmer zum selben Zeitpunkt senden können.[34] Ein großer Nachteil dieser Topologie liegt darin, dass bei einem möglichen Defekt der Hauptbusleitung (der Ausfall einer Stichleitung wirkt sich nur auf die darauf angeschlossenen Teilnehmer aus) das komplette Bussystem ausfällt. Dies könnte beispielsweise durch einen Kabelbruch geschehen.[35]

---

[30] Vgl. Reif (2014), S. 7.

[31] Vgl. Reif (Hrsg.) (2011), S. 71.

[32] Vgl. Reif (2014), S. 7.

[33] Vgl. Reif (Hrsg.) (2011), S. 72.

[34] Vgl. Reif (2014), S. 7.

[35] Vgl. Reif (Hrsg.) (2011), S. 71.

Das wesentliche Merkmal der Ringtopologie ist die geschlossene Verknüpfung von gerichteten Punkt-zu-Punkt-Verbindungen. Nachdem der Ausfall eines Netzwerkteilnehmers die Unterbrechung des Ringes nach sich zieht, sind in der Regel zusätzliche Maßnahmen zur Steigerung der Systemsicherheit notwendig.[36] Dazu können Ringe in der Form eines Doppelringes organisiert werden. In diesem erfolgt die Datenübertragung in beide Richtungen. Aus diesem Grund hat der Ausfall eines Teilnehmers oder einer Verbindung keine direkte Auswirkung auf das übrige Netzwerk, da weiterhin alle Datenpakete an die übrigen fehlerfreien Teilnehmer des Rings gesendet werden. Fällt jedoch eine weitere Station oder Verbindung aus, können etwaige Fehlfunktionen nicht weiter ausgeschlossen werden. Bei der Maschentopologie sind alle Teilnehmer mit einem oder mehreren weiteren verknüpft. Ist das Netzwerk vollständig vermascht, sind alle Teilnehmer miteinander verknüpft. Sollte es zu einem Ausfall eines Teilnehmers oder einer Verbindung kommen, werden die entsprechenden Daten umgeleitet. Aus diesem Grund ermöglicht dieser Topologietyp eine hohe Ausfallsicherheit. Demgegenüber ist der Aufwand für den Datentransport und die Verkabelung wesentlich höher als bei den übrigen Topologien. Demnach besitzt die Maschentopologie das Merkmal einer Bustopologie in Bezug auf den Austausch der gesendeten Nachrichten und das einer Sterntopologie in Hinsicht auf die Datenübertragung untereinander, da alle Teilnehmer die gesamten Daten der übrigen Busteilnehmer erhalten und der Ausfall von Verbindungen keine Auswirkung auf den Datenverkehr hat.[37]

2.1.4 Netzwerkorganisation und Adressierung

Um Botschaften innerhalb eines Netzwerks übermitteln und deren Inhalte verarbeiten zu können, beinhalten diese zusätzlich zu den Nutzdaten, auch Payload genannt, Informationen, welche der Datenübermittlung dienen. Diese Informationen können entweder eindeutig in der Übermittlung enthalten oder indirekt durch entsprechende Vorgaben bestimmt sein. Damit eine Botschaft vom richtigen Teilnehmer empfangen wird, ist eine Adressierung erforderlich. Die Adressierung ist eine wesentliche Information für die Datenübertragung. Diesbezüglich gibt es unterschiedliche Prinzipien.[38] Die zwei gängigen Verfahren für die Adressierung von Botschaften in der Bustechnik sind die gerätebasierte Adressierung (Teilnehmer- oder Stationsadressierung) und die inhaltsbasierte Adressierung (Identifier, Botschaftskennung). Diese Mechanismen dienen der gezielten Identifizierung von einzelnen (Unicast) oder mehreren Empfängern (Multicast).[39]

Bei der gerätebasierten Adressierung wird jeder Teilnehmer anhand einer einmaligen Adresse (Nummer) erkannt. Dabei beinhaltet jede Nachricht die Zieladresse, die Adresse des Empfängers, und für gewöhnlich auch die Quelladresse, die Adresse des Senders. Für die Übermittlung von Nachrichten an mehrere (Multicast) oder die gesamten Teilnehmer eines Bussystems (Broadcast) sind gesonderte Adressen, auch funktionale Adressen genannt, vorhanden. Sind in einem Bussystem nur wenige Teilnehmer vorhanden, genügen einige Bit für die Adressierung. Werden hingegen mehrere Bussysteme

[36] Vgl. Reif (2014), S. 7.

[37] Vgl. Reif (Hrsg.) (2011), S. 72f.

[38] Vgl. Reif (Hrsg.) (2011), S. 74.

[39] Vgl. Zimmermann/Schmidgall (2014), S. 21.

über ein Gateway verknüpft, wird in den oberen Protokollebenen eine eindeutige Adresse verwendet, die für das gesamte System gilt. Diese Geräteadresse wird vom Gateway in einer unteren Protokollebene in eine ausschließlich für das adressierte Bussystem explizite Adresse umgelegt.[40] Gateways können dabei ankommende Datenpakete zerlegen und neu zusammenfügen bevor diese weitergeleitet werden.[41]

Die nachkommende Abbildung zeigt die Unterschiede zwischen Broadcast-, Multicast- und Unicast-Nachrichten anhand der Kommunikation unter Steuergeräten.

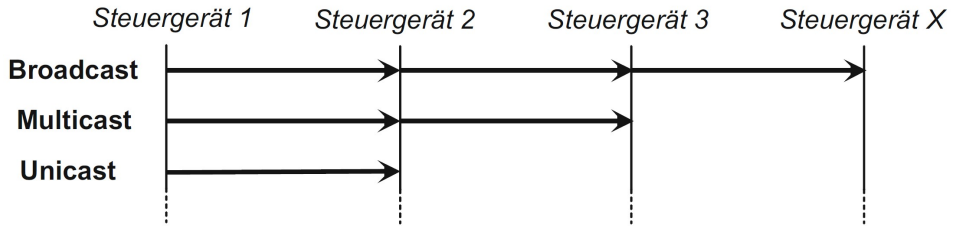


Abb. 9: Nachrichtenformen, Quelle: Zimmermann/Schmidgall (2014), S. 21.

Neben der gerätebasierten- kann auch eine inhaltsbasierte Adressierung eingesetzt werden. Diese wird beispielsweise bei CAN genutzt. Dabei wird einer Nachricht eine spezielle, adressartige Kennung, auch Identifier genannt, angefügt, welche den Nachrichteninhalt kennzeichnet. Damit ist ersichtlich von welchem Sender diese Nachricht stammt. Befinden sich in einem Netzwerk viele Teilnehmer, sind für die Adressierung längere Adressen erforderlich. Durch die inhaltsbasierte Adressierung kommt es zu einer geringeren Busbelastung. Zusätzlich führt diese Adressierungsvariante zu einer höheren Datenkonsistenz im gesamten Bussystem, wenn Daten regelmäßig an mehrere Busteilnehmer gesendet werden. Bei beiden Varianten ist jede Nachricht für alle Teilnehmer im Bussystem sichtbar. Auf Basis der Adresse müssen die Teilnehmer entscheiden, ob sie die Nachricht erhalten und verarbeiten oder ignorieren (Akzeptanzfilterung). Diese Akzeptanzprüfung wird bei Bussystemen mit niedriger Bitrate für gewöhnlich in der Protokollsoftware durchgeführt. Im Gegensatz dazu wird bei höheren Bitraten die Akzeptanzfilterung im Zuge der Protokollverarbeitung für die Ebenen 1 und 2 im Kommunikationscontroller durchgeführt. Der Kommunikationscontroller ist ein spezieller IC oder ein auf einem Mikrocontroller integriertes Schaltungsteil, auch als Media Access Controller (MAC) bezeichnet. Der Aufwand der Akzeptanzfilterung ist bei der gerätebasierten im Vergleich zur inhaltsbasierten Adressierung einfacher, da lediglich die eigene Zieladresse und eventuell die Broadcast- und Multicast-Adressen detektiert werden müssen. Demgegenüber müssen bei der inhaltsbasierten Adressierung oftmals Hunderte Adressen, auch Botschaftskennungen genannt, erfasst werden. Herkömmliche Kommunikationscontroller können in der Regel so eingestellt werden, dass sie die Adresse von maximal 16 Nachrichten hardwareseitig detektieren. Dabei können mittels Bitmasken komplette Bereiche an Adressen bestimmt werden. Für sonstige Botschaftskennungen, die ebenso erhalten werden sollen, ist eine Multicast-Kennung erforderlich, für welche die entsprechende Akzeptanzfilterung auf Softwarebasis durchgeführt wird.[42]

[40] Vgl. Zimmermann/Schmidgall (2014), S. 21f.

[41] Vgl. Streichert/Traub (2012), S. 21.

[42] Vgl. Zimmermann/Schmidgall (2014), S. 22.

2.1.5 Kommunikation

Für den Vorgang der Kommunikation zwischen Teilnehmern in einem Bussystem gibt es zwei gängige Grundformen, das Client-Server-Modell und das Producer-Consumer-Modell. Die diesbezügliche Darstellung dieser beiden Kommunikationsmodelle in Sequenzdiagrammen, in welchen die senkrechten Linien Teilnehmer oder die Ebenen im Referenzmodell darstellen und die Datenübertragung anhand der schrägen Striche angedeutet wird, ist von Vorteil. Das Client-Server-Modell definiert dabei eine Punkt-zu-Punkt-Verbindung, welche sich aus den vier Abschnitten Anzeige (Indication), Anforderung (Request), Bestätigung (Confirmation) und Antwort (Response) zusammensetzt. Das Producer-Consumer-Modell hingegen definiert die Vorgänge einer Kommunikation zwischen Busteilnehmern nach den Multicast- bzw. Broadcastprinzipien.[43]

Bei einer Broadcast-Kommunikation übermittelt der Sender unabhängig von einer Anfrage Botschaften. Dies ist mit der Ausstrahlung von Fernsehprogrammen vergleichbar. Demgegenüber werden beim Client-Server-Modell mittels einer Anfrage die entsprechenden Informationen angefordert. Beim Multicast-Verfahren wiederum wird der Sender Antworten von mehreren Empfängern erhalten. Werden dieselben Daten eines Senders wiederholt benötigt, kann das Bussystem durch den Einsatz des Publisher-Subscriber-Prinzips entlastet werden. Dazu muss der entsprechende Busteilnehmer sein Interesse an den jeweiligen Daten einmalig anmelden. Dieser Vorgang wird ist auch unter der Bezeichnung Subscribe bekannt. Im Anschluss an diesen Vorgang übermittelt der gewählte Sender die entsprechenden Daten regelmäßig (Periodic) oder nur bei Änderung der Daten (On Event). Eine wiederholte Anfrage ist dazu nicht erforderlich.[44]

Die grundsätzlichen Abläufe der genannten Kommunikationsprinzipien sind in der nachkommenden Darstellung ersichtlich. Auf der linken Seite der Abbildung sind die Kommunikationsvorgänge des Client-Server-Verfahrens und auf der rechten Seite die des Producer-Consumer-Verfahrens anhand eines Beispiels ersichtlich.

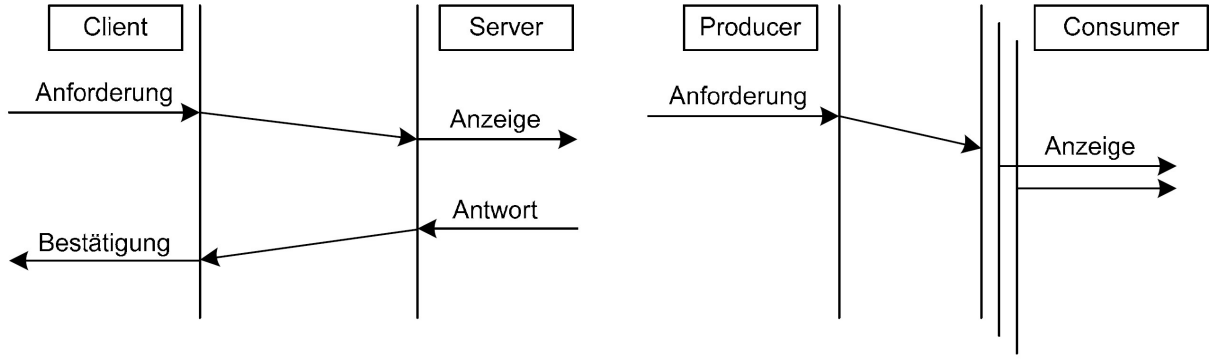


Abb. 10: Kommunikationsprinzipien, Quelle: Reif (2014), S. 6.

[43] Vgl. Reif (2014), S. 6.

[44] Vgl. Zimmermann/Schmidgall (2014), S. 23.

2.1.6 Übertragungsarten

Bei der Datenübertragung über Busleitungen wird zwischen den beiden Verfahren, zeichen-basierte und bitstrom-basierte Übertragung, unterschieden. Das zeichen-basierte Verfahren wird üblicherweise bei niedrigen Bitraten, wie zum Beispiel bei K-Line und LIN, verwendet. Demgegenüber wird die bitstrom-basierte Übertragung in der Regel bei hohen Bitraten, beispielsweise bei CAN- oder FlexRay-Bussystemen, eingesetzt. Die zeichen-basierte Übertragung wird auch Start-Stopp-Verfahren, asynchrone Übertragung oder UART-basierte Übertragung (Universal Asynchronous Reciever and Transmitter) genannt. Bei diesem Verfahren werden die zu übermittelnden Informationen inklusive Kopf und Datensicherung in 8 Bit-Gruppen eingeteilt und jedes 8 Bit-Paket einzeln übermittelt. In der Zeit zwischen den Zeichen befindet sich die Busleitung im Standby-Modus (logisch 1). Der Beginn eines Zeichens erfolgt mit einem Startbit (logisch 0). Im Anschluss daran werden die 8 Bit des Zeichens beginnend mit dem LSB (Least Significant Bit), eventuell ein Paritätsbit zur Fehlererkennung und ein Stoppbit (logisch 1) übermittelt. Bis zum Start des darauffolgenden Zeichens befindet sich die Busleitung abermals im Standby-Modus. Der Vorteil dieser Übertragungsart ist, dass der benötigte Kommunikationscontroller sehr einfach aufgebaut ist und als UART auf beinahe jedem Mikrocontroller bereits implementiert ist.[45]

In der folgenden Abbildung sind eine zeichen-basierte Übertragung einer Botschaft, die Aufteilung dieser in 8 Bit-Gruppen (Zeichen) und der Beginn der nächsten Botschaft dargestellt. Die Abkürzungen ICB und IFB stehen für Inter Character Break und Inter Frame Break.

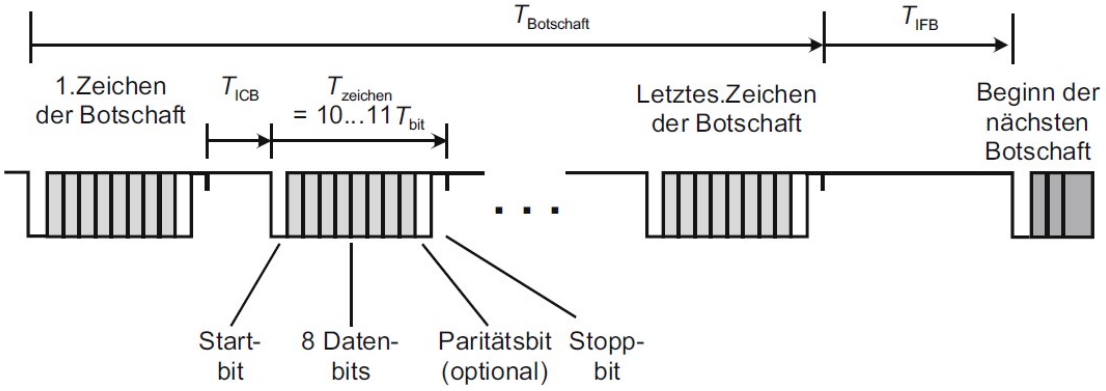


Abb. 11: Zeichen-basierte Übertragung, Quelle: Zimmermann/Schmidgall (2014), S. 26.

Beim bitstrom-basierten Verfahren, auch synchrone Übertragung genannt, werden alle zu einer Nachricht zugeordneten Daten einschließlich Kopf und Datensicherung ohne Unterbrechung als ein zusammengesetzter Block (Frame) im Bittakt übermittelt. Dieser Vorgang wird auch als Bitstrom bezeichnet. Die Unterbrechung (Pause, Inter Frame Break) zwischen den Frames wird über das Protokoll mit Minimal- und Maximalwerten definiert. Durch die pausenlose Übertragung einer Botschaft ergibt sich gegenüber dem zeichen-basierten Verfahren eine wesentlich höhere Nutzdatenrate. Demgegenüber ist der dazu notwendige Kommunikationscontroller deutlich komplexer aufgebaut. Dieser wickelt die gesamte Datenübermittlung hardwareseitig ab, damit die entsprechende Protokollsoftware bloß die

[45] Vgl. Zimmermann/Schmidgall (2014), S. 25.

Nachricht abholen und die weitere Verarbeitung durchführen muss. Empfänger und Sender arbeiten typischerweise mit einem eigenen Taktgenerator für den Bittakt. Kleinere Toleranzen sind trotz derselben nominalen Bitrate nicht zu vermeiden. Aufgrund dieser Toleranzen bei größeren Frames muss der Bittaktgenerator in der Zeit des Empfangs einer Nachricht nachsynchronisiert werden. Ein Beispiel für eine derartige Synchronisation in der Kfz-Bustechnik ist das Bit-Stuffing, das beispielsweise beim CAN-Protokoll verwendet wird.[46]

Wie die Übertragung einer Botschaft bei einem bitstrom-basierten Verfahren durchgeführt wird, ist in der folgenden Darstellung veranschaulicht. Im Unterschied zur zeichen-basierten Übertragung wird hier die gesamte Botschaft ungeteilt übermittelt.

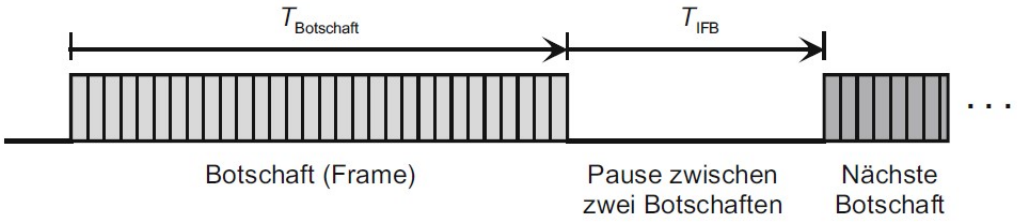


Abb. 12: Bitstrom-basierte Übertragung, Quelle: Zimmermann/Schmidgall (2014), S. 26.

2.1.7 Protokolle

Protokolle können grundsätzlich in Bezug auf zwei unterschiedliche Prinzipien eingeteilt werden. Teilnehmerorientierte Protokolle sehen vor, dass die übermittelten Botschaften eine explizite Identifikation der am Sendevorgang beteiligten Busteilnehmer beinhalten. Für gewöhnlich ist diese Identifikation die Adresse des Empfängers der Nachricht. Demgegenüber werden bei nachrichtenorientierten Protokollen die Botschaften durch eine spezielle Kennung, auch Identifier genannt, eindeutig erkannt. Das Ziel der Übermittlung wird dabei nicht festgelegt, da die jeweiligen Teilnehmer selbst entscheiden ob die Botschaft verarbeitet wird oder nicht.[47]

Für die Verwirklichung dieser Arbeit sind die beiden Protokollgruppen Transportprotokolle und Anwendungsprotokolle wesentlich. Diese werden im Folgenden kurz und in den nachkommenden Abschnitten näher erläutert.

Die Aufgaben eines Transportprotokolls liegen in der Segmentierung bzw. Desegmentierung von Informationen, in der Flusskontrolle und in der Weiterleitung von Botschaften in fremde Netze (anderer Adressraum oder andere Adressierungsmethode). Die Flusskontrolle umfasst die Steuerung des zeitlichen Abstands der Botschaften und die zeitliche Überwachung der gesamten Kommunikationsabläufe.[48] Mit dem Begriff Segmentierung wird das Zerstückeln einer Information bezeichnet. Das Zusammensetzen auf der Empfängerseite wird wiederum Desegmentierung genannt.[49]

[46] Vgl. Zimmermann/Schmidgall (2014), S. 26ff.

[47] Vgl. Reif (2014), S. 6.

[48] Vgl. Zimmermann/Schmidgall (2014), S. 153.

[49] Vgl. Zimmermann/Schmidgall (2014), S. 23.

Anwendungsprotokolle werden hingegen, wie die Bezeichnung vermuten lässt, für unterschiedliche Applikationen eingesetzt. Beispiele dafür sind Diagnoseanwendungen wie On-Board Diagnostic (OBD) Unified Diagnostic Services (UDS) oder Applikationsprotokolle wie CAN Calibration Protocol (CCP) und Extended Calibration Protocol (XCP).[50]

2.1.8 Buszugriffsverfahren

Das Buszugriffsverfahren definiert, inwiefern Teilnehmer auf das verwendete Bussystem zugreifen können. Dabei wird speziell die Handhabung von zeitgleichen Zugriffsversuchen festgelegt. Auch wenn eine Vielzahl an verschiedenen Bussystemen definiert sind und eingesetzt werden, bauen alle auf den zentralen Grundprinzipien des Buszugriffs auf.[51]

Dabei wird zwischen vorhersagbaren und zufälligen Verfahren unterschieden. Bei den vorhersagbaren Verfahren wird der Buszugriff über definierte zeitabhängige Parameter des Netzwerks festgelegt. Dabei kann nie mehr als ein Busteilnehmer gleichzeitig senden. Zugriffskollisionen können daher ausgeschlossen werden, sofern alle Busteilnehmer das Verfahren befolgen. Bei zufälligen Verfahren hingegen versucht jeder Teilnehmer Daten zu übermitteln, wenn der Bus scheinbar nicht von einem anderen Teilnehmer belegt wird. Der Zeitpunkt des Buszugriffs erfolgt also zufällig. Dementsprechend können Kollisionen zwischen Übertragungen auftreten, denen unbedingt entgegengewirkt werden muss.[52]

Vorhersagbare Verfahren, auch deterministische Buszugriffsverfahren genannt, werden in zentral und dezentral gesteuerte Verfahren gegliedert. Zentral arbeitende Verfahren sind durch eine geringere Komplexität in Bezug auf die Steuerungslogik gekennzeichnet. Demgegenüber sind diese Verfahren jedoch hinsichtlich der direkten Abhängigkeit des zentralen Steuerungselements weniger performant. Da bei den zufälligen Verfahren der gleichzeitige Zugriff mehrerer Teilnehmer zugelassen wird, ist das Systemverhalten nicht deterministisch. Diese Verfahren werden aufgrund dessen auch mit der Abkürzung CSMA (Carrier Sense Multiple Access) gekennzeichnet. Eingeteilt werden diese Verfahren in kollisionsfreie und nicht kollisionsfreie Verfahren. Nicht kollisionsfreie Verfahren können Kollisionen zwar detektieren, können jedoch für gewöhnlich erst nach dem Übermitteln von Botschaftsteilen oder einer gesamten Nachricht Korrekturmaßnahmen setzen. Demgegenüber verhindern kollisionsfreie Verfahren das Entstehen von Kollisionen während eines Kommunikationsablaufs zur Gänze. Ein Bussystem, das dieses Verfahren nutzt ist der CAN-Bus.[53]

In der nachkommenden Abbildung (Abb. 13) sind die beschriebenen Buszugriffsverfahren und deren genannte Unterteilungen dargestellt.

[50] Vgl. Zimmermann/Schmidgall (2014), S. 9.

[51] Vgl. Reif (2014), S. 8.

[52] Vgl. Reif (Hrsg.) (2011), S. 74f.

[53] Vgl. Reif (2014), S. 8f.

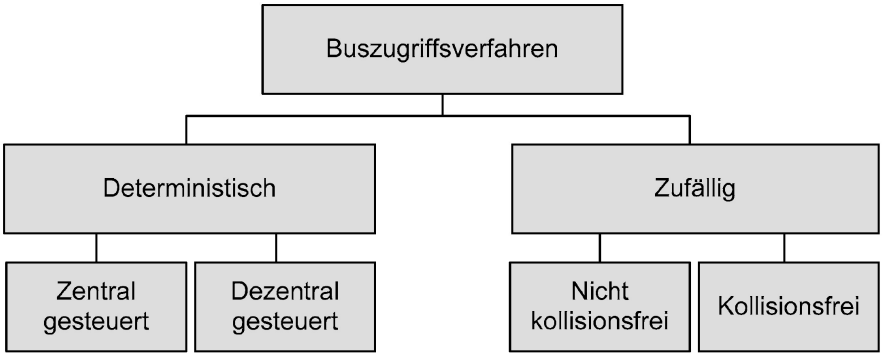


Abb. 13: Buszugriffsverfahren, Quelle: Reif (2014), S. 8.

Ein deterministisches und zentral gesteuertes Buszugriffsverfahren ist das Master-Slave-Verfahren. Bei diesem Verfahren agiert ein Busteilnehmer als Master, welcher die Datenübermittlung starten darf. Dieser Master fragt die übrigen Busteilnehmer, die somit als Slaves agieren, wiederholt oder bei Bedarf ab. Die Slaves dürfen Informationen nur als Antwort auf eine an sie gesendete Anfrage übertragen. Auch bei dringenden Botschaften müssen Slaves auf die Anfrage durch den Master warten.[54] Es gibt jedoch einige Master-Slave Protokolle, die es erlauben, dass sich ein Slave beim Master meldet, um Informationen zu übermitteln.[55] Eine Kommunikation zwischen Slaves ist hingegen nicht möglich. Sollte der Master ausfallen, führt dies zu einem vollständigen Versagen der Kommunikation, außer es wird ein Verfahren eingesetzt, welches automatisch einen neuen Master festlegt (z.B. LIN).[56]

Dezentral gesteuerte Zugriffsverfahren in Bussystemen werden entweder token- oder zeitgesteuert abgehandelt. Bei tokengesteuerten Verfahren wird die Buszuteilung mittels Vergabe eines Tokens geregelt. Demgegenüber legen zeitgesteuerte Verfahren für Busteilnehmer gesonderte Zeitfenster für den Zugriff auf das Bussystem fest. Dieses Prinzip des Buszugriffs wird auch als TDMA, Time Division Multiple Access, bezeichnet.[57] Das tokengesteuerte Verfahren wird auch Token-Passing genannt. Dabei wird das Token, welches als Sendeberechtigung dient, reihum weitergegeben. Nach einem definierten Prinzip gibt der Busteilnehmer, sofern entweder der Zeitrahmen zum Senden vorüber ist oder falls keine weiteren Informationen zum Übertragen vorhanden sind, das Token an den nächsten Teilnehmer weiter. Dabei sollte die Über- bzw. Weitergabe der Sendeberechtigung überwacht werden.[58] Bei TDMA ist aufgrund der festgelegten Sendezeitfenster der Busteilnehmer ein fixer Ablaufplan notwendig. Es ist jedoch für gewöhnlich kein zentraler Busteilnehmer vorhanden, der diesen festlegt. Dagegen sind Konzepte verfügbar, bei denen ein Wechsel zwischen unterschiedlichen Ablaufplänen entsprechend des Bedarfs möglich ist. Wichtig bei einem Einsatz von TDMA ist, dass die integrierten Uhren aller Busteilnehmer synchron laufen, da die Sendezeitfenster sehr genau eingehalten werden müssen.[59]

[54] Vgl. Zimmermann/Schmidgall (2014), S. 28.

[55] Vgl. Reif (Hrsg.) (2011), S. 75.

[56] Vgl. Zimmermann/Schmidgall (2014), S. 28.

[57] Vgl. Reif (2014), S. 9.

[58] Vgl. Zimmermann/Schmidgall (2014), S. 28.

[59] Vgl. Reif (Hrsg.) (2011), S. 75.

Die Weiterentwicklung von TDMA ist FTDMA (Flexible Time Division Multiple Access). Bei diesem Verfahren läuft die Steuerung über verteilte Slot-Zähler in den Teilnehmern des Bussystems. Diese sogenannten Slot-Zähler werden beim Start eines Datenübertragungszyklus abgeglichen (synchronisiert) und daraufhin gestartet. Dabei kennzeichnet der entsprechende Zählerstand die Identifikation, den Identifier, einer Nachricht. Demnach erhält ein Teilnehmer mit seiner Nachricht Buszugriff, wenn der Zählerstand mit dem definierten Identifier übereinstimmt. Während dem Sendevorgang werden die Zähler gestoppt. Hat der jeweilige Teilnehmer keine Informationen zu übertragen, wird der Ablauf nach einer kurzen Verweilpause mit dem darauffolgenden Zählerstand fortgesetzt.[60]

Bei den Verfahren mit zufälligen Buszugriff wird zwischen kollisionsfrei und nicht kollisionsfrei unterschieden. Die nicht kollisionsfreien Verfahren werden mit CSMA/CD (Collision Detection) bezeichnet, da diese den zeitgleichen Versuch eines Buszugriffs detektieren können. Dabei sind alle Nachrichten am Bus mit der gleichen Berechtigung versehen. Folglich kann aufgrund der möglichen Kollisionen nicht gewährleistet werden, in welchem Zeitrahmen eine Botschaft übermittelt werden kann. Somit kann kein deterministischer Betrieb sichergestellt werden. Kollisionsfreie Verfahren werden hingegen mit CSMA/CR (Collision Resolution) bezeichnet. Dieses Verfahren sieht vor, dass derjenige Sender übermitteln darf, dessen Nachricht die höhere Priorität, die sich im Header der Nachricht befindet, besitzt. Dieser Ablauf nennt sich Arbitrierung. Ein Teilnehmer mit einer Botschaft, welche eine niedrigere Priorität aufweist, nimmt seinen Sendewunsch sofort zurück. Die Nachricht mit der höheren Priorität wird hingegen ohne Kollision weiter übermittelt. Dieses Verfahren kann jedenfalls für Nachrichten mit einer hohen Priorität ein deterministisches Verhalten garantieren.[61]

2.1.9 Botschaftsaufbau

Ein Datenübertragungsprotokoll ist ein Paket, dass der Übermittlung von Daten unter Teilnehmern in einem Bussystem dient und sich aus den Bereichen Kopf (Header), Datensicherung (Trailer) und den zu übermittelten Daten (Body), auch Nutzdaten genannt, zusammensetzt. Besteht eine Botschaft lediglich aus wenigen Bits an Nutzdaten, sollte eine hohe Dateneffizienz angestrebt werden um eine bestmögliche Ausnutzung der Bandbreite des Bussystems sicherstellen zu können.[62]

Inwiefern eine Botschaft aufgebaut ist, zeigt die nachfolgende Darstellung eines Datenübertragungsprotokolls. Dabei sind die beschriebenen Teile Kopf, Daten und Datensicherung ersichtlich.

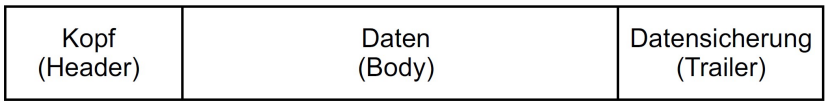


Abb. 14: Aufbau eines Datenübertragungsprotokolls, Quelle: Paulweber/Lebert (2014), S. 255.

[60] Vgl. Reif (2014), S. 9.

[61] Vgl. Zimmermann/Schmidgall (2014), S. 28.

[62] Vgl. Paulweber/Lebert (2014), S. 255.

Der Kopf beinhaltet Informationen wie die Adressierung des Senders und Empfängers, die Anzahl der zu übermittelnden Informationen und fallweise deren Art. An die Nutzdaten einer Botschaft wird meist ein Nachspann, auch Trailer oder Datensicherung genannt, angefügt. Dieser enthält Parameter zur Fehlerkorrektur und -prüfung. Im Umfang der Nutzdaten, auch Payload genannt, werden oftmals mehrere, vom Inhalt nicht zwingend direkt zusammenhängende Informationen übermittelt. Handelt es sich dabei um Messwerte, wird von Signalen gesprochen. Bestimmt wird ein solches Signal durch die Länge und die Lage in Bezug auf den Anfang des Payload-Bereichs (Offset). Zusätzlich wird das Signal fallweise anhand einer Berechnungsregel bestimmt, welche die Beziehung zwischen dem physikalischen Wert und dessen hex-Codierung definiert. Laut dem ISO/OSI-Schichtenmodell durchlaufen die Botschaften von der Applikation bis zur Übermittlung über das Bussystem mehrere Ebenen, die jeweils spezifische Header sowie Trailer benutzen. Die Nachricht der nächsthöheren Schicht wird beim Übertragen auf der nächstniedrigeren Schicht als Nutzdatenpaket behandelt und im Zuge dessen mit Header und Trailer der aktuellen Ebene erweitert. Beim Empfangen einer Botschaft hingegen werden die beiden genannten Teile, Trailer und Header, der aktuellen Ebene entfernt, ehe die Nutzdaten dieser Ebene an die nächsthöhere Schicht übergeben werden. So werden bestenfalls innerhalb einer Ebene lediglich die Informationen der Schicht entsprechenden Header und Trailer bearbeitet, der Nutzdateninhalt ist hingegen für die Ebene selbst nicht von Bedeutung. Gemäß diesem genannten Ablauf entsteht ein so genannter Protokollstapel, auch als Protocol Stack bezeichnet. Der Kopf einer Ebene wird meist Protocol Control Information (PCI) genannt, da er das für die Ebene zugeordnete Protokoll definiert. Die Payload wird als Service Data Unit (SDU) und die gesamte Botschaft als Protocol Data Unit (PDU) bezeichnet.[63]

2.1.10 Kopplung von Bussystemen

Je nach Anforderung an das Bussystem werden bestimmte Netzwerktopologien und Netzwerkprotokolle gewählt. Die verschiedenen Netzwerkprotokolle sind untereinander aber nicht kompatibel. Dementsprechend können Informationen ohne weitere Zusatzmaßnahmen nicht unter den unterschiedlichen Netzwerken versendet werden. Aus diesem Grund müssen Bussysteme gekoppelt werden.[64] Unter der Kopplung von Bussystemen versteht man die Verknüpfung von gleichen oder unterschiedlichen Bussystemen. Dies ist bei einer Vielzahl an Applikationen in der modernen Bustechnik erforderlich. Die Verbindung kann dabei lediglich als Signalverstärkung dienen oder beispielsweise die Übermittlung einer Nachricht in ein Netzwerk, welches ein anderes Protokoll nutzt, ermöglichen. Dazu haben sich gemäß den Anforderungen die vier Systemkomponenten Repeater, Bridge, Router und Gateway etabliert, welche auch als Hybridformen eingesetzt werden können.[65]

Die wichtigsten Funktionen dieser vier Einheiten (Repeater, Bridge, Router und Gateway) sowie die Zuordnung zu den jeweiligen Ebenen des ISO/OSI-Referenzmodells sind in der folgenden Abbildung ersichtlich.

[63] Vgl. Zimmermann/Schmidgall (2014), S. 18f.

[64] Vgl. Reif (Hrsg.) (2011), S. 87.

[65] Vgl. Reif (2014), S. 7.

Baustein	Funktionen	Zuordnung ISO/OSI-Modell
Repeater	Verbindung von Bussystemen zur Signalverstärkung oder Signalaufbereitung	Schicht 1
Bridge	Speicherung und zeitverschobene Übertragung Verbindung von Bussystemen zur Weiterleitung von Botschaften an verbundene Bussysteme ohne explizite Adressierung	Schicht 2
Router	Verbindung von Bussystemen zur gerichteten Weiterleitung von Botschaften an verbundene Bussysteme	Schicht 3
Gateway	Verbindung von Bussystemen zur Adress-, Geschwindigkeits- oder Protokollwandlung	Schichten 5 bis 7

Abb. 15: Übersicht Systembausteine in Bussystemen, Quelle: Reif (2014), S. 8.

2.2 Bussysteme in der Fahrzeugtechnik

Bussysteme in der Fahrzeugtechnik haben zur Aufgabe, die Kommunikation zwischen verschiedenen elektronischen Systemkomponenten zu gewährleisten. Die Kommunikation kann sich dabei auf das vollständige Kraftfahrzeug oder lediglich auf einzelne Funktionsbereiche wie zum Beispiel auf die Getriebesteuerung des Antriebsstrangs beziehen.[66]

In diesem Abschnitt werden die allgemeinen Inhalte zu den gängigsten Bussystemen in der Fahrzeugtechnik, anschließend die Bussysteme CAN, CAN FD und FlexRay und schlussendlich die Steuergerätetechnik erläutert.

2.2.1 Grundlagen

Durch den raschen Anstieg an elektronischen Systemen und dem damit verbundenen Bedarf an gegenseitigem Informationsaustausch im Fahrzeug wurden ab den 1990er Jahren serielle Bussysteme eingesetzt, die damit die bis dahin angewandte Technik der konventionellen Verdrahtung abgelöst haben. Das Streben und das Verlangen nach mehr Zuverlässigkeit, Sparsamkeit, Luxus und vor allem die zunehmenden rechtlichen Ansprüche im Bereich der Umweltverträglichkeit für Fahrzeuge kann lediglich anhand weiterer Elektronik erreicht werden. Dementsprechend steigt die Anzahl an elektronischen Systemkomponenten in Kraftfahrzeugen kontinuierlich bzw. unaufhörlich an.[67]

In der folgenden Darstellung ist die Entwicklung der Anzahl an vernetzten elektronischen Systemen, Signalen und Bussystemen einer Fahrzeugbaureihe im Premiumsegment eines bekannten deutschen Automobilherstellers ersichtlich. Zusätzlich ist die erforderliche Bandbreite angeführt, die entsprechend der Anzahl an elektronischen Systemkomponenten ebenso beständig ansteigt.

[66] Vgl. Reif (2014), S. 13.

[67] Vgl. Reif (Hrsg.) (2011), S. 82.

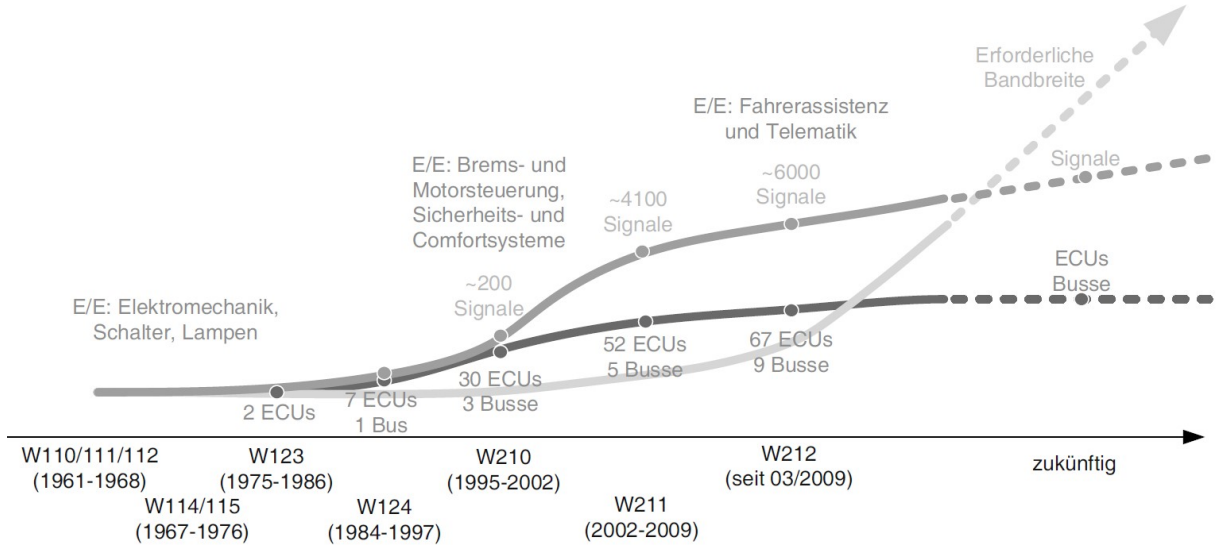


Abb. 16: Entwicklung der Elektronikkomponenten im Kfz, Quelle: Streichert/Traub (2012), S. 13.

Derartige Systeme erfüllen dabei zahlreiche unterschiedliche Funktionen, wie Getriebe- oder Motorsteuerung, wobei diese nochmals in Teilfunktionen gegliedert werden. Die Unterteilung der Fahrzeugfunktionen in deren Teilbereiche wird auch als Partitionierung bezeichnet. Ebenfalls Partitionierung oder Mapping wird die Aufteilung der Aufgaben auf die Steuergeräte im Fahrzeug genannt. Naheliegend wäre dabei die etlichen elektronischen Systemkomponenten in wenigen oder nur in einem Steuergerät unterzubringen. Dadurch könnte man eine wesentliche Kostensenkung erreichen. Demgegenüber stehen aber enorme Nachteile wie zu geringe Flexibilität hinsichtlich der massiven Variantenvielfalt an Ausstattungen im Fahrzeug oder zu geringe Störsicherheit der einzelnen Bereiche, da sich diese bei einer zentralen Steuereinheit gegenseitig beeinflussen könnten. Folglich werden getrennte Funktionen auf getrennten Steuergeräten implementiert.[68]

Daher ist bei systemübergreifenden Funktionen eine Koordination zwischen den einzelnen Einheiten notwendig. Diesbezüglich sind hohe Informationsmengen zu übertragen und leistungsfähige Komponenten sowie Kommunikationssysteme erforderlich. Hinsichtlich dieser Anforderungen wurden spezielle serielle Bussysteme für die Fahrzeugtechnik entwickelt. Im Vergleich zur konventionellen Verdrahtung ergeben sich bei der Anwendung von Bussystemen mehrere Vorteile. Ein bedeutender Vorteil ist die Einsparung von Kosten, Bauraum und Gewicht, da weniger Leitungen im Kabelbaum benötigt werden. Die höhere Funktionssicherheit und Zuverlässigkeit aufgrund einer geringeren Anzahl an Steckverbindern ist ebenso ein Vorteil wie auch die daraus resultierende Vereinfachung der Fahrzeugmontage während der Fertigung. Des Weiteren sind die Mehrfachnutzung von verschiedenen Bussignalen und die unkomplizierte Verknüpfung zwischen Systemkomponenten und dem entsprechenden Bussystem Eigenschaften, die wesentliche Vorteile bringen. Schlussendlich spricht aber auch die Flexibilität in Bezug auf die Ausstattungsmöglichkeiten (inklusive Sonderausstattungen) eines Fahrzeugs für den Einsatz von Bussystemen.[69]

[68] Vgl. Borgeest (2014), S. 89.

[69] Vgl. Reif (Hrsg.) (2011), S. 83.

2.2.2 Anforderungen

Bei der Auswahl eines Bussystems müssen sowohl wirtschaftliche Aspekte, wie Leitungs- und Komponentenkosten, als auch technische Rahmenbedingungen bedacht werden. Die wichtigsten technischen Auswahlkriterien sind die Datenübertragungsrate, die Störsicherheit, die Echtzeitfähigkeit und die Anzahl der erforderlichen Busteilnehmer im Fahrzeug. Die notwendige Übertragungsrate ist vom Anwendungsfall abhängig. Die Übertragung von Audiosignalen benötigt beispielsweise eine höhere Datenrate als das Signal eines Türkontakts. Bei der Störsicherheit muss zwischen sicherheitsrelevanten und weniger sicherheitskritischen Signalen unterschieden werden. Bei sicherheitsrelevanten Daten muss ein Netzwerkprotokoll mit Fehlererkennung, wie Checksummenüberprüfung oder Paritätsbits, genutzt werden. Auch bei der Echtzeitfähigkeit muss der Anwendungsfall berücksichtigt werden. Beispielsweise muss ein Antiblockiersystem im Fahrzeug in Echtzeit reagieren, ein Fensterhebermotor hingegen nicht zwingend. Dementsprechend ergibt sich die Anforderung nach weicher oder harter Echtzeit gemäß den unterschiedlichen Anwendungen. Die Anzahl der notwendigen Teilnehmer ist abhängig von der Fahrzeugklasse. Bei Komfortsystemen kann sich eine hohe Anzahl an Knoten ergeben. Bei Bedarf werden mehrere gleichartige Bussysteme verwendet und miteinander verknüpft.[70]

Darüber hinaus müssen zukünftig in den stark vernetzten Systemen weitere technische Ansprüche wie ausfallsichere Netzwerktopologien oder deterministisches Kommunikationsverhalten bei der Auswahl des entsprechenden Bussystems bedacht werden. Schlussendlich müssen aber auch Anforderungen aus dem Servicebereich, wie die Diagnosefähigkeit, oder der Produktion, wie ein möglicher Teilnetzbetrieb, bei der Auswahl eines Systems miteinbezogen werden. Entsprechend dem Einsatzbereich sowie der erforderlichen Datenübertragungsrate und der Nachrichtenlänge können spezifische Anforderungsklassen für die unterschiedlichen Bussysteme (z.B. CAN, LIN, MOST) festgelegt werden. Diese Anforderungsklassen werden auch SAE-Klassen genannt.[71]

In der folgenden Abbildung sind die vier SAE-Klassen inklusive deren wesentlichen Merkmalen angeführt.

SAE-Klasse	Merkmale	Typisches Bussystem
A	Vernetzung von Aktoren und Sensoren Geringe Datenraten (ca. 10 kBit/s) Geringe Ausprägung von Fehlererkennungs- und -behebungsmechanismen	LIN-Bus
B	Vernetzung von Steuergeräten (z. B. Komfortbereich) Mittlere Datenraten (ca. 125 kBit/s) Komplexe Mechanismen zur Fehlererkennung und -behebung	CAN-Bus („Low Speed“)
C	Vernetzung von Steuergeräten mit „einfachen“ Echtzeitanwendungen (z. B. Antriebsstrang) Hohe Datenraten (bis zu 1 MBit/s)	CAN-Bus („High Speed“)
D	Multimedia-Anwendungen Sehr hohe Datenraten (bis zu 10 MBit/s) und Botschaftslängen	MOST-Bus

Abb. 17: SAE-Klassen für Bussysteme, Quelle: Reif (2014), S. 14.

[70] Vgl. Reif (Hrsg.) (2011), S. 83ff.

[71] Vgl. Reif (2014), S. 13.

2.2.3 Vernetzungsarchitektur im Fahrzeug

Die Ebene der Vernetzungsarchitektur umfasst Steuergeräte, Sensoren und Aktoren, die notwendig sind, um die Systemteile (Funktionsblöcke, Softwarekomponenten) der Funktions- und Softwarearchitektur ausführen zu können. Zusätzlich müssen diese Komponenten die Datenübermittlung gewährleisten und dabei elektrisch versorgt werden. Dementsprechend wird die genannte Systemschicht üblicherweise in den vier Teilebenen Kommunikationsstruktur, Leistungsversorgung, Komponentenarchitektur und Leitungssatz eingeteilt. Die Kommunikationsstruktur, die Steuergeräte, Sensoren und Aktoren umfasst, beinhaltet den Aufbau von Netzwerken. Die Kommunikation unter diesen Komponenten erfolgt entweder über Bussysteme wie FlexRay, CAN und LIN oder über eine dedizierte bzw. proprietäre Verbindung, wie beispielsweise bei der Anbindung eines Sensors oder Aktors. Die Verbindung zwischen verschiedenen Kommunikationssystemen oder Strängen eines Busses kann je nach ISO/OSI-Ebene mittels Komponenten wie Sternkoppler, Repeater, Hubs, Bridges, Switches, Router oder Gateways erfolgen. Die Ebene Leistungsversorgung umfasst die elektrische Versorgung der Komponenten. In der heutigen Fahrzeugtechnik erfolgt dies üblicherweise über die Batterie oder einen Generator. In Zukunft können dazu auch On-Board-Charger, welche die elektrische Leistung dem öffentlichen Stromnetz entnehmen, eingesetzt werden. In der Komponentenarchitektur wird die interne Beschaltung von Steuergeräten, Sensoren und Aktoren sowie Leistungsverteilern beschrieben. Damit umfasst diese Ebene eine Darstellung der genannten Komponenten aus den Teilbereichen Vernetzungsarchitektur und Leistungsversorgung. Im Umfang der Unterebene Leitungssatz werden die Verknüpfungen unter den Systemkomponenten der Kommunikationsstruktur und der Leistungsversorgung spezifiziert. Dabei wird die elektrische Verknüpfung festgelegt sowie der Leitungssatz spezifiziert. Dies umfasst beispielsweise die Festlegung der Anzahl an Pins und Leitungen pro Verbindung sowie die verschiedenen Leitungs- bzw. Kabeltypen.[72]

Inwiefern die Kommunikationsstruktur eines Bussystems in einem Fahrzeug aufgebaut sein kann, wird in der folgenden Abbildung in vereinfachter Weise dargestellt.

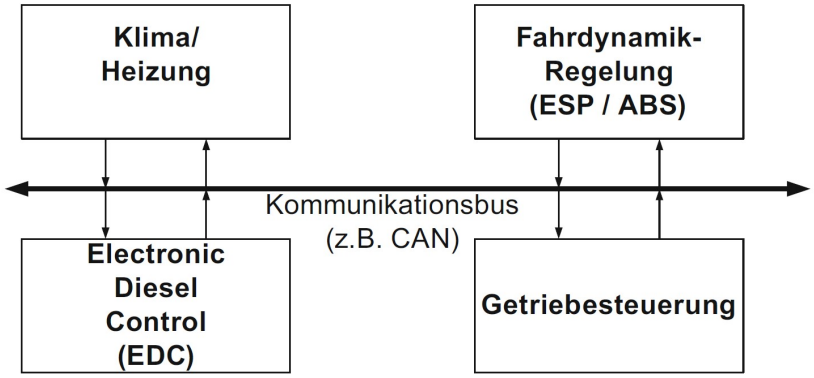


Abb. 18: Kommunikationsstruktur, Quelle: Borgeest (2014), S. 91.

[72] Vgl. Streichert/Traub (2012), S. 21ff.

2.2.4 Protokolle und Standards

Im Folgenden sind die wichtigsten Transportprotokoll- bzw. Anwendungsprotokollstandardisierungen von Kfz-Bussystemen und deren Anwendungsgebiete angeführt. Dazu sind die entsprechenden europäischen und amerikanischen Standards gelistet. Protokolle mit geringer Relevanz und Markbedeutung sind nicht dargestellt.

Transportprotokoll	Anwendung	Europ. Standards	US-Standards
ISO TP	CAN	ISO 15765-2	
FlexRay TP	FlexRay	ISO 10681-2	
SAE J1939	CAN		SAE J1939/21
TP 1.6, TP 2.0	CAN	Hausstandard VW/Audi/Seat/Skoda	
DoIP	Ethernet	ISO 13400-2	

Abb. 19: Transportprotokolle, Quelle: Zimmermann/Schmidgall (2014), S. 9.

Protokoll	Anwendung	Europ. Standards	US-Standards
ISO 9141-CARB	Diagnose US OBD	**ISO 9141-2** Veraltete US-Diagnoseschnittstelle	SAE J1979, J2190
KWP 2000 Keyword Protocol	Diagnose (allgemein und OBD)	**ISO 14230** KWP 2000 Diagnostics on K-Line	
UDS Unified Diagnostic Services	Diagnose (allgemein und OBD)	**ISO 14229** UDS Unified Diagnostic Services **ISO 15765** UDS on CAN	
OBD	Diagnose US OBD European OBD	**ISO 15031** (identisch mit den US-Standards)	SAE J1930, J1962, J1978, J1979, J2012, J2186
CCP Can Calibration Protocol **XCP** Extended Calibration Protocol	Applikation	**ASAM AE MCD** ASAM-Konsortium Automotive Electronics Measurement, Calibration and Diagnostics	

Abb. 20: Anwendungsprotokolle, Quelle: Zimmermann/Schmidgall (2014), S. 9.

Die für diese Arbeit relevanten Protokolle, wie XCP und CCP werden in den entsprechenden Kapiteln genauer erarbeitet. Dieser Abschnitt dient lediglich zur Übersicht der gängigen Protokolle. Ebenso werden die Standardisierungen der Bussysteme CAN, CAN FD und FlexRay in den nachfolgenden Abschnitten angeführt und erläutert.

2.2.5 CAN

CAN ist die Abkürzung für Controller Area Network und ist das aktuell am häufigsten verwendete Bussystem in der Fahrzeugtechnik sowohl im Lowspeed- als auch im Highspeed-Bereich.[73] Das CAN-Protokoll wurde im Jahr 1991 erstmalig als Bussystem in einer Fahrzeug-Serienproduktion eingesetzt.[74] Der CAN-Bus ist über die Jahre zur globalen Standardlösung für die unterschiedlichen Kommunikationsbereiche im Fahrzeug aufgestiegen. Bei der Entwicklung bzw. der Spezifikation vom CAN-Protokoll ist es gelungen, den Ansprüchen nach vielfältiger Erweiterbarkeit und vielseitiger Einsatztauglichkeit nachzukommen und diese dabei mit Fehlerfreiheit, Übersichtlichkeit und Ausfallsicherheit zu verbinden. Aufgrund der enormen Stückzahlen im Automobilbereich sind CAN-Komponenten immer kostengünstiger geworden.[75] Das CAN-Bussystem wurde vom Unternehmen Bosch spezifiziert bzw. entwickelt und war der erste digitale Bus, der sich markenübergreifend in der Automobilbranche etablierte. Nach der Entwicklung durch Bosch wurden durch die ISO unter anderem die CAN-Normen ISO11898-1/2/3 definiert.[76] Die Norm ISO11989-1 entspricht dabei den Bosch Spezifikationen CAN 2.0A und 2.0B.[77]

Der CAN-Bus liefert eine maximale Übertragungsrate von 1 MBit/s. Dazu wird in der Regel ein geschirmtes sowie verdrilltes Adernpaar als Busleitung verwendet. Die maximal zulässige Länge der Busleitung ist abhängig von der gewählten Bitrate, da beim CAN-Protokoll die Ausbreitungszeit eines Bits aufgrund der eingesetzten Busarbitrierung von Bedeutung ist.[78] In der Fahrzeugtechnik werden meist Übertragungsraten von 125 kBit/s, 250 kBit/s oder 500 kBit/s gewählt, da Raten über 500 kBit/s die Auswahl der Netzwerktopologie im Fahrzeug einschränken. Eine Botschaft kann maximal acht Byte an Nutzdaten beinhalten. Als Bitcodierungsverfahren wird die NRZ-Codierung verwendet. Zur Synchronisation der Teilnehmer in einem CAN-Netzwerk wird Bit-Stuffing eingesetzt. Die Arbitrierung am Bus erfolgt gemäß dem Buszugriffsverfahren CSMA/CR. Diese Busarbitrierung wird dabei für jede Nachricht über einen, einmalig im Netzwerk vorhandenen, Identifier, welcher die Botschaftspriorität kennzeichnet, geregelt.[79] Somit zählt das CAN-Bussystem zu den nachrichtenorientierten Bussen. Die Nachrichten werden entweder als Broadcast oder Multicast gesendet. Alle Busteilnehmer prüfen und beurteilen dabei anhand des Identifiers selbstständig ob die aktuell am Bus vorhandene Information relevant ist oder nicht. Dementsprechend werden die Botschaften weiterverarbeitet oder nicht.[80] Das CAN-Protokoll wird in der Fahrzeugtechnik in den verschiedensten Bereichen verwendet, wobei sich deutliche Unterschiede in Bezug auf die Anforderungen an das Bussystem ergeben. Beispielsweise

[73] Vgl. Zimmermann/Schmidgall (2014), S. 57.

[74] Vgl. Reif (Hrsg.) (2011), S. 92.

[75] Vgl. Paulweber/Lebert (2014), S. 255.

[76] Vgl. Borgeest (2014), S. 94.

[77] Vgl. Zimmermann/Schmidgall (2014), S. 58.

[78] Vgl. Paulweber/Lebert (2014), S. 255f.

[79] Vgl. Streichert/Traub (2012), S. 116.

[80] Vgl. Reif (2014), S. 15.

werden Informationen im Bereich der Getriebesteuerung erheblich schneller benötigt als im Komfortbereich, wo die Leitungsabstände zwischen den einzelnen Systemkomponenten wesentlich höher sind. Aufgrund dieser vielfältigen Ansprüche wird bei CAN auf entsprechende Übertragungsraten zurückgegriffen, die für den jeweiligen Anwendungsfall ein bestmögliches Kosten-Nutzen-Verhältnis mit sich bringen. Dabei unterscheidet man zwischen Highspeed- und Lowspeed-CAN-Systemen. Der Highspeed-CAN-Bus wird auch CAN-C genannt, ist in der ISO11898-2 definiert und läuft mit Übertragungsgeschwindigkeiten von 125 kBit/s bis 1 MBit/s. Damit entspricht die Datenübermittlung beispielsweise den Echtzeitanforderungen in Antriebsstrangsystemen. Folglich werden Highspeed-CAN-Busse in den Bereichen Motormanagement, Getriebesteuerung, Fahrzeugstabilisierungssystem sowie bei Kombiinstrumenten eingesetzt. Der Lowspeed-CAN-Bus ist in der ISO11898-3 spezifiziert und wird auch als CAN-B bezeichnet. Die Übertragungsraten liegen bei CAN-B zwischen 5 und 125 kBit/s. Für einige Anwendungen im Karosserie- und Komfortbereich sind diese Raten für deren Echtzeitanforderungen ausreichend. Angewendet wird CAN-B beispielsweise in den Bereichen Komfortsitzverstellung, Fensterhebersteuerung, Außenspiegelverstellung, Beleuchtungsanlage, Panoramadachsteuerung oder Steuerung der Navigationsanlage. Zusätzlich verbreitet sich der CAN-Bus vermehrt in der Fahrzeugdiagnose. Dazu werden die Steuergeräte direkt an das CAN-Bussystem angeschlossen und empfangen so die erforderlichen Daten zur Durchführung der Diagnose.[81]

2.2.5.1 Botschaftsaufbau

Bei CAN-Botschaften wird grundsätzlich zwischen den zwei Varianten, Standard, auch Basic genannt, und Extended unterschieden. Standard-Nachrichten nutzen 11 Bit für den Identifier. Extended-Messages hingegen verwenden 29 Bit für den Identifier. Geeignete CAN-Controller sind zwar in der Lage beide Varianten am gleichen Bus zu verarbeiten, jedoch wird in der Regel ein Typ für das gesamte Bussystem bestimmt. Eine Botschaft-Sonderform ist der Errorframe. Dieser wird von einem Busteilnehmer ausgesendet, wenn ein Übertragungsfehler festgestellt wird. Nachdem sich ein Errorframe aus sechs dominanten Bits zusammensetzt, erzwingt er eine Fehlererkennung bei allen empfangenden Teilnehmern im Bussystem, da er sich gegen alle Botschaften durchsetzt und dabei eine absichtliche Verletzung der Stuffbit-Regel hervorruft.[82]

Die folgende Darstellung gibt einen Überblick über den Aufbau einer CAN-Botschaft mit einem 11 Bit Identifier. Dabei sind die Hauptteile Arbitrierungsfeld, Steuerfeld, Datenfeld, CRC-Feld sowie ACK-Feld ersichtlich. Im Anschluss werden die einzelnen Teile erläutert.

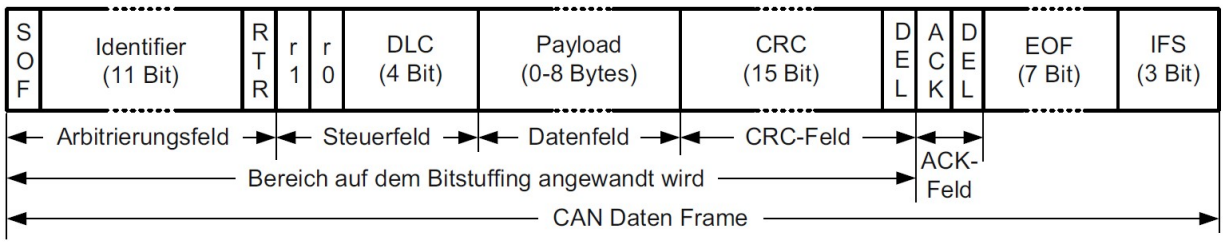


Abb. 21: Aufbau einer CAN-Botschaft, Quelle: Streichert/Traub (2012), S. 116.

[81] Vgl. Reif (Hrsg.) (2011), S. 92.

[82] Vgl. Paulweber/Lebert (2014), S. 257.

Den Anfang einer Botschaft bildet das Start-of-Frame Bit (SOF). Darauf folgen der Identifier, der die Aufgaben Adressierung und Arbitrierung übernimmt und das Remote Transmission Request Bit (RTR), welches angibt ob die Nachricht Daten enthält oder zum Senden auffordert. Das Bit r1 wird als Identifier Extension genutzt. Das r0-Bit ist reserviert. Darauf folgt das aus vier Bit bestehende DLC-Feld. Dieses Feld gibt die Länge des Datenfeldes, auch Payload-Länge genannt, an. Die Länge kann in einem Wertebereich zwischen 0 und 8 Byte angeführt werden. Nach dem DLC-Feld folgt der Nutzdatenbereich und das CRC-Feld. Mittels CRC (Cyclic Redundancy Check) können fehlerhafte Datenübermittlungen detektiert werden. Vor und nach dem anschließenden Acknowledge (ACK) Bit befinden sich zwei Delimiter Bits (DEL). Das Acknowledge Bit wird vom Empfänger gesetzt. Bei den Delimiter Bits handelt es sich um festgelegte Buspegel in einer Botschaft. Die abschließenden Felder End-of-Frame und Interframe Space sind ebenfalls festgelegte Buspegel, welche auf eine CAN-Botschaft folgen.[83]

Damit CAN-Botschaften in einem Netzwerk unter den Busteilnehmern ausgetauscht werden können, müssen diese definiert werden. Dementsprechend haben sich drei Formate zur Beschreibung von CAN-Nachrichten etabliert, die in der folgenden Abbildung dargestellt sind. Diese Dateien beinhalten mitunter das Mapping und das Format der Daten und eine Skalierung. Unter Mapping der Daten versteht man die Position des Startbits und die Bitlänge innerhalb der bis zu 8 Bytes an Nutzdaten einer CAN-Botschaft. Das Format der Daten gibt beispielsweise an, ob diese vorzeichenbehaftet sind oder nicht. Die Skalierung wird in der Regel als Parameter einer gebrochenen rationalen Funktion angegeben. Mittels dieser Beschreibungen können Nachrichtendefinitionen problemlos von einer Anwendung auf eine andere übergeben werden.[84]

Beschreibung	Definiert durch	Format
A2L	ASAM	ASCII-Textdatei
DBC	Vector GmbH	ASCII-Textdatei
FIBEX	ASAM	XML

Abb. 22: Formate zur Beschreibung eines CAN-Netzwerks, Quelle: Paulweber/Lebert (2014), S. 258.

2.2.5.2 Zugriffsverfahren

CAN nutzt CSMA/CR als Zugriffsverfahren.[85] Grundsätzlich können bei diesem Verfahren mehrere Busteilnehmer zeitgleich senden. Ein derartiger Konflikt wird durch die Arbitrierungsphase verhindert. In dieser Phase wird durch bitweisen Vergleich festgestellt, welche Nachricht die höchste Priorität aufweist. Dieser bitweise Vergleich beruht auf der Differenzierung der zwei spezifizierten Buspegel Low und High. Der dominante Low-Pegel ist überstimmend, der rezessive High-Pegel ist nachgebend. Gestartet wird der Arbitrierungsvorgang mit dem dominanten SOF-Bit. Darauf folgt der bitweise Vergleich der Message-Identifier der konkurrierenden Teilnehmer am Bus. Dazu vergleichen die arbitrierenden Busteilnehmer den Wert ihres gesendeten Bits mit dem aktuell am Bus befindlichen Bit. Sollte dabei ein Teilnehmer auf

[83] Vgl. Streichert/Traub (2012), S. 116f.

[84] Vgl. Paulweber/Lebert (2014), S. 258.

[85] Vgl. Zimmermann/Schmidgall (2014), S. 79.

einen Low-Pegel treffen, während er einen High-Pegel übermittelt, bricht er seinen Sendevorgang aufgrund der niedrigeren Priorität ab und geht anschließend in den Empfangsmodus. Dieses Prinzip wird auch als verlustlose Arbitrierung bezeichnet, da der Teilnehmer mit der höchsten Priorität nach Abschluss des Arbitrierungsvorgangs keinen Teil der Nachricht erneut senden muss.[86]

In der folgenden Darstellung ist ein Arbitrierungsvorgang mit drei Teilnehmern ersichtlich. Bis zum fünften Bit des Identifiers stimmen die aufgeschalteten Bits der Sender mit den Bits am Bus überein. Bei Bit 5 nimmt Teilnehmer 2 aufgrund einer Abweichung seinen Sendewunsch zurück und wechselt in den Empfangsmodus. Teilnehmer 1 folgt bei Bit 2. Somit bleibt Teilnehmer 3 übrig. Dieser hat den Identifier mit der höchsten Priorität übertragen und darf damit seine Botschaft übermitteln.

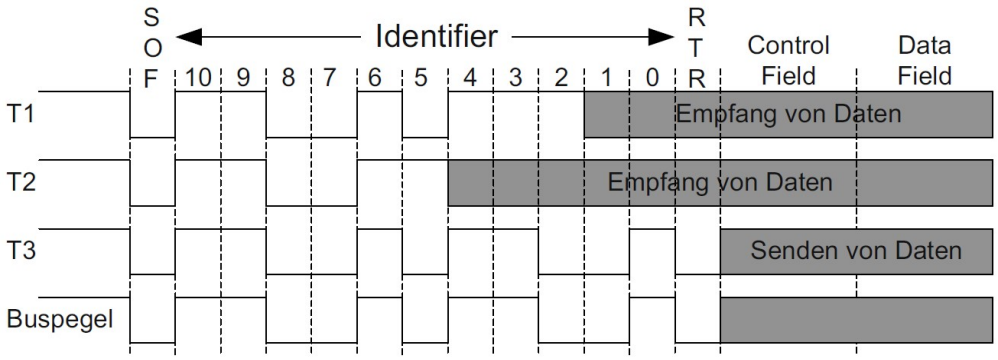


Abb. 23: Arbitrierung bei CAN-Botschaften, Quelle: Streichert/Traub (2012), S. 118.

2.2.5.3 Fehlererkennung und Fehlerbehandlung

Entsprechend der Anforderungen an die funktionale Ausfallsicherheit in der Fahrzeugtechnik, ist eine zuverlässige Detektion und Behandlung von auftretenden Fehlern erforderlich. Bedingt durch den starken Kostendruck in der Fahrzeugindustrie ist die Entwicklung geeigneter Systeme jedoch eingeschränkt. Bei CAN wird durch die Verknüpfung verschiedener Mechanismen eine hohe Fehlererkennungsrate erreicht. Die Hamming-Distanz liegt dabei bei 6.[87] In Bezug auf zwei unterschiedliche gültige Codewörter wird mit diesem Wert die Anzahl der abweichenden Bits angegeben. Bei einem vollständigen Code gibt der Hamming-Abstand das Minimum aller Hamming-Distanzen der einzelnen Codewörter, welche Teil des vollständigen Codepakets sind, an. Anhand dieser Hamming-Distanz ergibt sich die entsprechende Fehlererkennungs- bzw. Fehlerkorrekturfähigkeit. Somit können bei einer Hamming-Distanz von 6 beispielsweise fünf Bitfehler detektiert und vier Bitfehler behoben werden.[88] Das CAN-Protokoll beinhaltet fünf Methoden zur Fehlererkennung. Diese Mechanismen lauten Bitmonitoring, Überwachung des Telegramformats, Cyclic Redundancy Check (CRC), Acknowledgement Überwachung und Bit-Stuffing Überwachung. Beim Bitmonitoring prüft der Sender, ob der gewünschte Pegel am Bus ankommt. Bei der Überwachung des Telegrammformats wird von jedem Busteilnehmer geprüft, ob die auf dem Bus gesendete Nachricht etwaige Fehler enthält. Der Cyclic Redundancy Check ist hingegen aufwändiger. Dabei wird aus Botschaftsbeginn, Arbitrierungsbereich, Steuerfeld und Payload eine Checksumme durch

[86] Vgl. Reif (2014), S. 15.

[87] Vgl. Paulweber/Lebert (2014), S. 257.

[88] Vgl. Reif (2014), S. 11f.

Polynomdivision berechnet. Diese Summe wird beim Empfänger gleichermaßen berechnet und anschließend mit der übermittelten Checksumme verglichen. Bei der Acknowledgement Überwachung erwartet der Sender einer Nachricht die Bestätigung des fehlerlosen Empfangs durch das Setzen eines dominanten Bits im ACK-Feld durch den Empfänger. Ohne diese Bestätigung geht der Sender von einem Fehler bei der Übertragung aus. Bei der Bit-Stuffing Überwachung wird von allen Busteilnehmern die Einhaltung der Stuffbit-Regel geprüft.[89] Wird von einem Teilnehmer am Bus ein Fehler bei der Übermittlung durch einen dieser Mechanismen erkannt, so kann dieser eine Active-Error-Message übertragen. Bei dieser Message wird die Stuffbit-Regel verletzt, da sich diese aus sechs logischen Nullen zusammensetzt. Alle übrigen Busteilnehmer detektieren die Verletzung des Bit-Stuffings und übertragen gleichermaßen eine Error-Message. Auf diese Error-Message folgt ein Error-Delimiter. Im Anschluss beginnt die wiederholte Übermittlung der Botschaft nach dem Inter Frame Space mit dem SOF-Bit.[90]

2.2.5.4 Kommunikationstechnik und Realisierung

CAN dient der Verknüpfung verschiedener Steuergeräte in der Fahrzeugtechnik und der dementsprechenden Übertragung von Regelungs-, Steuer- und Messdaten im Echtzeitbetrieb bei hoher Zuverlässigkeit sowie Fehlersicherheit. Dabei handelt es sich um ein bitstrom-orientiertes Bussystem mit einer bidirektionalen Zweidrahtleitung. Als Topologie wird bei CAN grundsätzlich die Linientopologie gewählt. Dabei sind die Leitungslänge und die Stichleitungslänge von der gewählten Übertragungsrate abhängig. Bei 1 MBit/s ergeben sich beispielsweise maximale Längen von 40 m für die gesamte Busleitung und 30 cm für die einzelnen Stichleitungen.[91]

Wichtig dabei zu beachten ist, dass an den beiden Enden der Busleitung 120 Ohm Abschlusswiderstände angebracht werden müssen, um Signalreflexionen zu verhindern. Für den Aufbau von CAN-Netzwerken werden spezielle Systemkomponenten benötigt. Diese sind der CAN-Controller, der Host-Controller und der Bus-Transceiver. Hauptbestandteil eines CAN-Systems ist der CAN-Controller, der wichtige Funktionen wie die Busarbitrierung, die Fehlererkennung und -behandlung sowie die CRC-Berechnung und -Überprüfung zur Aufgabe hat. Zusätzlich beinhaltet der CAN-Controller Funktionen zur Nachrichtenspeicherung und -filterung sowie die automatische Generierung von Antworten auf Anfragen. In Bezug auf diese Umfänge wird in der Regel zwischen den zwei Prinzipien Basic-CAN und Full-CAN unterschieden. Der CAN-Controller übernimmt somit die Funktionen der Ebenen 1 und 2 des ISO/OSI-Schichtenmodells und ermöglicht dem übergeordneten Host-Controller den Zugriff darauf. Damit kann der Host-Controller sein Aufgabengebiet auf die Übermittlung bzw. den Empfang von Nachrichten eingrenzen. Dabei können CAN- und Host-Controller entweder getrennt (Stand-Alone-Controller) oder zusammen mit einem Mikrocontroller realisiert werden. Bei Integration von CAN- und Host-Controller, ergibt sich ein Mikrocontroller mit einer oder auch mehreren CAN-Interfaces. Der Bus-Transceiver dient den Teilnehmern zur Ankopplung an die Busleitung.[92]

---

[89] Vgl. Reif (2014), S. 18.

[90] Vgl. Streichert/Traub (2012), S. 119f.

[91] Vgl. Zimmermann/Schmidgall (2014), S. 78f.

[92] Vgl. Reif (2014), S. 19f.

2.2.6 CAN FD

CAN FD, Controller Area Network with Flexible Data Rate, ermöglicht eine nahtlose Erweiterung der klassischen CAN-Technologie.[93] Ferner ist CAN FD der Versuch, durch simple Anpassungen den eher komplizierten Wechsel zu FlexRay bei einem Großteil der Systeme vermeiden zu können.[94] Die Entwicklung von CAN FD wurde aufgrund der immer weiter steigenden Anforderungen an die Übertragungsrate in der Fahrzeugtechnik im Jahr 2011 vom Unternehmen Bosch in Zusammenarbeit mit verschiedenen Automobilherstellern und CAN-Experten gestartet, da die Datenraten des gewöhnlichen CAN-Protokolls nicht mehr ausreichend waren. CAN und CAN FD sind aktuell in der ISO-Norm 11898-1:2015 spezifiziert.[95] Vom Unternehmen Bosch wurde CAN FD erstmals im Jahr 2012 als Konzept in der Version 1.0 vorgestellt.[96] Gegenüber dem gewöhnlichen CAN-Protokoll bietet das FD-Protokoll eine höhere Datenübertragungsrate sowie mehr Nutzdaten pro Botschaft. Für die Sicherheit bei der Übermittlung von Daten wird aufgrund des vergrößerten Nutzdatenbereichs eine längere CRC-Prüfsumme als bei CAN eingesetzt.[97]

Einer der Vorteile dieses Protokolls ist jener, dass CAN FD-Interfaces in einem gewöhnlichen CAN-Netzwerk problemlos implementiert werden können, da diese abwärtskompatibel sind und CAN-Software-Applikationen bis auf die grundlegende Konfiguration nicht weiter angepasst werden müssen.[98] Die Elemente und Struktur der physikalischen Schicht, wie beispielsweise die Datenleitung oder die Endwiderstände, können dabei ebenfalls beibehalten werden.[99]

Eine Anwendung bei der CAN FD in Frage kommt, ist beispielsweise die Aktualisierung der Steuergerätesoftware im Automobilwerk oder bei einer Servicewerkstätte, bei der eine höhere Datenrate als bei CAN gewünscht ist um, aufgrund der Zeitersparnis, Kosten zu reduzieren. Während einer solchen Aktualisierung werden alle nicht CAN FD-fähigen Systemkomponenten in einen Ruhemodus gesetzt und die CAN FD-fähigen ECUs im Bereich zwischen dem Arbitrierungsfeld und dem CRC-Feld auf eine höhere Bitrate gestellt. Dazu können pro Botschaft bis zu 64 Byte an Nutzdaten übermittelt werden. Bei CAN können hingegen nur maximal 8 Byte an Nutzdaten pro Nachricht gesendet werden. Nach Abschluss des Aktualisierungsvorgangs können die CAN FD-Teilnehmer wieder auf die CAN-Rate gestellt und die übrigen CAN-Teilnehmer in Betrieb genommen werden, damit die Kommunikation unter allen Busteilnehmern wiederhergestellt werden kann.[100]

---

[93] Vgl. Lindenkreuz (2012), Online-Quelle [29.April.2016], S. 21.

[94] Vgl. Zimmermann/Schmidgall (2014), S. 11.

[95] Vgl. CAN in Automation (2016), Online-Quelle [30.April.2016].

[96] Vgl. Zimmermann/Schmidgall (2014), S. 77.

[97] Vgl. Zeltwanger (2014), Online-Quelle [29.April.2016].

[98] Vgl. Hartwich (2012), Online-Quelle [29.April.2016], S. 8.

[99] Vgl. Lindenkreuz (2012), Online-Quelle [29.April.2016], S. 21.

[100] Vgl. Borgeest (2014), S. 119.

2.2.6.1 Botschaftsaufbau

Die Länge des Datenfeldes einer CAN-Botschaft wird also von maximal 8 Byte auf 64 Byte erweitert. Damit können CAN FD-Nachrichten wie bislang bei CAN 0 bis 8 Byte bzw. auch 12, 16, 20, 24, 32, 48 oder 64 Byte Nutzdaten beinhalten. Aufgrund dessen, dass die Längenangabe (DLC-Feld) des Nutzdatenbereichs innerhalb der Control Bits im Kopf der Botschaft schon bei CAN 4 Bit umfasst hat, müssen dazu bloß die bislang nicht benutzten Codes verwendet werden. Damit zwischen CAN- und CAN FD-Nachrichten unterschieden werden kann, wurde das Botschaftsformat um das Extended Data Length-Bit (EDL) erweitert.[101] Dabei signalisieren die Bitzustände rezessiv, dass es sich um eine CAN FD-Botschaft handelt und dominant, dass eine Standard CAN-Nachricht übertragen wird.[102] Zusätzlich wurde die CRC-Prüfsumme angepasst, damit bei längeren Nachrichten dieselbe Datensicherheit gewährleistet werden kann. Nachrichten mit bis zu 16 Byte Payload verwenden demnach anstatt der 15 Bit bei CAN einen 17 Bit CRC. Ab 16 Byte Nutzdaten wird die Checksumme auf 21 Bit erhöht.[103] Ein weiteres neues Bit gegenüber einer CAN Botschaft ist das Markierungsbit Bit Rate Switch (BRS). Dieses Bit legt fest, ob die Bitrate verändert werden soll. Des Weiteren gibt es bei CAN FD ein Error State Indicator-Bit (ESI). Mit diesem wird signalisiert, ob ein Teilnehmer Error Active oder Error Passive ist.[104]

Die folgende Abbildung zeigt den möglichen Aufbau einer CAN FD-Botschaft. Dazu sind die drei Bereiche Header, Payload und Trailer mit deren Inhalten ersichtlich. Zusätzlich sind die Bereiche der normalen und der veränderten Bitrate dargestellt. Das Bit-Stuffing wurde bei den Längenangaben der einzelnen Felder nicht berücksichtigt.

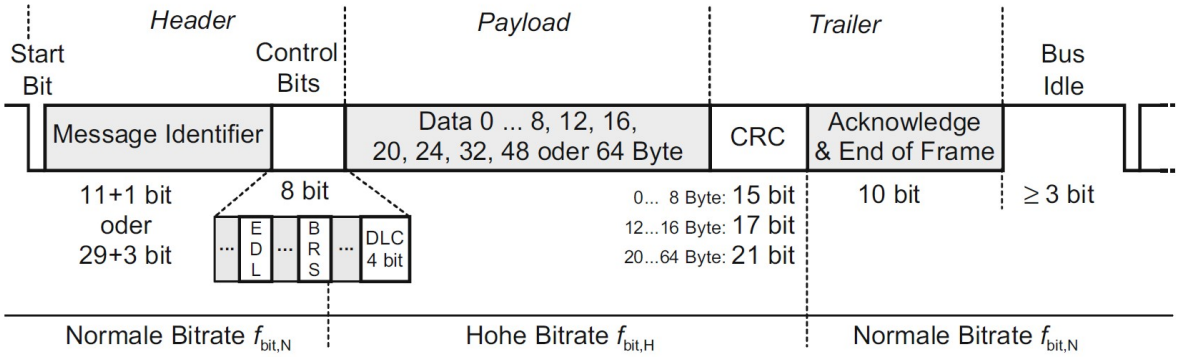


Abb. 24: Aufbau einer CAN FD-Botschaft, Quelle: Zimmermann/Schmidgall (2014), S. 77.

In der Darstellung sind die Bits r0, r1, ESI und IDE nicht angeführt. Darüber hinaus beinhaltet ein CAN FD-Frame neun Control Bits. Das r1 Bit wird im Anschluss an den Message Identifier übertragen und befindet sich im Arbitrierungsfeld. Danach folgt zu Beginn der Control Bits das Identifier Extension-Bit (IDE), das EDL-Bit und das r0-Bit. Auf das anschließende BRS-Bit folgt schlussendlich der Error State Indicator (ESI) bevor das DLC-Feld beginnt. Diese Reihenfolge gilt für eine Botschaft mit einem 11 Bit

[101] Vgl. Zimmermann/Schmidgall (2014), S. 77.

[102] Vgl. Lindenkreuz (2012), Online-Quelle [29.April.2016], S. 6.

[103] Vgl. Mutter/Hartwich (2015), Online-Quelle [4.August.2016], S. 2.

[104] Vgl. Borgeest (2014), S. 119.

Identifier. Bei einem 29 Bit Identifier befindet sich das IDE-Bit nach dem 11 Bit Identifier und dem Substitute Remote Request-Bit (SRR). Die restlichen Bits folgen nach der anschließenden 18 Bit Identifier Extension. Die beiden Bits r0 und r1 sind für zukünftige Protokollvarianten reserviert und werden dominant übertragen. Beim Error State Indicator wird zwischen dominant für Error Active und rezessiv für Error Passive unterschieden. Auch beim BRS-Bit wird zwischen dominant und rezessiv unterschieden. Dominant bedeutet, dass mit normaler Bitrate, sprich mit der gleichen Rate wie in der Arbitrierungsphase, gearbeitet wird und rezessiv, dass mit veränderter bzw. erhöhter Bitrate in der Datenphase übertragen wird.[105]

2.2.6.2 Zugriffsverfahren

CAN FD ist eine Weiterentwicklung des klassischen CAN-Busses. Dabei wurde das CAN-Protokoll in den zwei Bereichen Bitrate und Nutzdatengröße angepasst. Nun werden Datenraten größer 1 MBit/s und Nutzdaten größer 8 Byte unterstützt. Umgesetzt wird dies durch ein neues Botschaftsformat. Dieses Format ermöglicht das Umschalten der Bitrate und nutzt das neu eingeführte Extended Data Length Bit, um bis zu 64 Byte an Nutzdaten übermitteln zu können. Das bedeutet, dass sich bei CAN FD im Vergleich zu CAN lediglich das Botschaftsformat verändert hat. Im Speziellen wurden Bits im Kontrollfeld zum Aktivieren der erwähnten Optionen und der neuen CRC Sequenz eingeführt. Dementsprechend wurde bei der Entwicklung darauf geachtet, die Hauptbestandteile von CAN beizubehalten. Aus diesem Grund sind CAN FD-Controller abwärtskompatibel, da auch das bei CAN eingesetzte Zugriffsverfahren (CSMA/CR mit Arbitrierung) weiterhin verwendet wird.[106]

2.2.6.3 Fehlererkennung und Fehlerbehandlung

Aufgrund des größeren Nutzdatenbereich und dem Wunsch die hohe Fehlererkennung des CAN-Protokolls beizubehalten, musste der CRC bei CAN FD angepasst werden. So wird die Prüfsumme entsprechend der Nutzdatengröße von 15 Bit auf bis zu 21 Bit erhöht. Damit kann garantiert werden, dass weiterhin fünf Bitfehler erkannt werden.[107]

Insgesamt besitzt das CAN FD-Protokoll zusätzlich zu den im CAN-Protokoll integrierten Mechanismen vier neue oder verbesserte Mechanismen zur Fehlererkennung bzw. -behandlung. Die erste Anpassung umfasst die bereits erwähnte Erweiterung der CRC-Prüfsumme (1). Die drei neuen Mechanismen in der CAN FD-Spezifikation werden wie folgt bezeichnet: Inkludierung der dynamischen Stuffbits in der CRC-Berechnung (2), Einführung einer Stuffbit-Zählung (3) und Einführung fester Stuffbits im CRC-Feld (4). Die Erweiterung der CRC-Prüfsumme wurde aufgrund der Vergrößerung des Nutzdatenfeldes und der Inkludierung der dynamischen Stuffbits in der CRC-Berechnung notwendig. Diese nun inkludierten dynamischen Stuffbits, die sich vor dem CRC-Feld befinden, wurden bei CAN nicht in die CRC-Berechnung miteinbezogen. Zusätzlich wurden feste Stuffbits im CRC-Feld eingeführt. Dabei werden Stuffbits an definierten bzw. fixen Positionen eingefügt. Das erste feste Stuffbit wird am Beginn des CRC-Feldes und die nächsten nach jedem vierten Bit von diesem Feld gesetzt. Der Wert jedes dieser Stuffbits

[105] Vgl. Hartwich (2012), Online-Quelle [29.April.2016], S. 3f.

[106] Vgl. Hartwich (2012), Online-Quelle [29.April.2016], S. 1.

[107] Vgl. Zeltwanger (2014), Online-Quelle [29.April.2016].

hat den umgekehrten Wert als das vorangegangene Bit. Das garantiert, dass keine Sequenz von sechs aufeinanderfolgenden gleichen Bits auftreten kann, damit die Stuffbit-Regel nicht verletzt wird. Nach dem Empfang der Nachricht überprüft der Empfänger die festen Stuffbits. Sollte ein Stuffbit fehlerhaft (falscher Wert) sein, meldet der Empfänger einen Fehler. Bei der Stuffbit-Zählung werden die dynamischen Stuffbits der Nachricht vom Sender gezählt und mit einem Paritätsbit am Beginn des CRC-Feldes übermittelt. Die im CRC-Feld eingefügten festen Stuffbits werden dabei nicht mitgezählt. Nach der Übermittlung der Daten führen die Empfänger ebenfalls eine Stuffbit-Zählung durch und vergleichen diese mit dem übermittelten Wert. Stimmen die Werte dabei nicht überein, gibt der entsprechende Empfänger einen Fehler aus. Diese Stuffbit-Zählung wurde im Zuge der Normierung durch die ISO eingeführt. Die Mechanismen 2 und 3 ermöglichen die Fehlererkennung von Datenbits, welche bei einem Fehler in Stuffbits durch Bit-Flips umgewandelt wurden und umgekehrt. Der Mechanismus 4 erweitert die Fähigkeit zur Überprüfung des Botschaftsformats. Zusätzlich reduzieren die Mechanismen 1, 3 und 4 die Wahrscheinlichkeit, dass eine zufällige Bitsequenz mit einem erwarteten CRC-Feld in verschiedenen Größenordnungen übereinstimmt. Die Fehlerbehandlung nach einer Fehlererkennung ist bei CAN FD mit der von CAN ident. Bei Erkennung eines Fehlers sendet der entsprechende Empfänger eine Error-Message aus. Diese zerstört die aktuelle Nachricht am Bus und der Sender übermittelt anschließend seine Botschaft erneut. Fehlt das Acknowledgement eines Empfängers, das den fehlerlosen Erhalt der Nachricht signalisiert, wird die Botschaft ebenfalls erneut vom entsprechenden Busteilnehmer übertragen.[108]

2.2.6.4 Kommunikationstechnik und Realisierung

Bei der Realisierung unterscheidet sich ein CAN FD-Netzwerk von einem CAN-Netzwerk nur durch den Einsatz von CAN FD-Controllern. Die übrigen Teile und Bausteine der physikalischen Schicht können beibehalten werden. CAN FD-Controller sind bereits am Markt verfügbar und können demnach erworben werden.[109]

In der Kommunikationstechnik unterscheiden sich die Protokolle ebenfalls. Die Datenrate bei CAN ist mit 1 MBit/s begrenzt. Dies ergibt sich durch die bitweise Arbitrierung des Identifiers zur Kollisionserkennung im Header und dem Acknowledge Bit zur Empfangsbestätigung im Trailer. Die Auswertung dieser genannten Bits muss bei allen Busteilnehmern garantiert im Zeitraum desselben Bittaktes durchgeführt werden. Da die Signallaufzeit am Bus nicht verändert werden kann, entsteht daher eine direkte Abhängigkeit zwischen Busleitungslänge und minimaler Bitdauer sowie maximaler Bitrate. Diese Anforderung gilt hingegen nicht für den Nutzdatenbereich und die CRC-Checksumme. Aufgrund dessen kann die Bitrate bei CAN FD mittels BRS-Bit in diesem Teil der Nachricht erhöht werden. Dabei sollen Raten um bis zu 4 MBit/s möglich sein. In Folge dessen ergeben sich für CAN FD erheblich höhere Nutzdatenraten als bei CAN.[110]

[108] Vgl. Mutter/Hartwich (2015), Online-Quelle [4.August.2016], S. 3ff.

[109] Vgl. Lindenkreuz (2012), Online-Quelle [29.April.2016], S. 13ff.

[110] Vgl. Zimmermann/Schmidgall (2014), S. 77f.

2.2.7 FlexRay

FlexRay ist ein Kommunikationssystem und zählt zu den gängigen Bussystemen in der Fahrzeugtechnik. Die Entwicklung von FlexRay wurde im Zuge einer Arbeitsgemeinschaft verschiedener Automobilhersteller, Zulieferer und Halbleiterhersteller erarbeitet. Der Grund dafür lag im Bedarf nach einer Kommunikationstechnologie für sicherheitskritische X-By-Wire-Applikationen. Bei derartigen Applikationen soll die Fahrzeugsteuerung elektrisch oder elektronisch durchgeführt werden. Dabei ist keine mechanische Rückfallebene vorhanden. Dementsprechend besitzt FlexRay eine große Anzahl an Mechanismen, welche eine deterministische und ausfallsicherere Datenübertragung gewährleisten. In der Automobiltechnik wird FlexRay vermehrt in Bereichen eingesetzt, in denen eine hohe Datenrate benötigt wird. Dabei ermöglicht das FlexRay-Protokoll einen ein- oder zweikanaligen Betrieb in Stern- oder Linientopologie bzw. in der Kombination dieser beiden Topologien bei einer maximalen Bitrate von 10 MBit/s.[111] Bei einem zweikanaligen Betrieb besteht die Möglichkeit entweder Nachrichten redundant zu übermitteln, beispielsweise in sicherheitskritischen Anwendungen, oder die Datenübertragungsrate auf 20 MBit/s zu verdoppeln. Als Übertragungsmedium sind gewöhnliche Kupferleitungen mit differenzieller Übermittlung und alternativ optische Datenleitungen spezifiziert.[112] Die internationale Standardisierung von FlexRay lautet ISO 17458.[113]

Das Ziel von FlexRay ist, ein fehlertolerantes und deterministisches Kommunikationssystem mit hohen Bitraten zu liefern, das dabei flexibel im Applikationsbereich und in Bezug auf die Erweiterungsmöglichkeiten ist. Die Anwendungsbereiche in der Fahrzeugtechnik liegen dabei zwischen aktiven (ohne Rückfallebene) und passiven Sicherheitssystemen sowie Karosserie- und Komfortelektronik. Des Weiteren wird FlexRay vermehrt in Antriebsstrangkomponenten eingesetzt. Um diese verschiedenen Anwendungsgebiete unterstützen zu können, werden bei FlexRay-Systemen zwei Prinzipien des Buszugriffs eingesetzt. Bei Anwendungen, die ein deterministisches Kommunikationsverhalten erfordern wird ein zeitgesteuerter Buszugriff gewählt. Bei weniger zeitkritischen Applikationen wird eine optimale Ausnutzung der vorhandenen Übertragungskapazität angestrebt. Damit diese genannten Verfahren verknüpft werden können, wird die Kommunikation bei FlexRay in Zyklen durchgeführt.[114]

2.2.7.1 Botschaftsaufbau

Das Botschaftsformat bei FlexRay besteht, wie auch bei CAN, aus den drei Teilen Header, Payload und Trailer. Dabei startet jede Nachricht mit einem Header, der mehrere Steuerbits, die Frame ID, die Nutzdatenlänge und den Zykluszähler beinhaltet. Die Steuerbits lauten Reserved-Bit, Payload Preamble Indicator-Bit, Null Frame Indicator-Bit, Sync Frame Indicator-Bit und Startup Frame Indicator-Bit. Das Reserved-Bit wird dabei immer mit dem Wert 0 übertragen. Die restlichen Steuerbits dienen zur Anzeige von Sonderbotschaften. Das Payload Preamble Indicator-Bit gibt im statischen Teil eines

[111] Vgl. Streichert/Traub (2012), S. 121.

[112] Vgl. Borgeest (2014), S. 123f.

[113] Vgl. Zimmermann/Schmidgall (2014), S. 96.

[114] Vgl. Reif (Hrsg.) (2011), S. 132f.

Kommunikationszyklus an, dass die ersten 0 bis 12 Datenbytes Zusatzinformationen für das Netzwerkmanagement beinhalten und im dynamischen Teil, dass die ersten beiden Bytes des Datenfeldes eine Message ID tragen. Dies ist eine Kennung, welche die restlichen Daten der Nachricht kennzeichnet und die mit dem Message Identifier bei CAN vergleichbar ist. Anhand dieser Kennung kann der Empfänger eine Akzeptanzfilterung durchführen. Das darauffolgende Null Frame Indicator-Bit dient zur Anzeige eines Senders, dass die übermittelte Nachricht keine Nutzdaten beinhaltet. Diese Anzeige wird benötigt, da im statischen Teil des Kommunikationszyklus alle Nachrichten stets dieselbe Anzahl an Nutzdatenwörtern tragen müssen. Dabei ist es bedeutungslos ob die Daten gültig oder ungültig sind. Nach diesem Bit werden das Startup Frame Indicator- und das Sync Frame Indicator-Bit übertragen. Ersteres dient zur Synchronisation der gesamten Teilnehmer am Bus bei Netzwerkstart. Letzteres wird zur Synchronisation während des Betriebs genutzt. Im Anschluss an die genannten Steuerbits folgen das Frame ID-Feld mit der Nummer des Zeitslots in dem die Nachricht übermittelt wird und die Payloadlänge als Anzahl von 16 Bit-Datenworten. Die Daten können dazu willkürlich in Bytes unterteilt werden, müssen aber als gerade Anzahl von Nutzdatenbytes übertragen werden. Am Ende des Header-Bereichs befindet sich die Nummer des aktuellen Kommunikationszyklus, auch Zykluszähler genannt. Dieser wird bei der Initialisierung des Netzwerks mit 0 gestartet und anschließend nach jedem Kommunikationszyklus erhöht. Davor befindet sich das Header CRC-Feld, in dem sich die Checksumme des CRC vom Header ohne Zykluszähler zur Überprüfung von Übertragungsfehlern befindet. Auf den 5 Byte großen Header folgt der Payloadbereich mit maximal 254 Byte Nutzdaten und der Trailer mit einer Größe von 3 Byte, in dem sich eine 24 Bit-CRC-Prüfsumme der Nutzdaten befindet. Innerhalb der Bitübertragungsschicht kommen noch, ähnliche wie bei CAN-Nachrichten, weitere Steuerbits und Bitfolgen hinzu. Aufgrund dessen wird die Nutzdatenrate um ca. 20 % reduziert.[115]

In der folgenden Darstellung ist das FlexRay-Botschaftsformat ersichtlich. Dazu sind die beschriebenen Teilbereiche und deren Inhalte dargestellt. Des Weiteren sind die Größen sowie die Bezeichnungen der erläuterten Felder angeführt.

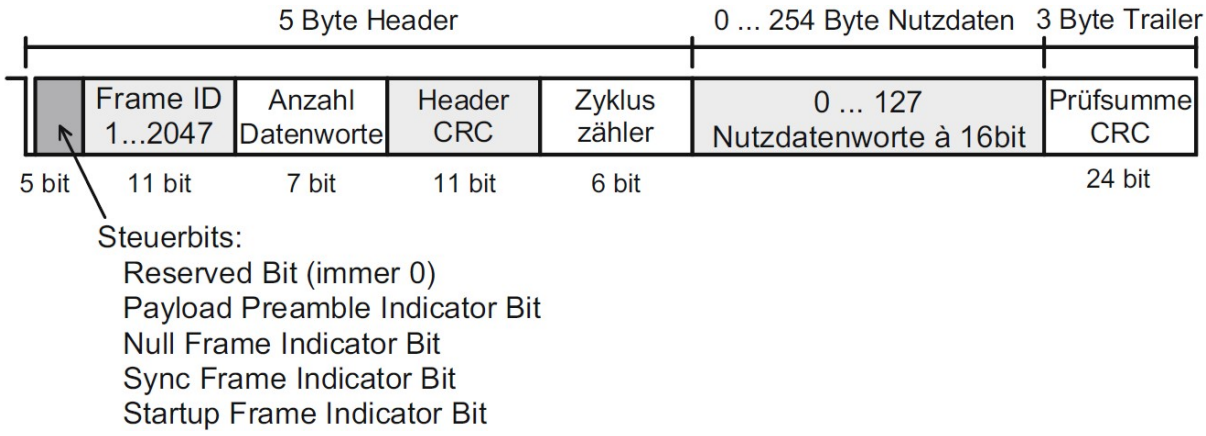


Abb. 25: Aufbau einer FlexRay-Botschaft, Quelle: Zimmermann/Schmidgall (2014), S. 101.

[115] Vgl. Zimmermann/Schmidgall (2014), S. 101f.

2.2.7.2 Zugriffsverfahren

Ein FlexRay-Kommunikationszyklus setzt sich aus einem statischen Teil mit fixen Zeitbereichen, einem optionalen dynamischen Teil, der prioritätsgesteuert ist, einem optionalen Symbol Window und einer Network Idle Time (NIT) zusammen. Die NIT dient als Ruhepause zwischen dem Ende eines Zyklus und dem Beginn des nächsten Zyklus.[116] Das Symbol Window kann zur Übermittlung eines Collision Avoidance (CAS) bzw. Media Access Test (MTS) Symbols, mit dem beispielsweise der Bus Guardian geprüft werden kann, verwendet werden.[117] Der Buszugriff im statischen Teil wird mittels TDMA abgehandelt. Entsprechend diesem Verfahren ist jeder Nachricht ein definierter Zeitbereich und damit ein fixer Sendezeitpunkt zugewiesen. Demzufolge wird eine deterministische Datenübermittlung gewährleistet. Dagegen wird im dynamischen Teil das FTDMA-Verfahren, das unter anderem auch die Bezeichnung Mini-Slotting trägt, für den Buszugriff genutzt. Dieser Bereich ist für Nachrichten mit niedrigen Ansprüchen an eine vorhersagbare Latenzzeit ausgelegt.[118] Primär dient der dynamische Teil der ereignisgesteuerten Übermittlung von Daten. Während des Betriebs zählt jeder Teilnehmer am Bus die Anzahl an Kommunikationszyklen mit. Der Zähler dazu nennt sich Cycle Counter, auch als Zykluszähler bezeichnet, und startet bei 0. Sämtliche Zeitbereiche eines Zyklus sind ganzzahlige Vielfache der so genannten Makroticks. Diese Makroticks sind ein für das komplette Netzwerk und beide FlexRay-Kanäle identischer Zeitraster. Die Dauer der Makroticks soll zwischen 1 und 6 µs liegen.[119] Die Zeitschlitze gliedern sich demzufolge in Makroticks, die durch eine Uhrensynchronisation, auch Uhrenvergleich genannt, als Zeiteinheit im gesamten Bussystem gelten. Die Makroticks sind dabei ein Vielfaches der Mikroticks. Diese Mikroticks werden aus dem internen Oszillatortakt von jedem Busteilnehmer abgeleitet. Die Summe der Mikroticks je Makrotick unterscheidet sich zwischen allen Teilnehmern und wird daher durch eine regelmäßige Uhrensynchronisation geprüft und angepasst. Die Zeitkorrektur im Zuge der Uhrensynchronisation besteht aus zwei Funktionen, der Offsetkorrektur während der NIT und der Geschwindigkeitsanpassung, in welcher das Verhältnis von Makroticks zu Mikroticks korrigiert wird. Davor muss allerdings von jedem Busteilnehmer bestimmt werden, ob seine eigene Uhr im selben Takt der übrigen Teilnehmer läuft. Nachdem die Botschaften im statischen Teil zu fixen Zeitpunkten übermittelt werden, errechnen sich die Teilnehmer durch Vergleich der Ist- und Sollzeit der empfangenen Nachricht eine Zeitabweichung zu den übrigen Busteilnehmern. Die relative Uhrengeschwindigkeit kann dabei ebenfalls ermittelt werden.[120]

Der statische Teil besteht aus Slots, auch Zeitschlitze genannt, die so groß sind, dass innerhalb eines dieser Slots eine komplette Nachricht übermittelt werden kann. Diese Slots, auch als statische Slots bezeichnet, haben dabei alle die gleiche definierte Größe und laufen auf beiden FlexRay-Kanälen synchron. Das Zugriffsrecht und damit die Sendeberechtigung innerhalb eines Slots ist einem einzigen Busteilnehmer zugeordnet, damit zu keinem Zeitpunkt eine etwaige Kollision auftreten kann.

[116] Vgl. Borgeest (2014), S. 124.

[117] Vgl. Zimmermann/Schmidgall (2014), S. 101.

[118] Vgl. Reif (2014), S. 25.

[119] Vgl. Zimmermann/Schmidgall (2014), S. 99.

[120] Vgl. Borgeest (2014), S. 124f.

Vorausgesetzt dafür ist ein fehlerfreier Betrieb aller Teilnehmer am Bus. Das Senderecht kann bei redundanten Übertragungen oder zur Erhöhung der Datenrate auf beiden Kanälen im selben Slot an einen Busteilnehmer vergeben werden. Alle Busteilnehmer zählen für beide Kanäle getrennt mittels eines Slot Counters die Zeitschlitze während eines Zyklus mit. Der Counter startet dabei am Beginn jedes Kommunikationszyklus bei 1. Der Wert dieses Zählers dient der Anzeige im statischen Teil, welcher Busteilnehmer zurzeit das Senderrecht besitzt. Die Länge des statischen Segments ist mit minimal zwei und maximal 1023 Slots definiert. Der dynamische Teil setzt sich ebenso aus Zeitschlitzen zusammen. Diese werden Minislots genannt und sind erheblich kleiner als die statischen Slots. Im Zeitraum eines Minislots bekommt lediglich ein Busteilnehmer auf jeweils einem Kanal unabhängig voneinander die Sendeberechtigung. Die zu übermittelnde Nachricht darf dabei eine variable und auf beiden Kanälen unterschiedliche Länge haben. Die Gesamtlänge des dynamischen Teils darf dabei jedoch nicht überschritten werden. Nach Abschluss des Sendevorgangs einer Nachricht, wird der Slot Counter mit dem folgenden Minislot erhöht. Dies geschieht auch wenn keine Nachricht übertragen wird. Parallel wird die Sendeberechtigung auf den nächsten Busteilnehmer übergeben. Die Slot Counter nehmen aufgrund der variablen Anzahl und Länge der Nachrichten während eines Zyklus unterschiedliche Werte an und verlaufen in beiden Kanälen während des dynamischen Teils asynchron. Neben der Sendeberechtigung definiert der Slot Counter die Priorität der Nachrichten. Demnach wird die Nachricht mit dem niedrigsten Slotzähler-Wert zuerst übertragen. Im Anschluss folgt die Botschaft mit der nächst höheren Priorität. Im Zuge dieses Vorgangs ist es möglich, dass der dynamische Teil bereits vollständig ausgeschöpft ist und Botschaften mit höherem Slot Counter-Wert auf die nachfolgenden Zyklen warten müssen. Die maximale Summe der Zeitslots für beide Teile eines Zyklus ist mit 2047 definiert.[121]

Inwiefern sich ein Kommunikationszyklus aus den beschriebenen Teilen zusammensetzt, ist in der folgenden Darstellung ersichtlich.

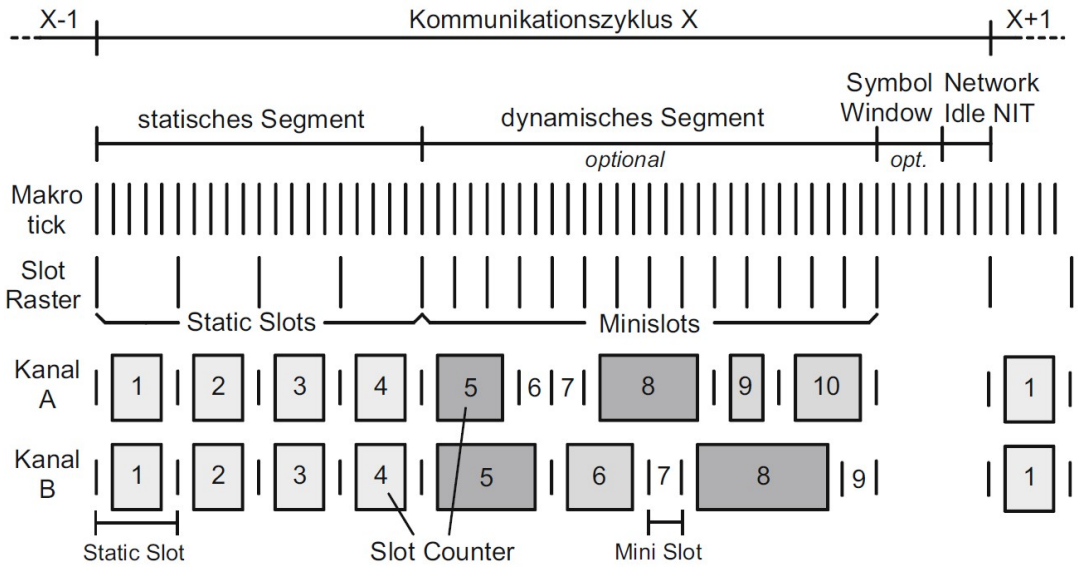


Abb. 26: Aufbau eines FlexRay-Zyklus, Quelle: Zimmermann/Schmidgall (2014), S. 100.

[121] Vgl. Zimmermann/Schmidgall (2014), S. 99f.

2.2.7.3 Fehlererkennung und Fehlerbehandlung

Durch den Einsatz der Cycle bzw. Slot Counter, der Zeitüberwachung und den beiden CRC-Summen ist ein FlexRay-Kommunikationscontroller in der Lage eine hohe Anzahl an Sendefehlerarten zu erkennen. Entsprechend der Komplexität des Fehlers gibt es zwei Möglichkeiten inwiefern der Controller reagiert. Entweder wechselt dieser in einen passiven Modus oder stoppt den Betrieb zur Gänze. Im passiven Modus können Daten nur mehr empfangen jedoch nicht mehr gesendet werden. Der Empfang dient dem Versuch sich neu zu synchronisieren. In beiden Fällen gibt der Controller den Fehler der darüber liegenden Software-Ebene weiter. Dabei wird, im Gegensatz zu CAN, keine automatische Sendewiederholung durch den Kommunikationscontroller durchgeführt. Dies ist aufgrund der Sicherstellung eines deterministischen Systembetriebs erforderlich. Zusätzlich zu den genannten Methoden zur Fehlererkennung gibt es die Möglichkeit einen Bus Guardian, auch Buswächter genannt, zu verwenden, der Probleme aufgrund eines fehlerhaften Busteilnehmers verhindern kann. Dieser Wächter ist mit einem einfachen Kommunikationscontroller vergleichbar. Demnach kann der Bus Guardian keine Nachrichten übertragen und ist dahingehend parametriert, dass er den zeitlichen Verlauf eines Zyklus kennt und dem Kommunikationscontroller bzw. Bustreiber lediglich in den Zeitrahmen eine Sendefreigabe erteilt, in denen der entsprechende Busteilnehmer seine Daten übermitteln darf.[122] Erkennt der Buswächter einen Fehler in Bezug auf den zeitlichen Verlauf, kann er den entsprechenden Teilnehmer deaktivieren. Angeordnet wird der Bus Guardian entweder integriert in jeden Busteilnehmer oder als zentraler Wächter in einem aktiven Sternpunkt des FlexRay-Bussystems.[123]

2.2.7.4 Kommunikationstechnik und Realisierung

FlexRay ist ein bitstrom-orientiertes Bussystem mit einer bidirektionalen Zweidrahtleitung ausgeführt in Linien- oder Sterntopologie und wird in der Fahrzeugtechnik zur Kommunikation unter Steuergeräten verwendet. Das Bussystem dient dabei dem Austausch von Regler-, Mess-, und Steuerdaten mit niedrigem Fehlerrisiko. In jedem FlexRay-Netzwerk können maximal 64 Busteilnehmer eingebunden werden. Für den Betrieb werden ein FlexRay-Controller, ein Transceiver, ein Mikroprozessor und optional ein Bus Guardian benötigt. Die Länge der Datenleitung bis zum Sternpunkt ist mit 24 Meter begrenzt. Als Datenrate sind grundsätzlich 2, 5 und 10 MBit/s einstellbar. Bei 10 MBit/s ergibt sich eine nutzbare Datenrate von etwa 1 MBit/s je Kanal. Die Übertragung der Daten wird durch zwei CRC-Prüfsummen abgesichert. Eine automatische Sendewiederholung bei Fehlern gibt es allerdings nicht.[124]

Zusammengefasst ermöglicht FlexRay eine deterministische und kollisionsfreie Datenübertragung mit einer Datenrate von bis zu 10 MBit/s im einkanaligen Betrieb. Im zweikanaligen Betrieb kann die Bitrate verdoppelt oder eine redundante Übertragung eingerichtet werden.[125]

---

[122] Vgl. Zimmermann/Schmidgall (2014), S. 105f.

[123] Vgl. Borgeest (2014), S. 126.

[124] Vgl. Zimmermann/Schmidgall (2014), S. 118.

[125] Vgl. Reif (Hrsg.) (2011), S. 143.

2.3 Steuergeräte

Die Aufgaben eines Steuergeräts, auch Electronic Control Unit (ECU) genannt, liegen in der Verarbeitung bzw. Weitergabe von Sensordaten und in der daraus resultierenden sowie von der Situation abhängigen Steuerung von Aktoren. Dies kann über entsprechende Regelungsalgorithmen gehandhabt werden. Demzufolge sind Schnittstellenschaltungen in einer ECU erforderlich, welche auf die entsprechenden Aktoren und Sensoren parametriert sind. Neben diesen Schnittstellenschaltungen benötigt ein Steuergerät zumindest eine Kommunikationsschnittstelle, beispielsweise einen CAN-Transceiver, um eine Verbindung mit anderen Steuergeräten sowie Diagnosegeräten aufbauen zu können und eine integrierte Infrastruktur, welche der Spannungsversorgung und den digitalen Taktsignalen dient. Der Hauptbestandteil eines Steuergeräts ist jedoch der Rechenkern. Ein solcher Rechenkern ist mittlerweile in jedem Steuergerät zu finden, kann sich allerdings je nach Einsatzgebiet in dessen Leistungsfähigkeit erheblich unterscheiden. Die grundlegenden Bereiche bzw. Aufgaben eines Steuergeräts sind also die Auswertung von Sensorsignalen, der Rechenkern, die Ansteuerung von Aktoren, ein Kommunikationscontroller und eine entsprechende Infrastruktur inklusive einer Versorgungseinheit.[126]

Betrachtet man den Aufbau eines Motorsteuergeräts so kann man diesen in die drei Gruppen Eingänge, Signalverarbeitung und Ausgänge gliedern. Anhand dieser Komponenten wird gemäß den aufbereiteten Sensordaten und den Anforderungen des aktuellen Motorbetriebs die Steuerung der Aktoren durchgeführt. Die Eingangssignale können je nach Sensor zwischen digital und analog variieren. Analogsignale müssen jedoch vor der weiteren Verarbeitung in einem A/D-Wandler digitalisiert werden. Eingehende Pulssignale müssen ebenso digitalisiert werden. Die Hauptaufgabe der Signalverarbeitung übernimmt ein Mikrocontroller. Dieser bildet den Kern eines Motorsteuergeräts und beinhaltet eine Recheneinheit, Speicherelemente für die Software und verschiedene Schnittstellen. Die Regelungs- bzw. Steuerungsalgorithmen werden im Binärformat in der Speichereinheit abgelegt. Als Speicher dient zu meist ein Flash-EPROM-Chip (Erasable Programmable Read Only Memory). Dementsprechend ist das Ändern des Programmcodes und der Parametersätze ohne jeglichen mechanischen Eingriff möglich. Durch den hohen Bedarf an Daten und Funktionen in Steuergeräten werden zusätzlich zur integrierten Speichereinheit, Komponenten, die Programmteile und Daten aufnehmen können, eingesetzt. Während der Signalverarbeitung ist ein Speichersystem mit Lese- und Schreibmöglichkeiten notwendig. Dazu wird ein RAM-Chip (Random Access Memory) eingesetzt. Für Daten, die auch nach dem Ausschalten der Zündspannung vorhanden sein müssen, wird auf einen EEPROM-Chip (Electrical Erasable Programmable Read Only Memory) zurückgegriffen. Zu den zentralen Schnittstellen, die im Mikrocontroller implementiert werden, gehören entsprechende Kommunikationsschnittstellen (z.B. CAN) um eine Verbindung zu anderen ECUs oder Diagnosesystemen herstellen zu können. Nach der Signalverarbeitung werden entsprechend der Ergebnisse der Steuerungs- und Regelungskomponenten die Aktoren über die Ausgangseinheiten des Motorsteuergeräts betätigt. Die Ausgabe an den jeweiligen Anschlüssen wird nach verschiedenen Methoden, welche sich nach der Aktorenart richten, durchgeführt. Die Aufgabe einer Motor-ECU ist beispielsweise die bedarfsgerechte Auslösung der Einspritzmenge.[127]

[126] Vgl. Borgeest (2014), S. 135f.

[127] Vgl. Reif (2014), S. 136f.

3 ASAM STANDARDISIERUNGEN

ASAM Standards sind in der Automobilbranche weit verbreitet und werden folglich auch in Prüfstandsanlagen für Fahrzeugkomponenten eingesetzt. Im Umfang dieses Kapitels werden zu Beginn die Organisation selbst und anschließend die gängigsten bzw. für diese Arbeit relevanten Standardisierungen erläutert. Die Erarbeitung dieser Standards dient als Grundlage für die zwei nachkommenden Kapitel, in denen die Verwendung und der Einsatz dieser ASAM Spezifikationen am Prüfstand behandelt wird.

3.1 ASAM

Die Association for Standardization of Automation and Measuring Systems, kurz ASAM, ist eine Vereinigung von internationalen Fahrzeugherstellern und deren Zulieferer. Anfänglich wurde die Organisation als ASAP, Arbeitskreis zur Standardisierung von Applikationssystemen, bezeichnet. Ziel dieses Verbands ist es, die Anwendung und Testreihen elektronischer Systemkomponenten (z.B. Getriebe-ECUs) im Laufe der Entwicklung und Fertigung von Kraftfahrzeugen zu vereinfachen. Während der Entwicklung von Fahrzeugkomponenten werden bei unterschiedlichen Testläufen sowie in Prüflaboren Messdaten aus den Steuergeräten aufgenommen, ausgewertet und die internen Parameter des Steuergeräts entsprechend angeglichen. Diese Vorgänge werden auch als Measure und Calibrate bezeichnet. Nach Abschluss der Fahrzeugentwicklung wird in der Produktionsphase der einwandfreie Verbau und die einwandfreie Funktionsweise der Systemkomponenten auf Fertigungsprüfständen kontrolliert. Wird dabei ein Problem festgestellt, kann dies oft per Nachjustierung behoben werden. In beiden Bereichen werden Messwerte ermittelt (Measure), Diagnosedaten verarbeitet (Diagnose) und steuergerätinterne Parameter verändert (Calibrate). Die Steuerung dieser Abläufe, die Auswertung der Signale und die Speicherung der Daten werden durch vernetzte Rechnereinheiten umgesetzt, die eine Verbindung zu den Steuergeräten besitzen. ASAM versucht entsprechend dieser Abläufe, die Schnittstellen unter den verschiedenen Systemen in Prüfstandsanlagen und die dabei eingesetzten Datenformate zu spezifizieren.[128]

Diese Standardisierungen umfassen Protokolle, Datenmodelle, Datenformate und APIs (Application Programming Interfaces) für die Verwendung in der Entwicklung und Testphase von elektronischen Systemkomponenten. Eine hohe Anzahl an Tools in den Bereichen der Simulation, Datenerfassung, Kalibrierung und Erprobungsautomation sind konform zu den ASAM Standards. Dementsprechend können Daten zwischen unterschiedlichen Anwendungen ohne den Einsatz von Konvertern problemlos ausgetauscht und anschließend ausgewertet werden. Des Weiteren wird der Austausch von Spezifikationen zwischen Fahrzeugherstellern und Zulieferunternehmen und damit der vollständige Entwicklungsablauf erleichtert.[129]

[128] Vgl. Zimmermann/Schmidgall (2014), S. 229.

[129] Vgl. ASAM e.V. (2015), Online-Quelle [20.August.2016].

ASAM unterteilt deren Standards in drei Gruppen und weiter in Unterbereiche. Diese Gruppen werden als AE (Automotive Electronics), CAT (Computer Aided Testing) und COMMON (gemeinsam genutzte Standards für AE und CAT) bezeichnet. Die AE Standardreihe wird vorrangig in der Entwurf- und Implementierungsphase während der Entwicklung von ECU-Software verwendet und beinhaltet unter anderem die Standards CDF (Calibration Data Format), MDX (Model Data Exchange Format) und MCD (Measure, Calibrate, Diagnose). Die CAT Standards werden hingegen vorwiegend zur Validierung und Verifikation von ECU-Software genutzt und umfassen die Spezifikationen ACI (Automatic Calibration Interface), CEA (Components for Evaluation and Analysis), GDI (Generic Device Interface) und ODS (Open Data Services). Die COMMON Standards werden sowohl bei AE als auch bei CAT eingesetzt und beinhalten die Definitionen LXF (Layout Exchange Format) und MDF (Measurement Data Format). Da lediglich die ASAM AE MCD Reihe für diese Arbeit relevant ist, werden die übrigen Spezifikationen nicht näher erläutert.[130]

Die ASAM AE MCD Standards dienen als Zwischenebene (Middleware), die eine Verbindung zwischen Steuergeräten auf der untersten Schicht und den darüber angeordneten Test- bzw. Diagnoseapplikationen ermöglicht. Dementsprechend besteht die Aufgabe dieser Zwischenebene darin, den höher liegenden Ebenen ein einheitliches Interface bereitzustellen und Implementierungsinformationen der Steuergeräte aufzunehmen. Die Software dazu wird dabei auf einem Test- oder Diagnosesystem betrieben. Diese Systeme sind vergleichbar mit einem PC, besitzen Schnittstellen, wie beispielsweise Ethernet, um eine Verbindung mit anderen Systemen zu ermöglichen und können mit Steuergeräten über verschiedene Bussysteme, wie CAN, FlexRay oder LIN, kommunizieren. Für bestimmte Anwendungsaufgaben werden Steuergeräte oftmals modifiziert, um Applikationen sowie Parameter einfacher anpassen und Werte schneller verändern zu können. Bei einer solchen Modifikation werden der ROM-Speicher vergrößert und durch einen RAM-Speicherbereich ergänzt (ETK, Emulatortastkopf). Eine weitere wichtige Eigenschaft der MCD Standards ist die standardisierte und herstellerübergreifende Definition der ECU-Eigenschaften in Datenbanken mittels eines spezifizierten Datenaustauschformats. Demzufolge beinhaltet eine derartige Datenbank sämtliche Parameter zu den im Steuergerät vorhandenen Variablen und Funktionen, wie beispielsweise Speicheradressen und Umrechnungsfaktoren zwischen internen und physikalischen Werten. Zusammengefasst spezifiziert ASAM AE MCD drei unterschiedliche Schnittstellen mit der Bezeichnung ASAM 1, 2 und 3. Diese drei Schnittstellen sind in der folgenden Abbildung (Abb. 27) dargestellt. ASAM 1 dient als Schnittstelle zu den Steuergerätesystemen. ASAM 2 wird als Interface zur entsprechenden Steuergerätedatenbank eingesetzt und ASAM 3 ist eine Schnittstelle zu den darüber angeordneten Erprobungs- bzw. Diagnoseapplikationen.[131]

[130] Vgl. Paulweber/Lebert (2014), S. 275ff.

[131] Vgl. Zimmermann/Schmidgall (2014), S. 229f.

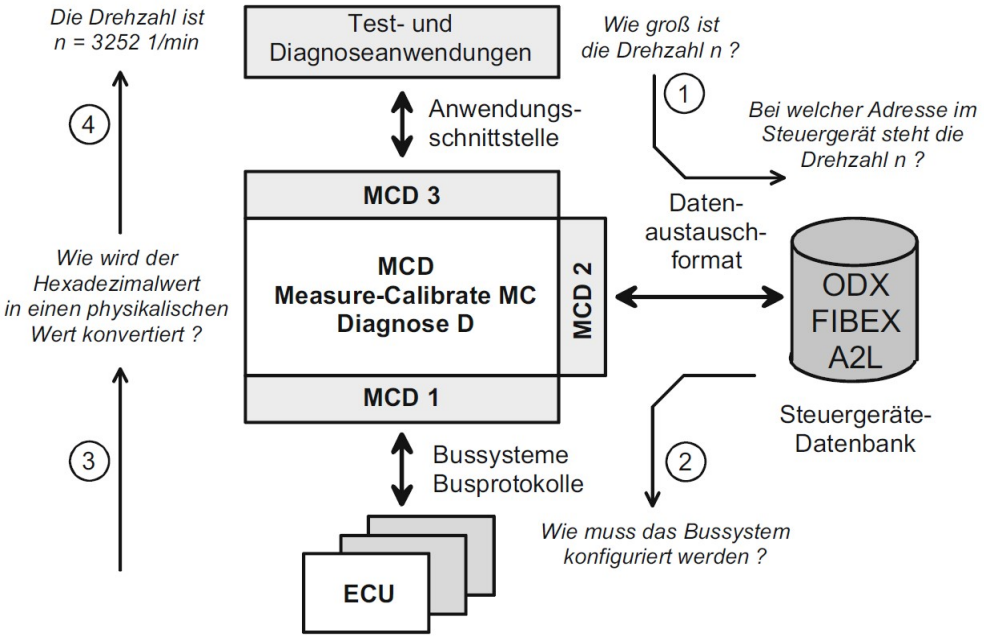


Abb. 27: ASAM AE MCD Schnittstellen, Quelle: Zimmermann/Schmidgall (2014), S. 231.

Entgegen der vorigen Darstellung ist die Struktur der MCD Standards nicht durchgehend vereinheitlicht. Nachdem sich die Vorgängerorganisation ASAP nur mit Applikationssystemen im Bereich Messen und Kalibrieren (MC) auseinandergesetzt hat, wurde die Diagnoseschnittstelle (D) von der ISO spezifiziert. Ab dem Beginn des Jahrs 2000 wurde versucht die Diagnosebereiche in die MC Standards miteinzubeziehen. Dazu mussten die bis dahin entstandenen Spezifikationen in beiden Bereichen berücksichtigt werden. Aufgrund dessen gibt es für die drei Schnittstellen ASAM 1, 2 und 3 jeweils einen MC und einen D Standard. Diese sind zwar so spezifiziert, dass sie gemeinsam verwendet werden können, sind aber unter den Schnittstellen nicht vereinheitlicht. Des Weiteren wurden die verschiedenen Standards über die Jahre wiederholt überarbeitet. So wurde beispielsweise das Busprotokoll zu den Steuergeräten (ASAM MCD-1 MC) von CCP (CAN Calibration Protocol) zu XCP (Extended Calibration Protocol) weiterentwickelt. Auch das Datenaustauschformat der ASAM 2 Schnittstelle wurde überarbeitet. Zu Beginn wurde ein proprietäres Textformat (ASAM MCD-2 MC) spezifiziert. Das Textformat wird auch AML (ASAM2 Meta Language) oder ASAP Classic genannt. In den nächsten Jahren soll dieses Textformat durch das XML-basierte (Extensible Markup Language) CDF (Calibration Data Format, 2 MC CDF) abgelöst bzw. ergänzt werden. Die Formate ODX (Open Data Exchange, 2 D), OTX (Open Test Sequence Exchange, ISO 13209 OTX) und FIBEX (Field Bus Exchange Format, 2 NET) sind bereits XML-basiert. ODX dient zur Definition der Diagnosedaten, OTX zur Beschreibung von Diagnoseabläufen und FIBEX zur Darstellung der ECU-internen Kommunikation. Standards die noch mit der Bezeichnung ASAP spezifiziert wurden, sind mittlerweile auf ASAM umgestellt und einzelne ASAM-Spezifikationen von der ISO genormt worden. Beispielsweise sind ODX nun als ISO 22901 und die ASAM MCD-3 D Schnittstelle als ISO 22900 definiert.[132]

[132] Vgl. Zimmermann/Schmidgall (2014), S. 230ff.

3.2 ASAM AE MCD-1

Die unterste Schicht der MCD Reihe (Schicht 1) wird bei Mess- und Kalibriertätigkeiten eingesetzt und besteht aus den Teilen 1a und 1b. Der Teil 1a beinhaltet die Spezifikation der Applikationsprotokolle CCP und XCP. Der Teil 1b definiert das Interface zur nächsthöheren Ebene. Bei beiden genannten Teilen werden zur Parametrierung Konfigurationsdateien benötigt.[133] Jedoch wurde diese 1b Schnittstelle (ASAP1b) lediglich bei einer geringen Anzahl an Anwendungen eingesetzt und hat daher nahezu keine Relevanz mehr. XCP ist dagegen derart anpassungsfähig gestaltet, dass es die Anforderungen einer universellen herstellerunabhängigen Schnittstelle alleine abdecken kann.[134]

Als Konfigurations- bzw. Beschreibungsdateien für CCP und XCP dienen Textdateien mit AML Syntax. Diese werden für gewöhnlich mit der Dateiendung „.a2l“ abgespeichert und beschreiben unter anderem XCP-Befehle sowie die Fähigkeiten und Parameter des entsprechenden Steuergeräts. Zur besseren Übersicht werden bestimmte Teile der Konfigurationsdaten (busabhängig und busunabhängig) ausgelagert, mit der Dateiendung „.aml“ abgespeichert (C-artiger Syntax) und in die zentrale Konfigurationsdatei inkludiert. In Zukunft könnten diese AML-Dateien von dem XML-basierten CDF in Kombination mit FIBEX für die Buskommunikation abgelöst werden.[135]

3.2.1 ASAM AE MCD-1 CCP

Das CAN Calibration Protocol, kurz CCP, ist ein Protokoll, dass dem CAN-Bus überlagert ist und dazu dient über diesen Kalibrier- und Messdaten zu übermitteln und dementsprechend auf steuergerätinterne Daten zugreifen zu können. Das CCP nutzt im Gegensatz zu CAN ein Master-Slave-Verfahren. Demzufolge hat bei Verwendung des CCPs lediglich das Applikationssystem als Master die Berechtigung selbstständig Informationen zu übertragen. Die Steuergeräte dürfen erst nach Aufforderung des Masters Informationen übertragen. Objekte, die vom Master übermittelt werden, bezeichnet man als CRO (Command Receive Object). Darauf antworten die Slaves mit DTOs (Data Transmission Objects). Zyklische Daten (fortlaufende Messdaten) von Steuergeräten an den Master werden als DAQ-DTO (Data Acquisition DTO) bezeichnet. Nachdem das CCP auf dem CAN-Protokoll aufbaut, muss das CAN-Botschaftsformat angepasst werden, da neben den CAN-Informationen CCP-abhängige Parameter (Protokollinformationen) übertragen werden müssen. Eine CAN-Botschaft beinhaltet maximal 8 Bytes an Nutzdaten. Bei CCP müssen einige dieser 8 Datenbytes (CRO 2 Bytes, DTO 3 Bytes, DAQ-DTO 1 Byte) für Protokollinformationen verwendet werden, da diese Informationen nicht außerhalb der CAN-Botschaft übermittelt werden können, um die Kompatibilität zur CAN-Spezifikation aufrechtzuerhalten. Die Nutzdatenrate verringert sich zwar durch diese Protokollschachtelung, bleibt jedoch in einem für die Praxis akzeptablen Bereich. Auf die Variante, den Identifier einer Nachricht anzupassen wird bei CCP nicht zurückgegriffen.[136]

---

[133] Vgl. Zimmermann/Schmidgall (2014), S. 232f.

[134] Vgl. Patzer/Zaiser (2014), S. 9.

[135] Vgl. Zimmermann/Schmidgall (2014), S. 254ff.

[136] Vgl. Borgeest (2014), S. 263f.

3.2.2 ASAM AE MCD-1 XCP

Das Universal Measurement and Calibration Protocol (XCP) ist eine Weiterentwicklung des CCP. Es handelt sich dabei um ein busunabhängiges Kommunikationsprotokoll mit Master-Slave-Verfahren. Folglich wurde das von CAN abhängige CCP zu dem universell einsetzbaren XCP weiterentwickelt, damit neben CAN (XCP on CAN) auch auf andere Kommunikationsschnittstellen (XCP on FlexRay, XCP on USB, XCP on Ethernet etc.) zugegriffen werden kann. XCP ermöglicht Flashprogrammierung, Lese- bzw. Schreibzugriff auf steuergerätinterne Daten, Datenakquisition und -stimulation sowie zusätzliche, optionale Funktionen.[137] Die Zugriffe werden adressorientiert abgehandelt. Dabei ermöglicht der lesende Zugriff das Auslesen von Daten aus dem RAM und der schreibende Zugriff das Verändern von Eigenschaften im RAM. Des Weiteren erlaubt XCP eine synchrone Messung zu den Ereignissen in der ECU. Demnach wird gewährleistet, dass die Messdaten untereinander zusammenhängen. Nach jeder erstmaligen Initialisierung einer Messung sind die Messsignale gesondert auswählbar. Wichtig für den schreibenden Zugriff ist, dass die zu verstellenden Parameter im RAM vorhanden sind. Dazu ist ein Kalibrierkonzept erforderlich. Die Daten wiederum werden bei XCP botschaftsorientiert zwischen Master und Slave übertragen. Dabei ist der gesamte XCP Datenrahmen in das Botschaftsformat der entsprechenden Transportschicht integriert. Bei XCP on Ethernet erfolgt dies mittels UDP in einem UDP-Paket. Der Rahmen setzt sich aus den Teilen XCP Header, XCP Packet und XCP Tail zusammen. Im Gegensatz zu Header und Tail ist das Packet unabhängig vom entsprechenden Transportprotokoll und beinhaltet stets die drei Bereiche Identification Field, Timestamp Field und Data Field. Das Identification Field startet mit einem Packet Identifier (PID), zur Identifikation des Pakets. Der Ablauf einer Kommunikation über das XCP Packet wird in einen Teil für Kommandos (CTO) und in einen Teil für die Übermittlung synchroner Daten (DTO) gegliedert.[138]

Bei XCP sind fünf verschiedene Typen von Command Transfer Objects (CTO) definiert. Gestartet wird die Übertragung mit einem Kommandocode (CMD) vom Master zum Steuergerät. Darauf folgen vom Steuergerät je nach Zustand die Objekte Antwort (RES), Fehler (ERR), Ereignis (EV) und Server Request Processor (Serv). Für das Übermitteln von Datenblöcken werden je nach Übertragungsrichtung DAQ- oder STIM-DTO (Stimulation) eingesetzt. DAQ-DTO werden für Übertragungen von der ECU zum Master genutzt, STIM-DTO wiederum vom Master zum Steuergerät.[139]

3.3 ASAM AE MCD-2

Die ASAM AE MCD-2 Reihe umfasst die Standards D, MC und NET. Die Standards sind auch unter ODX, ASAP2 und FIBEX bekannt und dienen als Beschreibungs- bzw. Konfigurationsdateien für die ECU-Diagnose, den Austausch von Mess- und Kalibrierdaten und die Kommunikation über die verschiedenen Bussysteme.[140]

---

[137] Vgl. Paulweber/Lebert (2014), S. 278.

[138] Vgl. Patzer/Zaiser (2014), S. 18ff.

[139] Vgl. Borgeest (2014), S. 264.

[140] Vgl. Paulweber/Lebert (2014), S. 278f.

3.3.1 ASAM AE MCD-2 D

ASAM AE MCD-2 D, auch bekannt unter ODX, definiert ein XML-basiertes Format, welches zur Definition von Programmierungs-, Diagnose- und fahrzeugtypischen Schnittstellendaten für die Übertragung von Informationen zwischen ECU und externen Testsystemen dient. Testgeräte, die diesem Standard entsprechen müssen nicht eigens programmiert werden, um Diagnosedaten von einer ECU auslesen zu können. Der Standard wird von ASAM als Data Model Specification for ECU Diagnostics betitelt.[141]

Die Diagnose eines Steuergeräts umfasst jene Aufgaben, die bei der Fehlersuche und Wartung sowie im Laufe der Produktion eines Kraftfahrzeugs getätigt werden müssen, wie beispielsweise das Auslesen von Parametern oder die Realisierung von verschiedenen funktionalen Tests.[142]

3.3.2 ASAM AE MCD-2 MC

Der ASAM AE MCD-2 MC Standard wird auch als ASAP2 bezeichnet und beschreibt den Aufbau eines Applikationsdatensatzes, der die Definition von Messwerten (Measurement) und Anwendungsparametern (Characteristic) in einem nicht XML-basierten Format (AML) beinhaltet. Der Datensatz ermöglicht den Lese- bzw. Schreibzugriff auf Informationen im Speicher der ECU mittels eines Anwendungssystems. Des Weiteren beinhaltet das Format die Definition der Hardwareschnittstelle der ECU für die Parametrierung der entsprechenden Treiber des Anwendungssystems. Der Titel dieses Standards lautet ECU Measurement and Calibration Data Exchange Format.[143]

Die Definition, ob es sich bei steuergerätinternen Variablen um Mess- bzw. Verstellobjekte handelt, wird mit derselben Syntax, die auch für die Definition der CCP- bzw. XCP-Schnittstelle zwischen Anwendungssystem und ECU eingesetzt wird, umgesetzt. Neben ASAP2 wird auch oftmals A2L (Dateiendung des Datensatzes) als Bezeichnung für diese Applikationsdatensätze gewählt. Ein derartiger Datensatz beinhaltet unter anderem Projektinformationen, Informationen zu einer bestimmten ECU, Adressen und Größe der Speicherbereiche, Defaultwerte für Datenstrukturen, das Interface zur ECU nach ASAM AE MCD-1, eine Beschreibung von veränderbaren Parametern, Kennfeldern sowie Kennlinien, die Beschreibung von Messdaten und Umrechnungstabellen bzw. -methoden für die Umrechnung von internen Hex-Werten in physikalische Größen. Das XML-basierte CDF ergänzt die ASAP2/A2L-Datensätze um übergeordnete Informationen und könnte diese in Kombination mit MDX in Zukunft ersetzen.[144] Beim CDF handelt es sich um ein XML-basiertes Format, das dem Aufzeichnen von Anwendungsinformationen und den dazugehörigen Metadaten dient. Es beinhaltet die Anwendungsparameter, welche im MCD-2 MC Standard definiert sind. MDX hingegen definiert ein XML-basiertes Format, das unter anderem der Beschreibung von Funktionsschnittstellen und den dazugehörigen Informationen dient.[145]

---

[141] Vgl. Paulweber/Lebert (2014), S. 278.

[142] Vgl. Zimmermann/Schmidgall (2014), S. 277.

[143] Vgl. Paulweber/Lebert (2014), S. 278.

[144] Vgl. Zimmermann/Schmidgall (2014), S. 272ff.

[145] Vgl. Paulweber/Lebert (2014), S. 277ff.

3.3.3 ASAM AE MCD-2 NET

Der Titel des ASAM AE MCD-2 NET Standards, auch unter den Namen FIBEX bekannt, lautet Data Model for ECU Network Systems. Es handelt sich dabei um die Spezifikation eines XML-basierten Formats, das der Definition von Nachrichten und deren Zeitverhalten für Bussysteme in Kraftfahrzeugen dient. FIBEX wird vorrangig bei FlexRay und MOST eingesetzt, unterstützt neben diesen jedoch auch TTCAN, LIN, CAN sowie Ethernet. Verwendung findet FIBEX bei der Parametrierung, Simulation, Datenspeicherung und beim Design der Buskommunikation.[146]

FIBEX soll mittelfristig bisher gängige proprietäre Datenformate wie beispielsweise LIN Description File (LDF), Node Capability File (NCF) oder CANdb (DBC Dateien) ersetzen. Das Hauptaugenmerk von FIBEX liegt in der Definition der On-Board Kommunikation im gewöhnlichen Fahrzeugbetrieb und wird von Simulations-, Spezifikations- und Erprobungstools eingesetzt. Die wesentlichen Inhalte einer FIBEX-Beschreibungsdatei sind die Struktur des Busnetzwerks, die eingesetzten Bussysteme sowie ECUs, der Ablauf der Kommunikation (Botschaften, Signale und Dateninhalte), verwendete Codierungen bzw. Einheiten und Daten für Netzwerkmanagement sowie Transportprotokolle. Der folgende Code zeigt den Ausschnitt einer FIBEX-Datei, die ein Beispielsystem abbildet (zwei ECUs/Bussysteme gekoppelt über ein Gateway). Dabei handelt es sich um die Beschreibung eines Steuergeräts mit der ID ecuMotorSteuergerät. Anhand von CHANNEL-REF wird auf das entsprechende Bussystem (hier chMotorCAN) an das die ECU angeschlossen ist, verwiesen. FRAME-TRIGGERING-REF gibt an, welche Nachrichten empfangen bzw. gesendet werden und im Abschnitt CONTROLLER können die Type des Kommunikationscontrollers und dessen Parameter angeführt werden.[147]

```
<fx:ECUS>
  <fx:ECU ID="ecuMotorSteuergerät">
    . . .
    <fx:CONNECTORS>
      <fx:CONNECTOR ID=". . .">
        <fx:CHANNEL-REF ID-REF="chMotorCAN"/>
        <fx:OUTPUTS ID=". . .">
          <fx:FRAME-TRIGGERING-REF ID-REF="ftgGeschwindigkeit"/>
        </fx:OUTPUTS>
      </fx:CONNECTOR>
    </fx:CONNECTORS>

    <fx:CONTROLLERS>
      <fx:CONTROLLER xsi:type="can:CONTROLLER-TYPE" ID="...">
        . . .                                                    {16}
        <fx:CHIP-NAME>SJA1000</fx:CHIP-NAME>
        . . .
      </fx:CONTROLLER>
    </fx:CONTROLLERS>
  </fx:ECU>
```

Abb. 28: FIBEX-Beschreibungsdatei, Quelle: Zimmermann/Schmidgall (2014), S. 265.

[146] Vgl. Paulweber/Lebert (2014), S. 279.

[147] Vgl. Zimmermann/Schmidgall (2014), S. 261ff.

3.4 ASAM AE MCD-3

Die ASAM AE MCD-3 Schnittstelle dient der Anknüpfung an übergeordnete Mess-, Verstell- und Diagnosetools, die beispielsweise zur Prüfstandsautomatisierung genutzt werden.[148] Die MCD-3 Standardreihe umfasst die Spezifikationen ASAP3 (veraltet), D und MC. ASAP3 ist mittlerweile vom MC Standard abgelöst worden.[149] Die übergeordneten Mess-, Kalibrier- und Diagnoseapplikationen werden durch die MCD-3 Schnittstelle von den Informationen zu den verwendeten Transportprotokollen und Bussystemen sowie von der vollständigen Datenhaltung für die diesbezüglichen Definitionsdateien (ASAM AE MCD-2) entkoppelt. Der Grundgedanke liegt darin, MCD-Tools nicht für jedes Fahrzeugsystem und jede ECU neu parametrieren zu müssen, sondern eine allgemein einsetzbare Anwendung, den MCD-3 Server, zu verwenden. Dieser konfiguriert sich eigenständig auf das entsprechende Fahrzeugsystem und dessen ECUs anhand der ASAM 2 Definitionsdateien. Der Client dieses Servers ist nicht spezifiziert, um individuelle Lösungen zu erlauben. Das Interface zum Server und dessen Aufbau ist hingegen definiert. Dabei kann die Kommunikation bzw. Datenübertragung zwischen Server und Client mit unterschiedlichen Programmiersprachen und gängigen Technologien umgesetzt werden. Beispielsweise sind Java, C/C++, COM/DCOM, Corba und GDI im Standard enthalten.[150]

3.4.1 ASAM AE MCD-3 D

ASAM AE MCD-3 D spezifiziert eine objektorientierte API für einen Diagnoseserver, der protokoll- und busunabhängige Funktionen für den Datenaustausch bzw. die Verbindung zwischen Clientanwendungen und ECUs bereitstellt. Der Standard umfasst dabei eine Darstellung der objektorientierten API auf COM-IDL, Java und C++ mit Beispielprogrammteilen und ergänzt den MCD-2 D Standard. Der Titel der MCD-3 D Spezifikation lautet Application Programming Interface for MVCI Diagnostic Server.[151]

3.4.2 ASAM AE MCD-3 MC

Der ASAM AE MCD-3 MC Standard mit dem Titel Application Programming Interface for Measurement and Calibration Server spezifiziert eine objektorientierte API für einen Anwendungs- und Messserver. Dieser Server stellt wie auch der Diagnoseserver protokoll- und busübergreifende Funktionen bereit, um eine Verbindung und damit einen Datenaustausch zwischen Clientanwendungen und ECUs aufbauen zu können. Der Standard beinhaltet unter anderem eine Darstellung der objektorientierten API auf COM-IDL mit Democodes und ergänzt die MCD-2 MC Definition. Des Weiteren löst 3 MC den veralteten ASAP3 Standard mit dem Titel Automation/Optimization and ECU Calibration System Interface ab. Dieser definiert ein auf RS232 aufsetzendes Protokoll zur Kommunikation zwischen Prüfstandssystem und einem MCD-Tool.[152]

---

[148] Vgl. Patzer/Zaiser (2014), S. 9.

[149] Vgl. Paulweber/Lebert (2014), S. 279.

[150] Vgl. Zimmermann/Schmidgall (2014), S. 270f.

[151] Vgl. Paulweber/Lebert (2014), S. 279.

[152] Vgl. Paulweber/Lebert (2014), S. 279.

Der Aufbau eines MCD-3 Servers und dessen Funktionsgruppen (M, C und D) sowie Schnittstellen (ASAM 1, 2 und 3) sind in der folgenden Darstellung ersichtlich. Der Server läuft dabei auf einem Mess- und Kalibriersystem (z.B. CANape), der Client auf einem übergeordneten Automatisierungssystem. Damit ist diese beschriebene Struktur mit dem Aufbau des bei dieser Arbeit betrachteten Prüfstandssystems vergleichbar.

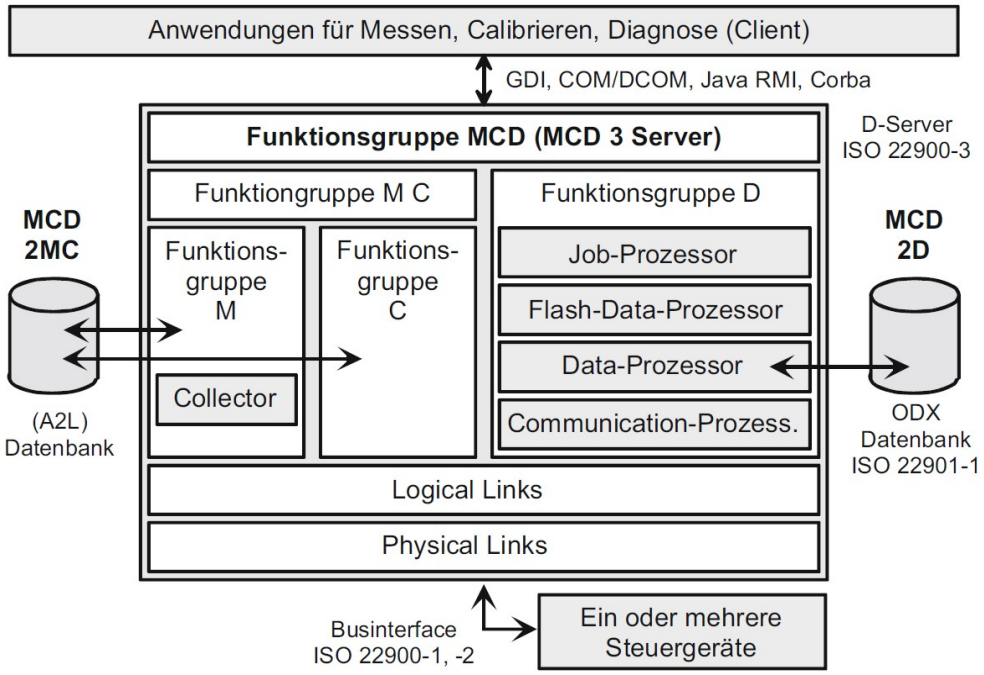


Abb. 29: ASAM MCD-3 Server, Quelle: Zimmermann/Schmidgall (2014), S. 271.

4 CANAPE AUTOMATISIERUNGSSCHNITTSTELLEN

In Prüfstandssystemen ist es wesentlich, dass ein schneller und einfacher Zugriff auf alle wesentlichen Ein- bzw. Ausgangsvariablen einer ECU möglich ist. MCD-Tools wie CANape von Vector bieten verschiedene Datenerfassungsmöglichkeiten und erlauben einen raschen sowie unkomplizierten Zugriff auf alle steuergerätinternen Variablen. Um Daten zwischen MCD-Tool am Applikationsrechner (Server) und übergeordnetem Prüfstandssystem (Client) austauschen zu können, wird eine sogenannte Automatisierungsschnittstelle benötigt.[153] [154]

Im Folgenden werden die von CANape unterstützten Standardschnittstellen sowie die Vector Interfaces (Abwandlungen der ASAM AE MCD-3 Standards) erläutert, miteinander verglichen und der mögliche Einsatz im geforderten Prüfstandsbetrieb beurteilt.

4.1 Standardschnittstellen

CANape unterstützt drei Standardschnittstellen in den angeführten Versionen: ASAM MCD-3 ASAP3 (2.1), ASAM AE MCD 3MC (1.01) und iLinkRT (1.3 & V2). Das ASAP3 Protokoll wurde 1999 von ASAM eingeführt und ist in vielen Prüfstandssystemen integriert. Die Kommunikation läuft dabei entweder über eine RS232- oder eine Ethernet-Verbindung und setzt die Installation des ASAP3Translators (inkludiert im Umfang der CANape Installation) auf dem Applikations-System-PC (CANape Server) voraus. Es handelt sich dabei um eine protokollorientierte Kommunikation zwischen Client (Prüfstandssystem/Vorgaberechner) und Server. ASAP3 unterstützt die Aufzeichnung von Messungen, den Lese- und Schreibzugriff auf steuergerätinterne Kalibrierparameter, das Starten von CANape Skripts, und dient als Konfigurationstool für iLinkRT. Die Vorteile dieses Protokolls sind die einfache Bedienung, die hohe Messrate von bis zu 500Hz (bei Verwendung via Ethernet) und, dass keine weitere CANape Option notwendig ist. Dagegen spricht jedoch, dass dieses Protokoll mittlerweile veraltet ist und keine weitere Entwicklung seitens ASAM zu erwarten ist. Darüber hinaus ist lediglich eine Messrate für alle Objekte wählbar. Nicht zyklische Raten werden ebenfalls nicht unterstützt. Bei der ASAM AE MCD-3 MC (1.01) Schnittstelle handelt es sich um eine API, die über DCOM kommuniziert. Im Vergleich zu ASAP3 wird eine erheblich höhere Anzahl an Funktionen unterstützt. Die aktuelle Version des ASAM AE MCD-3 MC Standards lautet 3.0. Diese ist nicht abwärtskompatibel zu den Versionen 2.0 und 1.01. CANape unterstützt jedoch lediglich die Version 1.01. Für den Betrieb ist am Applikations-System-PC, der wie bei der ASAP3 Schnittstelle als CANape Server fungiert, das Tool VMC3Server zu installieren. Die Schnittstelle unterstützt unter anderem Messaufzeichnungen, den Lese- und Schreibzugriff auf Kalibrierobjekte und kann ebenfalls als Konfigurationstool für das iLinkRT Interface genutzt werden. Die Nachteile dieser Schnittstelle sind das komplexe Design, die zusätzlich erforderliche Lizenzoption MCD-3 MC Server und der hohe administrative Aufwand auf dem Client- und Server-Rechner aufgrund von DCOM. iLinkRT wird hingegen zur Übertragung von Messdaten genutzt. Die Konfiguration der Messung

[153] Vgl. Knoll/Hendratno/Herzog (2015), S. 3.

[154] Vgl. Kless (2013), Online-Quelle [29.April.2016], S. 18f.

ist über diese Schnittstelle jedoch nicht möglich. Dazu wird parallel auf eine Schnittstelle wie CANapeAPI, ASAP3 oder MCD-3 MC zurückgegriffen. Nach erfolgreicher Konfiguration werden die Messdaten direkt über die iLinkRT Verbindung vom Server an den Client übertragen. Dabei entstehen keine Verzögerungen durch das Passieren der Schnittstelle wie bei ASAP3 oder MCD-3 MC. Seitens CANape wird die Version 1.3 unterstützt. Diese wurde von ETAS spezifiziert. Des Weiteren unterstützt CANape die von Vector entwickelte iLinkRT V2 (Vector 2) Schnittstelle. Dabei handelt es sich um eine Erweiterung von iLinkRT 1.3 für bestimmte Kalibrieranwendungsfälle. Demzufolge können bei dieser Version Mess- und Kalibrierdaten über die iLinkRT Verbindung übertragen werden. iLinkRT V2 basiert auf XCP on Ethernet. Die Messdaten werden demnach anhand des XCP on Ethernet DAQ Mechanismus (UDP) an den Client gesendet. Die Vorteile von iLinkRT V2 sind die einfache Bedienbarkeit, die hohen Messraten von über 500 Hz und der schnelle Schreibzugriff auf Kalibrierdaten (ab 3 ms pro Zugriff). Demgegenüber benötigt diese Schnittstelle jedoch ein zusätzliches Interface (CANapeAPI, ASAP3 oder MCD-3 MC) zur Messkonfiguration und die CANape Option MCD-3 MC Server.[155] Die folgende Darstellung zeigt eine beispielhafte Verbindungskonfiguration für iLinkRT V2.

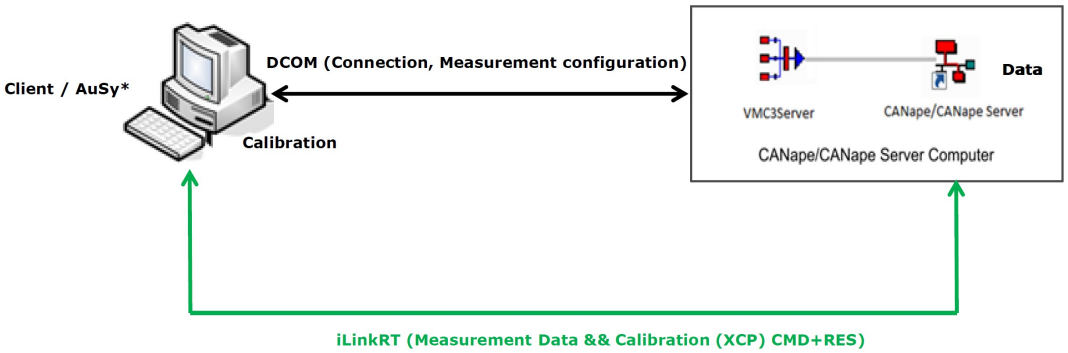


Abb. 30: iLinkRT V2, Quelle: Knoll/Hendratno/Herzog (2015), S. 24.

4.2 Vector Schnittstellen

Neben den erwähnten Standardschnittstellen unterstützt CANape die drei eigens von Vector entwickelten Interfaces CANapeAPI/CANapeTCP, CANape COM/Ethernet und MATLAB MCD-3.[156]

Bei CANapeAPI/CANapeTCP handelt es sich um ein C++ DLL Interface für die Verbindung zwischen Server und Client. Dazu umfasst die Installation von CANape zwei DLL-Dateien: CANapAPI.dll für den lokalen Betrieb und CANaptcp.dll für die Kommunikation über eine Netzwerkverbindung. Das API/TCP Interface verbindet alle übrigen fünf Schnittstellen mit CANape, kann jedoch auch direkt für die Kommunikation verwendet werden. Dementsprechend dient diese Schnittstelle als direktes Interface zu CANape. Die Vorteile dieser Schnittstelle sind die hohe Performance, die einfache Bedienung über Funktionsaufrufe, die unkomplizierte Einbindung für C++ Clients und die Tatsache, dass die DLL-Dateien im Umfang der CANape Installation inkludiert sind. Der Nachteil dieses Interfaces ist, dass es nicht bei skriptbasierten Clients eingesetzt werden kann. Das CANape COM Interface wiederum ermöglicht einen lokalen Betrieb und einen Netzwerkbetrieb. Aus diesem Grund wird die Schnittstelle als CANape

[155] Vgl. Knoll/Hendratno/Herzog (2015), S. 15ff.

[156] Vgl. Patzer/Herzog (2016), S. 92.

COM/Ethernet bezeichnet, jedoch wird auf den Ethernet Teil zumeist verzichtet. Die Ethernet-Verbindung (WinSock basiert) wird seit CANape 11 unterstützt. Die für diese Schnittstelle erforderlichen Client- und Serveranwendungen (COM Client und CANape Control Server) sind im CANape Installationsumfang inkludiert. Die Vorteile dieser Schnittstelle sind der hierarchische Systemaufbau, die Unterstützung von Programmiersprachen wie VBA/VBS, VB, C#, Python, LabVIEW und die Inkludierung in der CANape Installation. Darüber hinaus sind Anwendungsbeispiele unter anderem für C#, Excel und VB vorhanden. Die Nachteile sind die geringere Performance als beim API Interface und der hohe Aufwand bei nicht skriptbasierten Programmiersprachen (z.B. C++). Die CANape MCD-3 MATLAB Schnittstelle dagegen wird in Verbindung mit MATLAB verwendet. Dazu wird das Interface in sogenannte m-Files integriert. Das Interface ist im MATLAB Integration Package (MIP) verfügbar und basiert auf CANapeAPI und CANapeTCP, unterstützt jedoch nicht alle Funktionen dieser Schnittstellen. Alle drei genannten Vector Schnittstellen ermöglichen die laufende Anpassung der CANape Konfiguration, die Aufzeichnung von Messdaten und den Zugriff auf den Messrekorder, den Lese- und Schreibzugriff auf steuergerätinterne Kalibrierparameter, die Durchführung eines Flashprozesses (ODX-F), die Diagnose mit CDD/ODX, den Zugriff auf die globalen Variablen in CANape, das Ausführen von CANape Skripts, das Auslesen von Error-Codes und den Zugriff auf die Konfiguration von iLinkRT.[157]

4.3 Bewertung

Von den Standardschnittstellen kommt für dieses Konzept grundsätzlich lediglich die ASAM MCD-3 ASAP3 Schnittstelle in Frage, da die beiden übrigen Interfaces ASAM MCD-3 MC und iLinkRT eine zusätzliche Lizenzoption fordern. Darüber hinaus ist der verwendete Standard der ASAM MCD-3 MC Schnittstelle veraltet und die iLinkRT Schnittstelle sehr komplex aufgebaut, da ein zusätzliches Interface zur Messkonfiguration benötigt wird. Gegen die ASAM MCD-3 ASAP3 Schnittstelle spricht jedoch, dass es sich auch hier um einen veralteten Standard handelt und dieser weder seitens Vector noch von ASAM weiterentwickelt wird.

Die Vector Schnittstellen werden hingegen laufend weiterentwickelt, decken einen großen Bereich der Funktionen von CANape ab, benötigen keine zusätzliche Lizenzoption und sind verhältnismäßig einfach zu integrieren, da Vector für deren Schnittstellen detaillierte Bedienungsanleitungen und technischen Support anbietet. Jedoch sind die beiden Schnittstellen CANapeAPI/CANapeTCP und CANape MCD-3 MATLAB eher ungeeignet. Das CANapeAPI/CANapeTCP Interface ist lediglich für C++ Clients zweckmäßig und die MCD-3 MATLAB Schnittstelle fordert den Einsatz von MATLAB. Das CANape COM Interface kann hingegen direkt über LabVIEW implementiert werden. Nachdem LabVIEW als Prüfstandssoftware dient, ist die Implementierung dieser Schnittstelle mit einem erheblich niedrigeren Aufwand verbunden. Darüber hinaus deckt dieses Interface alle erforderlichen Anforderungen ab. Einzig die Performance könnte zu niedrig sein, da diese geringer ist als bei der CANapeAPI/CANapeTCP Schnittstelle. Dementsprechend wird das COM Interface für den Praxistest verwendet und geprüft, ob die Performance dieser Schnittstelle für den Einsatz im Prüfstandsbetrieb ausreicht.

[157] Vgl. Knoll/Hendratno/Herzog (2015), S. 7ff.

5 PRAXISTEST AM PRÜFSTAND

Um ein aussagekräftiges Ergebnis zur Einsatztauglichkeit der gewählten Automatisierungsschnittstelle erhalten zu können, muss diese am Prüfstand getestet und deren Eigenschaften beurteilt werden. Dazu wird in diesem Kapitel der Aufbau der Prüfstandsanlage detailliert betrachtet und beschrieben. Anschließend werden die notwendigen Schritte zur Implementierung der Automatisierungsschnittstelle erläutert und diese getestet. Anhand der gewonnenen Informationen wird daraufhin ein Beispielprogramm erstellt und die Performance der Schnittstelle anhand von verschiedenen Testmessungen im Prüfstandsbetrieb beurteilt.

5.1 Aufbau

Die Prüfstandsanlage besteht aus mehreren Systemkomponenten, die untereinander verbunden sind. Im Detail zeigt dies die folgende Darstellung. Im Anschluss werden die Komponenten, deren Aufgaben, die verschiedenen Kommunikationsnetzwerke sowie die unterschiedlichen Datenverbindungen erläutert.

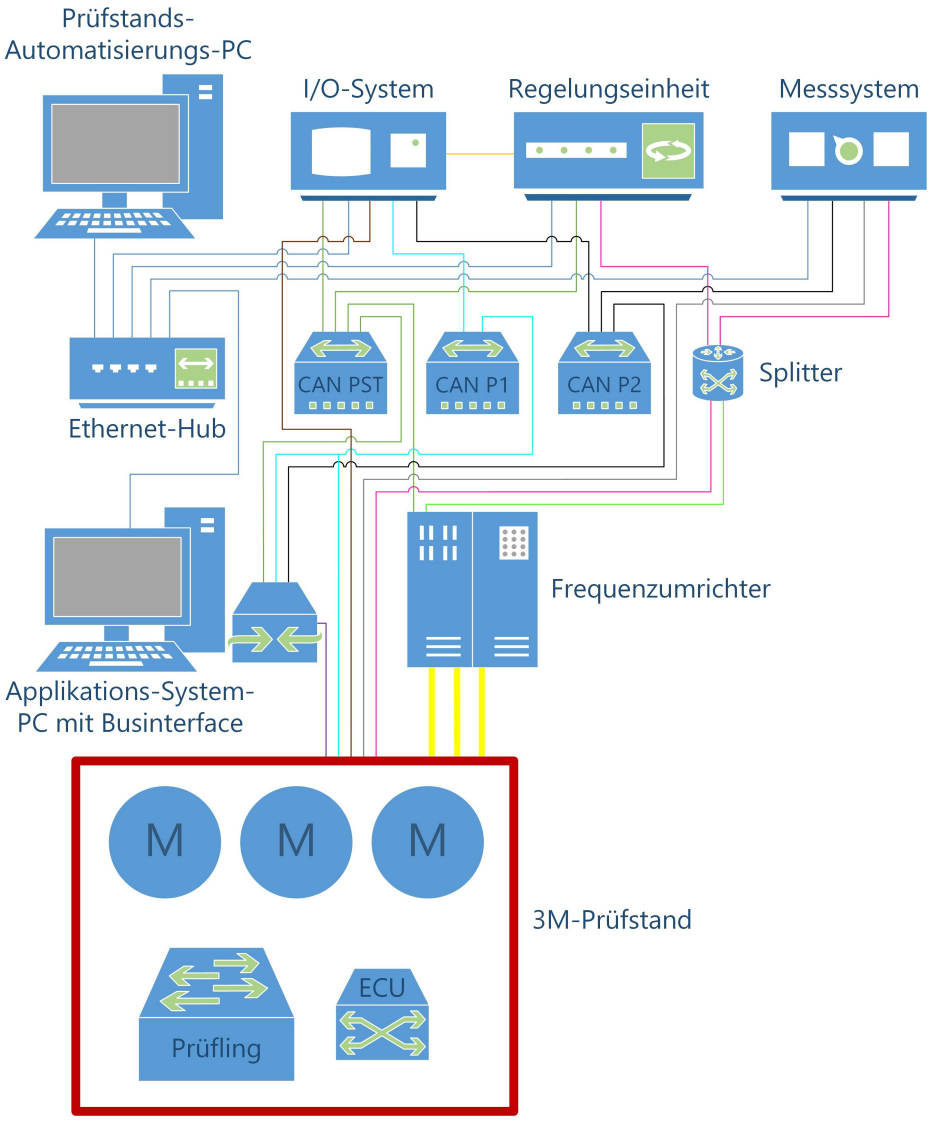


Abb. 31: Prüfstandsanlage, Quelle: Eigene Darstellung.

Wie in der vorhergehenden Abbildung (Abb. 31) ersichtlich, setzt sich die Anlage aus einem Prüfstands-Automatisierungs-PC, einem I/O-System, einer Regelungseinheit, einem Messsystem, einem Ethernet-Hub, drei CAN-Verteilern, einer Splittereinheit, einem Applikations-System-PC mit Businterface, drei Frequenzumrichtern und drei Elektromaschinen zusammen, um die Validierung von Antriebsstrangkomponenten anhand von unterschiedlichen Erprobungstests durchführen zu können. Die zu prüfende Komponente wird in der Darstellung als Prüfling bezeichnet. Die dazugehörige ECU wurde zur besseren Veranschaulichung einzeln als eigenes Objekt abgebildet. Nicht auf der Darstellung zu finden ist der Schnittstellenrechner. Dieser wird nicht weiter erwähnt, da auf diesem lediglich die Datensichtung, -auswertung und -ablage erfolgt und damit durch diesen kein direkter Einfluss auf den Prüfstandsbetrieb gegeben ist.

Der Prüfstands-Automatisierungs-PC bzw. Vorgaberechner, ein Industrie-PC mit leistungsstarker Hardware, dient dem Ausführen der Prüfstandssoftware (LabVIEW) und befindet sich im Ethernet-Netzwerk der Anlage. Über diese Ethernet-Verbindung (blau) erfolgt die Konfiguration und Parametrierung des I/O-Systems, der Regelungseinheit und des Messsystems anhand eigener Konfigurationstools. Darüber hinaus wird die Ethernet-Verbindung zum Datenaustausch zwischen den angeführten Komponenten inklusive dem Applikations-System-PC genutzt. Darunter fällt auch die Übermittlung der entsprechenden Vorgabeparameter bzw. Prüfprogramme vom Prüfstands-Automatisierungs-PC an das I/O-System. Demnach erfolgt die Abarbeitung der Vorgabeprogramme am I/O-System. Einfache Unterprogramme, beispielsweise zur Temperaturregelung, werden jedoch vollständig am Vorgaberechner ausgeführt. Dabei werden lediglich die Vorgabesignale zur weiteren Verarbeitung an das I/O-System übertragen. Die Anzeige der aktuellen Programme, Betriebszustände, Messwerte und Informationen, wie Laufzeit, Zyklusdauer, Programmname etc., läuft am Vorgaberechner. Darüber hinaus erfolgt auf diesem die Aufzeichnung und Fehlerüberwachung aller Signale des I/O-Systems, der Regelungseinheit und des Messsystems. Die Signale werden dazu in einer gemeinsamen Messdatei im .dat-Format (Diadem) gespeichert. Die Aufzeichnung wird in der Regel mit einer Messrate von 100 Hz durchgeführt.

Das I/O-System dient dem Einlesen von Prüfstandssignalen (DI, CAN und FlexRay), dem Übermitteln dieser Eingangssignale an den Prüfstands-Automatisierungs-PC und dem Ausgeben von Vorgabesignalen (CAN, FlexRay, DO und AO) entsprechend der vom Vorgaberechner gesendeten Prüfdaten. Dabei handelt es sich um ein National Instruments PXIe-System vom Typ 1062Q mit verbautem Real-Time Embedded Controller (NI PXIe-8840). Zusätzlich sind zwei CAN-Schnittstellenkarten (NI PXI-8512), eine FlexRay-Schnittstellenkarte (NI PXI-8517), ein Digital-I/O-Modul (NI PXI-6509) und ein Analogausgangsmodul (NI PXI-6733) integriert. Verbunden ist das System mit dem Ethernet-Netzwerk und den drei CAN-Netzwerken CAN-Prüfstand (grün), CAN-Prüfling 1 (türkis) und CAN-Prüfling 2 (schwarz). Darüber hinaus ist es via CAN (orange) mit der Regelungseinheit und über mehrere Signalleitungen (braun) mit dem Prüfstand verbunden. Diese Signalleitungen dienen dem Empfang bzw. der Ausgabe von digitalen Signalen (z.B. zum Schalten eines Relais zur Aktivierung einer Steckdose am Prüfstand) und der Ausgabe von Analogsignalen (z.B. Stellgröße für eine Temperaturregelung).

Die Regelungseinheit befindet sich ebenfalls im Ethernet-Netzwerk und ist via CAN mit dem Netzwerk CAN-Prüfstand sowie mit dem I/O-System verbunden. Mittels dieser beiden CAN-Verbindungen werden die Vorgabesignale vom I/O-System an die Regelung gesendet und nach anschließender Anpassung über das Netzwerk CAN-Prüfstand an die Frequenzumrichter weitergegeben. Dazu benötigt die Regelungseinheit die Drehmoment- und Drehzahlsignale der Maschinen (rosa). Drehmomente sowie Drehzahlen werden am Abtrieb der Maschinen mittels HBM-Drehmomentaufnehmer (T12) gemessen. Die an den Maschinen angebrachten Inkrementalgeber liefern jedoch genauere Werte. Aufgrund dessen werden für die Regelung die Drehmomentsignale der HBM-Drehmomentmesswellen und die Drehzahlsignale der Inkrementalgeber genutzt. Nachdem die Drehmomentsignale zusätzlich vom Messsystem aufgezeichnet und die Drehzahlsignale der Inkrementalgeber (hellgrün) auch von den Frequenzumrichtern für die interne Regelung benötigt werden, müssen diese gesplittet werden. Dazu werden Impulsverteiler mit Potenzialtrennung von motrona eingesetzt, da es sich um Frequenzsignale handelt.

Anhand des Messsystems werden Signale (grau), wie beispielsweise Temperaturen, Drehzahlen, Drehmomente, Kräfte etc., eingelesen und via Ethernet an den Prüfstands-Automatisierungs-PC zur Aufzeichnung bzw. Fehlerüberwachung übertragen. Bei diesem System handelt es sich um einen MGCplus von HBM mit verschiedenen Einschubkarten für Frequenz-, Temperatur- und Spannungssignale. Bei Restbussimulationen ist es oftmals notwendig Drehzahlen über CAN an das Steuergerät zu senden. Dementsprechend ist zusätzlich ein CAN-Ausgangsmodul mit zwei Kanälen verbaut. Mit diesem ist es möglich gemessene Signale über CAN wieder auszugeben. Dazu ist das Messsystem mit dem Netzwerk CAN-Prüfling 2 verbunden. Jedoch entsteht während dieses Vorgangs ein Zeitverzug, da die Signale eingelesen, verarbeitet, ausgegeben und schlussendlich am Businterface wieder eingelesen werden müssen.

Der Applikations-System-PC wird zur Bedatung der Steuergeräte und für Restbussimulationen mittels Businterface (Vector VN8900 oder VN7600) genutzt. Dazu sind auf diesem PC die Tools CANape 14.0 SP3, CANalyzer 8.2.98 SP4 und CANoe 8.5.80.4 SP3 von Vector installiert und an das Interface die drei CAN-Netzwerke CAN-Prüfstand, CAN-Prüfling 1, CAN-Prüfling 2 sowie das FlexRay-Netzwerk FR-Prüfling 1 (lila) angeschlossen. CANape wird zur Bedatung der Steuergeräte genutzt. Dabei werden beispielsweise Kennlinien angepasst, Parameter verstellt, Skripte ausgeführt, Kalibrierungen durchgeführt und Softwaremodelle aktualisiert. CANalyzer und CANoe werden für Restbussimulationen verwendet. Restbussimulationen werden genutzt, um für das Steuergerät einen realen Fahrbetrieb simulieren zu können. Signale die im Fahrzeug beispielsweise vom Motorsteuergerät an das Getriebesteuergerät gesendet werden, müssen am Prüfstand mittels Restbussimulation simuliert werden. Ohne diese am Prüfstand fehlenden Signale würde das Steuergerät nicht im vollen Umfang funktionieren. Darüber hinaus wird CANalyzer zur Übersicht des Datenverkehrs (Trace) der angeschlossenen Netzwerke genutzt. Je nach Steuergerät wird zu dessen Bedatung FlexRay oder CAN verwendet. Wie auch die übrigen Systeme ist der Applikations-System-PC mit dem Ethernet-Hub verbunden.

Die nachkommende Abbildung zeigt ein Vector Businterface vom Typ VN8900 mit einem VN8970 Einschubmodul. Damit kann auf FlexRay-, CAN- und LIN-Netzwerke zugegriffen werden.

Abb. 32: Vector Businterface VN8900, Quelle: Eigene Darstellung.

5.2 Vorbereitung

Um die Automatisierungsschnittstelle implementieren und testen zu können, müssen grundlegende Vorbereitungen getroffen und verschiedene Details überprüft werden. Demnach muss zunächst sichergestellt werden, dass eine Verbindung via Ethernet zwischen Prüfstands-Automatisierungs-PC und Applikations-System-PC besteht. Voraussetzung dafür ist der Anschluss beider Rechner an den Ethernet-Hub und die Vergabe einer IP-Adresse, die beide Rechner im selben Netz vorsieht. Mittels Windows-Kommandozeile (ping-Befehl) ist man in der Lage festzustellen, ob eine Verbindung aufgebaut werden kann. Am Applikations-System-PC muss die Kommunikation mit dem Businterface und anschließend die Verbindung zum Steuergerät geprüft werden. Das Businterface wird über ein USB-Kabel angeschlossen. Für die Kommunikation mit der ECU sind der Anschluss der entsprechenden CAN/FlexRay-Netzwerke, die Installation der Vector Software (CANape inklusive aller notwendigen Zusatzanwendungen), die korrekte Parametrierung des Businterfaces (Bitrate, Kanaldefinition, Initialisierung etc.) und die Erstellung einer CANape Konfiguration erforderlich. Die Bitrate muss der des Steuergeräts entsprechen. Wichtig dabei ist, die Treiberpakete des Businterfaces zu installieren. Ohne diese ist ein Betrieb nicht möglich. Darüber hinaus wird für das Ausführen der Vector Software eine Lizenz benötigt. Diese kann entweder an die Hardware gebunden sein, sprich am Businterface gespeichert sein, oder sich auf einem Lizenz-Stick (License Dongle) befinden. Beim Anschluss der CAN/FlexRay-Netzwerke ist auf das Anbringen der Abschlusswiderstände an den beiden entferntesten Enden des jeweiligen Bussystems zu achten. Bei CAN werden 120 Ohm, bei FlexRay 100 Ohm Widerstände eingesetzt. Diese dienen dem Verhindern von Signalreflexionen.

Im Anschluss daran kann mit der Erstellung eines CANape Projekts bzw. einer Konfigurationsdatei begonnen werden. Dazu wird zunächst ein Projekt erstellt und anschließend eine dazugehörige Konfigurationsdatei angelegt. Dies ist vorerst notwendig um die manuelle Bedatung des Steuergeräts testen zu können. Erst wenn die manuelle Bedatung einwandfrei funktioniert, kann sichergestellt werden, dass die Bedatung auch über die Automatisierungsschnittstelle durchgeführt werden kann, da diese auf die Inhalte des CANape Projekts zugreift. Die Konfiguration ist dabei unter anderem vom Softwarestand des Steuergeräts abhängig. So kann es vorkommen, dass bei einem Projekt unterschiedliche Konfigurationsdateien („.cna“) verwendet werden. Für das Einbinden der entsprechenden Netzwerke sind je nach eingesetztem Bussystem FIBEX- oder DBC- und A2L-Dateien erforderlich, die zur besseren Übersicht im erstellten Projektordner abgelegt werden sollten. Bei FlexRay ist eine FIBEX-Datei

notwendig bei CAN hingegen eine DBC-Datei, da FIBEX lediglich bei FlexRay verbreitet ist. Die A2L-Datei beinhaltet Informationen zu den steuergerätinternen Daten (Kennlinien, Parameter, Kennfelder etc.) und wird für den Lese- und Schreibzugriff (CCP oder XCP) auf diese benötigt. Demnach ist die Kommunikation zwischen CANape und ECU mit einer falschen oder defekten A2L-Datei nicht möglich. Mit temporären Mess- und Verstellfenstern kann überprüft werden, ob eine passende A2L-Datei verwendet wird. Werden dabei plausible Werte von steuergerätinternen Variablen angezeigt und können diese verändert werden, handelt es sich um die richtige A2L-Datei. Somit kann sichergestellt werden, dass die Konfiguration sowie die beiden Schnittstellen ASAM 1 und 2 bis hierhin funktionstüchtig sind. Das ist insofern wichtig, da diese die Grundlage für das ASAM 3 Interface bilden, das wiederum die Basis für die Verbindung zwischen LabVIEW und CANape formt. ASAM 1 dient der Kommunikation zur ECU via CCP oder XCP. Die ASAM 2 Schnittstelle stellt die Beschreibungsdateien bereit.

Mithilfe des ASAP2-Editors können etwaige Fehler einer A2L-Datei behoben werden. Mit diesem ist es möglich, A2L-Dateien zu erstellen, zu verändern sowie übersichtlich darzustellen. Dabei werden die in der A2L-Datei definierten Gruppen, Segmente, Objekte etc. in einer definierten Struktur angezeigt, wie in der nachfolgenden Abbildung dargestellt. Die Gruppen (hier z.B. Arrays) sind vergleichbar mit Ordnern, denen bestimmte Objekte wie Parameter, Messsignale, Kennlinien etc., entsprechend deren Zugehörigkeit zugewiesen werden. Neben diesen Gruppen werden unter anderem noch Umrechnungsregeln, Systemkonstanten und Speichersegmente angeführt. Die auf dieser Abbildung ersichtlichen Objekte (z.B. CALRAM_START) sind keiner Gruppe zugeordnet und werden daher in der Hauptübersicht angezeigt. Dabei ist es möglich verschiedene Attribute wie Name, Adresse, Datentyp, Kommentar etc. des jeweiligen Objekts einzublenden. Diese Attribute werden von CANape genutzt um auf die entsprechenden Objekte zugreifen zu können. Demnach kann über die Automatisierungsschnittstelle direkt über den Namen (z.B. Channel 1) auf ein Objekt zugegriffen werden, da CANape die Zuordnung der symbolischen Namen zu den Adressen übernimmt.

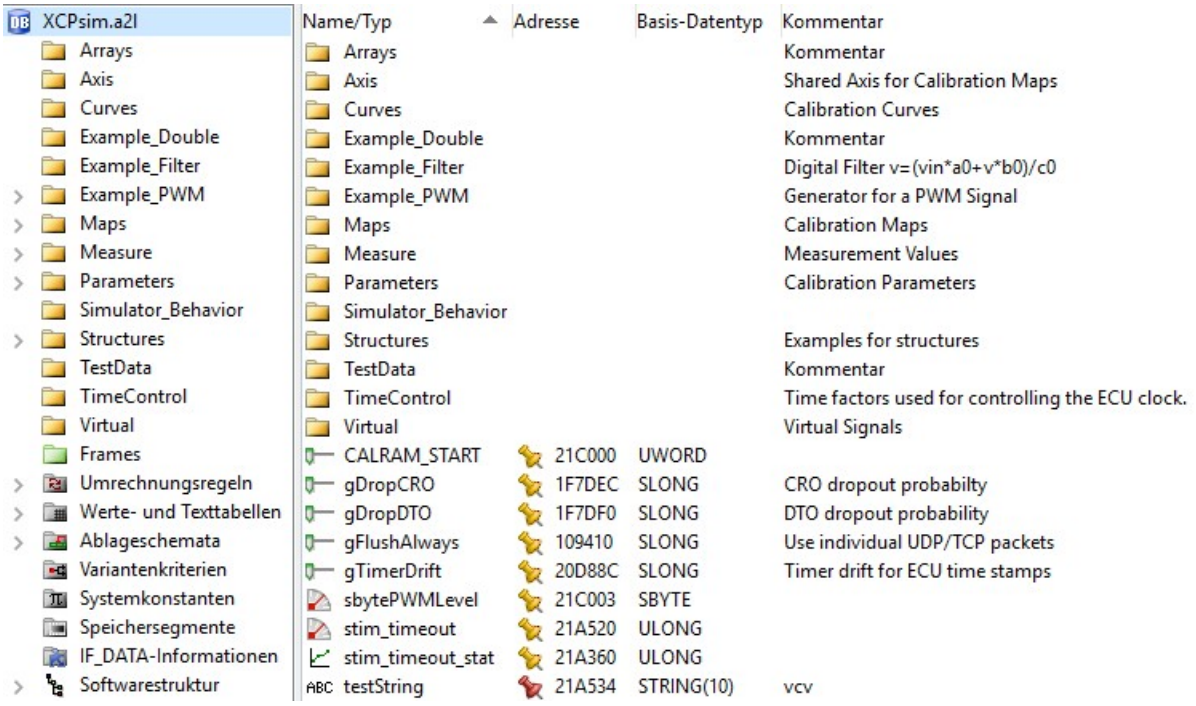


Abb. 33: A2L-Datei, Quelle: Eigene Darstellung.

Bestimmte Botschaften werden hingegen ohne XCP oder CCP via CAN bzw. FlexRay übermittelt. Für diese sind DBC- und FIBEX-Dateien notwendig. Bei diesen Botschaften handelt es sich beispielsweise um Regelungsmodi oder Temperatursignale. Da diese jedoch von der Prüfstandssoftware über das I/O-System vorgegeben und aufgezeichnet werden, dient die Inkludierung im CANape Projekt lediglich der Anzeige und Aufzeichnung der entsprechenden Signale. Werden plausible Werte angezeigt, handelt es sich auch hierbei um die richtigen Beschreibungsdateien. Zur Anzeige kann beispielsweise ein temporäres Trace-Fenster genutzt werden. In diesem werden alle Botschaften am Bus dargestellt. Ohne Beschreibungsdateien werden dabei lediglich Rohdaten eingeblendet. Auch etwaige Error-Frames, die auf bestimmte Fehler im entsprechenden Netzwerk hindeuten, werden im Trace-Fenster hervorgehoben dargestellt.

Die nachfolgende Abbildung (Abb. 34) zeigt eine beispielhafte Gerätekonfiguration in CANape anhand eines Demos (CCPDemo) von Vector. Dabei wird ein Steuergerät (CCPsim) mit einem CAN-Kanal simuliert. Bei den Signalen handelt es sich um CAN- und CCP-Botschaften, die über dieselbe Busleitung (Netzwerk CAN 1) übertragen werden. Eingebunden werden diese jedoch getrennt, da es sich um unterschiedliche Protokolle handelt. Dazu müssen zwei Geräte (CAN und CCP) angelegt werden. Für die CAN-Nachrichten ist eine DBC-Datei („SIM_VN.dbc“) und für die CCP-Botschaften eine A2L-Datei („CCPsim.a2l“) erforderlich. Als Interface dient bei dieser Demoanwendung der Virtual CAN Bus. Dieser kann für verschiedenste Simulationen verwendet werden. Bei einer realen Konfiguration wird an dieser Stelle das entsprechende Businterface (z.B. VN7600 oder VN8900) mit dem verwendeten Kanal (hier CAN 1) dargestellt. Bei Systemen mit XCP-Nachrichten erfolgt das Einrichten ebenfalls mit einer A2L-Datei, die alle erforderlichen Informationen zur Kommunikation beinhaltet. Der Unterschied liegt lediglich im Anlegen des Geräts (XCP).

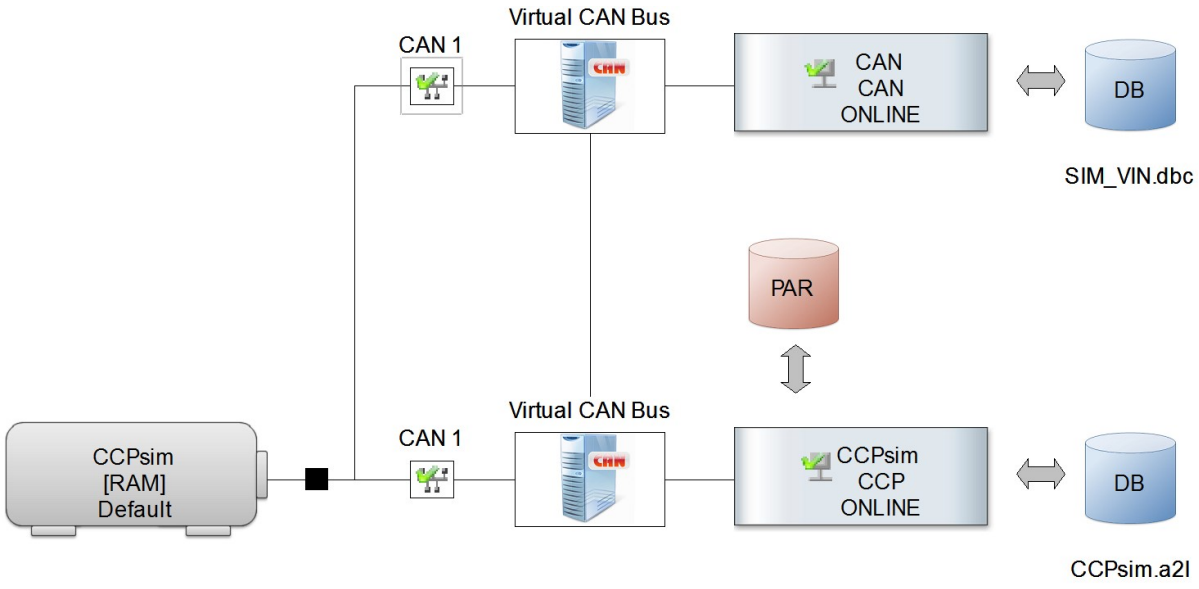


Abb. 34: CANape Gerätekonfiguration, Quelle: Eigene Darstellung.

Sind alle notwendigen Definitionsdateien (A2L, DBC, FIBEX) eingebunden und die Netzwerke funktionstüchtig, kann mit den nächsten und letzten Schritten der Projekterstellung fortgesetzt werden. Diese umfassen das Einfügen von Messfenstern, Signalverläufen und Verstellobjekten. Des Weiteren werden bei Bedarf Skripte erstellt. Diese dienen beispielsweise der automatischen Kalibrierung eines

Steuergeräts oder der Anpassung einer Kennlinie und können während des Betriebs ausgeführt werden. Schlussendlich erfolgt das Einrichten eines Rekorders zur Messdatenaufzeichnung. Es ist auch möglich, mehrere Rekorder zu erstellen. Dazu werden jedem Rekorder ein Name und die zu messenden Signale mit entsprechender Messrate zugewiesen. Die Messrate kann für jedes Signal einzeln eingestellt werden. Dabei darf jedoch die Buslastgrenze nicht überschritten werden. Hohe Messraten und eine Vielzahl an Signalen erhöhen die Busauslastung. CANape stellt die aktuelle Buslast während des Einrichtens dar und erzeugt eine Fehlermeldung sofern die Grenze des jeweiligen Netzwerks erreicht wird. Sollte dies geschehen, müssen die zu messenden Signale oder die Messraten von weniger wichtigen Signalen reduziert werden. Wird keine Fehlermeldung ausgegeben ist der Rekorder somit vollständig eingerichtet. Mit diversen Testmessungen können die Einstellungen und die Funktionsweise des Rekorders überprüft werden. Im Falle, dass diese positiv verlaufen, sind die Vorbereitungen dahingehend abgeschlossen und das CANape Projekt fertiggestellt. Demzufolge ist damit sichergestellt, dass das Clientsystem über die Automatisierungsschnittstelle auf eine funktionstüchtige CANape Konfiguration zugreifen kann und die Bedatung des Steuergeräts durchgeführt werden kann.

Die folgende Darstellung zeigt die Struktur der Kommunikation zwischen Prüfstands-Automatisierungssystem (Upper Level Automation System), CANape (Measurement and Calibration System) und der ECU. Demnach ist erkennbar, dass das Clientsystem (hier LabVIEW) auf CANape zugreift und nicht direkt auf das Steuergerät. Aufgrund dessen ist es essentiell, dass die CANape Konfiguration einwandfrei funktioniert. Zudem sind die drei ASAM Schnittstellen und deren Kommunikationsbereiche abgebildet. Das ASAM 3 Interface bildet die Grundlage für alle Automatisierungsschnittstellen, die in Kombination mit CANape eingesetzt werden können. Aus diesem Grund wird keine bestimmte Automatisierungsschnittstelle, sondern allgemein das ASAM 3 Interface als Verbindung zwischen Prüfstandssystem und CANape dargestellt.

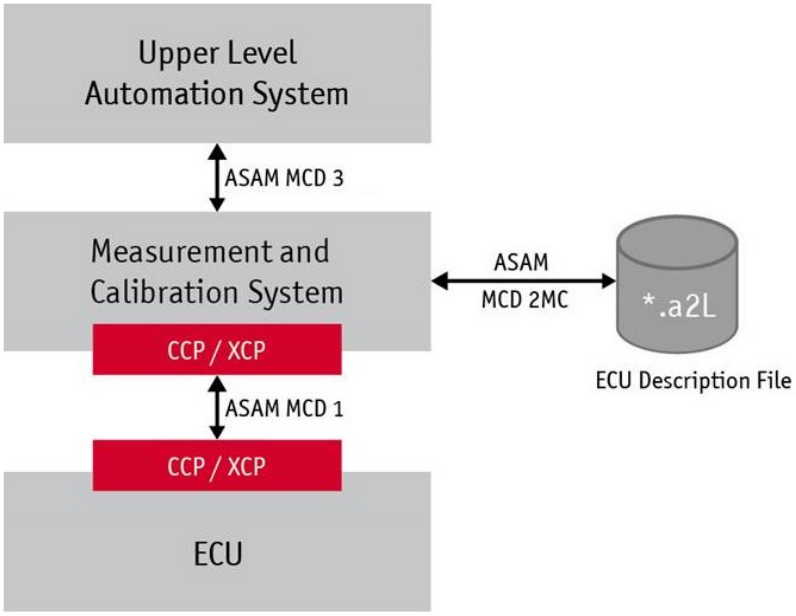


Abb. 35: Struktur der Automatisierungsschnittstelle, Quelle: Knoll/Hendratno/Herzog (2015), S. 3 (leicht modifiziert).

5.3 Inbetriebnahme und Test

Nachdem die Vorbereitungen abgeschlossen wurden und die CANape Konfiguration eine manuelle Bedatung des Steuergeräts erlaubt, kann mit der Inbetriebnahme des CANape COM Interfaces als Automatisierungsschnittstelle zwischen LabVIEW am Vorgaberechner (COM Client) und CANape am Applikations-System-PC (CANape Server) begonnen werden. Dazu muss zuerst die Installation der COM Clientanwendung von Vector am Prüfstands-Automatisierungs-PC durchgeführt werden, da auf diesem Rechner CANape nicht installiert wurde. Diese legt die für die Verbindung erforderlichen CANape Bibliotheken an, auf die das COM-Interface zugreift und damit eine Kommunikation mit CANape ermöglicht. Die folgenden Dateien werden bei der Installation erstellt und registriert: CANapAPI.dll, CANapAPI.lib, CANape.dll, CANaptcp.dll, CANaptcp.lib, CNPControl.dll und UtGetBuffer.dll. Dabei handelt es sich um Bibliotheken für den lokalen Betrieb sowie für Verbindungen via Ethernet (TCP). Die genannten .dll- und .lib-Dateien beinhalten alle Funktionen, Klassen, Objekte und Module der COM-Schnittstelle bzw. des CANapeAPI. Demnach werden der Programmieraufwand und die Komplexität einer Anwendung für das Clientsystem erheblich verringert, da die Programmteile dieser Bibliotheken je nach Bedarf von der jeweiligen Clientanwendung aufgerufen und genutzt werden. Ohne die erwähnten Dateien müsste die Schnittstelle zur Gänze in der gewählten Programmiersprache implementiert werden.

Am Applikations-System-PC ist hingegen keine zusätzliche Installation erforderlich. Die für die COM Schnittstelle erforderliche Serveranwendung (CNPControlServer) ist in der CANape Installation inbegriffen. Für den Betrieb dieser Anwendung müssen die Autostartfunktion aktiviert und ein TCP/IP Port (Voreinstellung: Port 2001) über die CNPControlServer.ini Datei definiert werden. Diese Portnummer muss wiederum am Clientrechner eingetragen werden um eine Verbindung zum Server und damit zu CANape herstellen zu können. Die Voreinstellung wurde nicht verändert, da dieser Port von keiner anderen Anwendung verwendet wird und es daher zu keiner Blockade bzw. Störung bei einer der übrigen Ethernet-Verbindungen der Prüfstandsanlage (z.B. Messsystem oder I/O-System) kommen kann.

Anhand der nachkommenden Abbildung ist die Verbindungsstruktur zwischen Client und Server bei Verwendung des COM-Interfaces ersichtlich. Dabei ist erkennbar, dass die COM-Schnittstelle auf das CANapeAPI zugreift, da über das COM-Interface keine direkte Verbindung zu CANape möglich ist. Folglich dient die COM-Schnittstelle als Interface zwischen Client und CANapeAPI. Die übrigen Automatisierungsschnittstellen in dieser Abbildung dienen lediglich zur Übersicht.

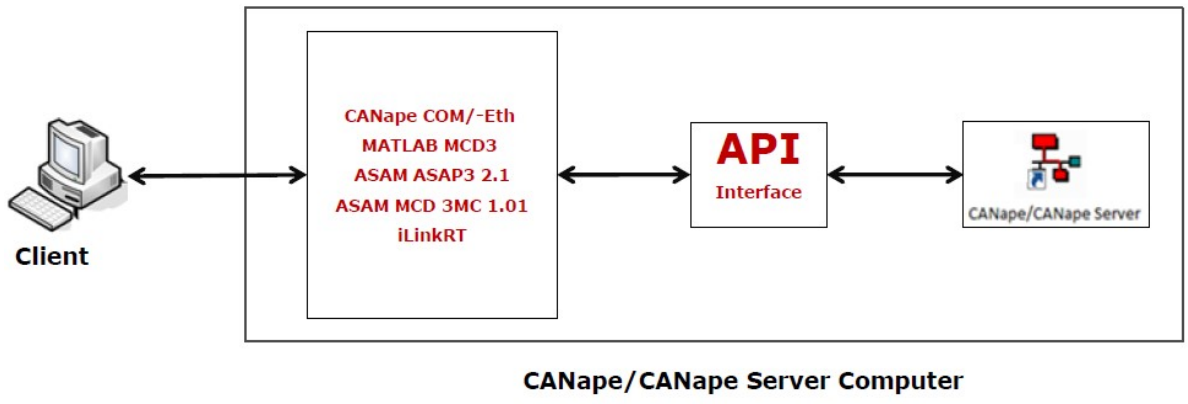


Abb. 36: CANape Automatisierungsschnittstellen, Quelle: Knoll/Hendratno/Herzog (2015), S. 9.

Der nächste Schritt ist, die Automatisierungsschnittstelle vorab zu testen. Dazu stellt Vector verschiedene Beispielanwendungen (Excel, C#, Visual Basic etc.) für den Client und vollständige Demo-Projekte (CCP, XCP etc.) für den Server bereit, die in der CANape Installation inkludiert sind. Mit diesen einfachen Clientanwendungen und Demos ist es möglich, die Verbindung und die grundlegenden Funktionen der Schnittstelle zu testen, ohne einen Schaden an einem Steuergerät verursachen zu können. Dabei können unter anderem CANape Projekte geladen, Messwerte aufgezeichnet, Parameter verändert sowie CANape Instanzen geöffnet bzw. geschlossen werden. Jedoch ist dies nicht mit einer einzelnen Anwendung möglich. Je nach Anwendungsfall (Kalibrierung, Messung, Diagnose oder Monitoring) sind entsprechende Beispielprogramme vorhanden. Die Oberfläche einer solchen Excel-Beispielanwendung für Kalibrierungen (Example Calibration over TCP), zeigt die nachkommende Abbildung (Abb. 37). Als Projekt dient zunächst das bereits erwähnte CCPDemo. Damit eine Verbindung mit dem CANape Server und anschließend mit dem entsprechenden Steuergerät aufgebaut werden kann, um beispielsweise ein Kennfeld auszulesen bzw. zu verändern, müssen die IP-Adresse des Applikations-System-PCs, das Arbeitsverzeichnis des CANape Projekts, der Name der A2L-Datei, der Gerätename, das Protokoll (Driver) und der entsprechende Kanal eingetragen werden. Das Laden einer bestimmten CANape Konfigurationsdatei („.cna“) bzw. die Auswahl des CANape-Verwendungsmodus (z.B. Modalmodus) wird von den Excel-Beispielanwendungen jedoch nicht unterstützt. Es wird lediglich eine CANape Instanz geöffnet, mit der Möglichkeit eine Verbindung mit dem angegebenen Gerät herstellen zu können, indem auf die CANape.ini-Datei in dem eingetragenen Projektverzeichnis zugegriffen wird. Diese .ini-Datei wird im Zuge der Projekterstellung von CANape angelegt und beinhaltet alle grundlegenden Informationen des Projekts, wie beispielsweise Arbeitsverzeichnisse, Beschreibungsdateinamen, Gerätekonfigurationen, Rekordereinstellungen, Messraten, Anzeigeeinstellungen, Skriptverzeichnisse und die Bezeichnungen der Konfigurationsdateien. Werden in einem Projekt mehrere Konfigurationsdateien verwendet, werden deren Dateinamen in der verwendeten Reihenfolge abgelegt (erstgereiht entspricht letztverwendet).

Nachdem kein Fenstermodus ausgewählt werden kann, wird CANape zwar gestartet, jedoch nur ein Write-Fenster geöffnet, das die Vorgänge zwischen Client und Server, wie beispielsweise das Starten einer Messung oder das Abfragen eines Parameters, anzeigt. Demzufolge sind nur begrenzte Testmöglichkeiten mit den Excel-Anwendungen gegeben. Das Öffnen der CANape Instanz inklusive dem Laden des CCP-Simulators und der einzelnen ASAP3 Module nimmt dabei mehr als 13 Sekunden in Anspruch. Damit anschließend eine Kalibrierung durchgeführt und dabei beispielsweise ein Kennfeld angepasst werden kann, muss der symbolische Name des Objekts (hier KF1) in der Excel-Anwendung im dafür vorgesehenen Feld eingetragen werden.

Bei den darauffolgenden Tests konnten verschiedene Objekte ausgelesen bzw. überschrieben werden. Dabei kam es weder zu einem Verbindungsabbruch noch zu einem Fehler. Entsprechend der Einstellung zur Darstellungsart (gelbes Fenster) wird der Inhalt eines Objektes als Rohwert oder als physikalischer Wert angezeigt. Inwiefern die Umrechnung zwischen diesen beiden Anzeigetypen erfolgt, wird aus der A2L-Datei entnommen. In dieser sind verschiedene Umrechnungsregeln abgelegt, die mit den jeweiligen Objekten verknüpft sind.

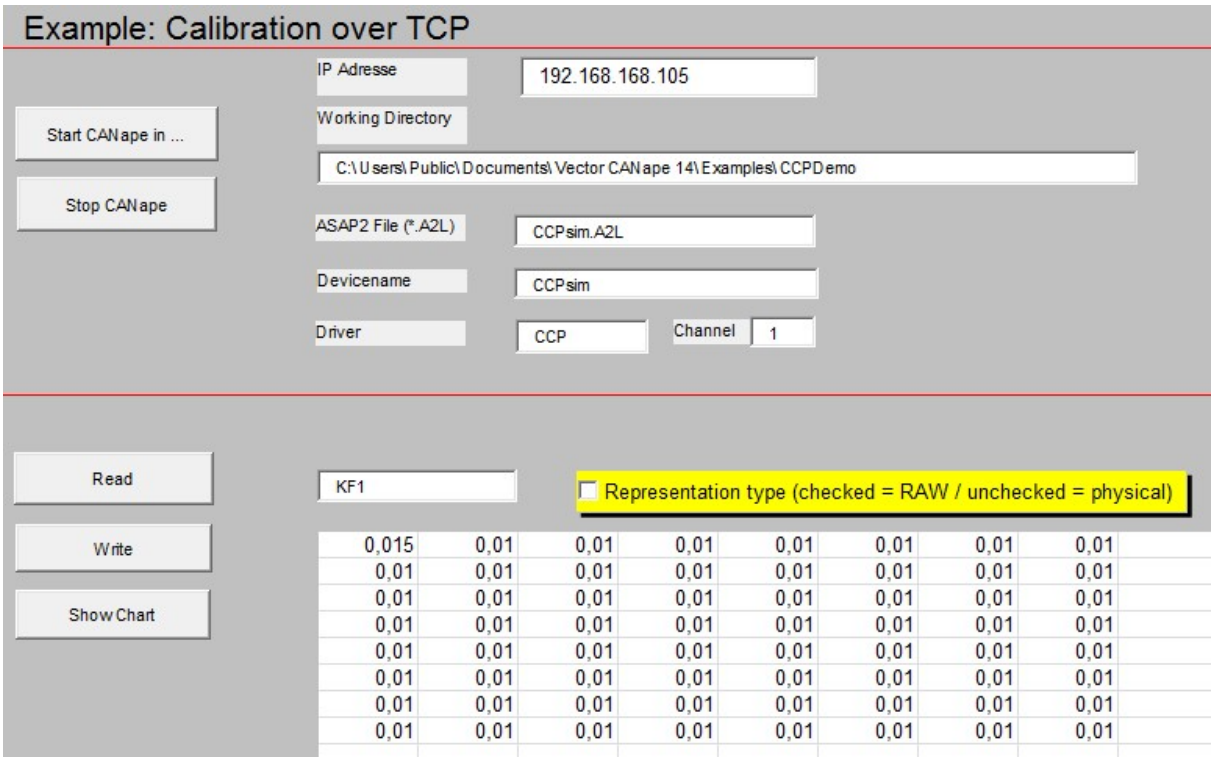


Abb. 37: Excel-Beispielanwendung COM Interface, Quelle: Eigene Darstellung.

Nachdem die Kalibrierung mehrerer Parameter einwandfrei durchgeführt werden konnte, wurde im Anschluss die Excel-Anwendung für Messdatenaufzeichnungen (Example Measurement over TCP) getestet. Dabei müssen ebenfalls die IP-Adresse, Arbeitsverzeichnis etc. eingetragen werden, da die Anwendung grundsätzlich gleich aufgebaut ist. Der Unterschied besteht darin, dass anhand zweier Buttons (anstatt der Read und Write Buttons) nach Eingabe des jeweiligen symbolischen Namens eines Objekts eine Messung gestartet (Start Measurement) bzw. gestoppt (Stop Measurement) werden kann. Im Zuge einer Messung werden die Daten des gewählten Kanals vom Server an die Excel-Anwendung über die COM-Schnittstelle übertragen und dabei in den gewählten Spalten abgelegt. Auch dieser Vorgang konnte ohne Probleme mit verschiedenen Kanälen mehrmals getestet werden. Jedoch besteht bei dieser Excel-Anwendung keine Möglichkeit die Messrate der einzelnen Signale anzuzeigen bzw. anzupassen. Diese wird lediglich der CANape.ini-Datei entnommen und im Programmcode verarbeitet, ohne darauf einen Zugriff zu haben. Das automatische oder manuelle Speichern der gemessenen Daten in einer externen Messdatei ist in dieser Beispielanwendung ebenso nicht vorgesehen. Dennoch konnten eine Vielzahl an Messungen mit unterschiedlichen Kanälen durchgeführt werden.

Somit konnte anhand der genannten Tests mit den bereitgestellten Beispielanwendungen festgestellt werden, dass die COM-Schnittstelle und damit die Kommunikation zwischen COM-Client und Server bzw. CANape problemlos funktioniert. Dementsprechend folgt als nächster Schritt die Excel-Beispielanwendungen mit dem für diese Arbeit erstellten Projekt anstatt der Demoprojekte von Vector zu testen. Mit diesem Projekt ist es möglich auf steuergerätinterne Daten mittels CCP zuzugreifen und CAN-Botschaften der ECU sowie des Messsystems aufzuzeichnen. Dazu wurden zwei CAN-Netzwerke (Netzwerk 1 und 2) inklusive A2L- bzw. DBC-Dateien eingebunden. CAN-Netzwerk 1 wird für die ECU-Daten (CAN und CCP), CAN-Netzwerk 2 für die Messsystem-Nachrichten (CAN) genutzt. Als Businterface dient ein VN7600 von Vector. Für die genannten CAN-Netzwerke wurden die Kanäle CAN 1

und CAN 2 des Businterfaces gewählt. Dabei wurden 500 kBaud (1kBaud = 1kBit/s) als Datenrate bei beiden Kanälen eingestellt, da diese Übertragungsrate bei allen CAN-Netzwerken der Prüfstandsanlage verwendet wird. Damit kann unter anderem die vollständige Übertragung der gesamten Daten des Steuergeräts in der geforderten Geschwindigkeit gewährleistet werden. Auch beim Kanal CAN 2 wurden 500 kBaud als Datenrate gewählt damit sichergestellt werden kann, dass es zu keinem Datenverlust zwischen Messsystem und Businterface bzw. CANape/CANalyzer/CANoe kommt und demnach ein einwandfreier Betrieb der Restbussimulation am Applikations-System-PC garantiert wird.

Einen Ausschnitt der eingebundenen A2L-Datei zeigt die nachkommende Darstellung. Dabei handelt es sich um die Definition eines Objekts, dass für die interne Verarbeitung der ECU verwendet wird. Im Zuge der unterschiedlichen Funktionstests wird unter anderem versucht, über die COM-Schnittstelle auf dieses Objekt zugreifen zu können und den physikalischen Wert des Objekts anhand einer Kalibrierung über die Clientanwendung anpassen zu können. Dazu wird der symbolische Name des Objekts (hier P_trqSetPointMax) genutzt.

```
/begin CHARACTERISTIC P_trqSetPointMax ""
  VALUE 0x8054D0 SWORD_COL_DIRECT 0 T_trq 0 4000
  EXTENDED_LIMITS 0 4000
  BYTE_ORDER MSB_FIRST
  FORMAT "%.3"
  /begin IF_DATA ASAP1B_CCP
    DP_BLOB 0x0 0x8054D0 2
  /end IF_DATA
  /begin IF_DATA CANAPE_EXT
    100
    LINK_MAP "P_trqSetPointMax" 0x8054D0 0x0 0 0x0 0 0x0 0x0
    DISPLAY 0 0 255
  /end IF_DATA
/end CHARACTERISTIC
```

Abb. 38: A2L-Datei Ausschnitt, Quelle: Eigene Darstellung.

Bei den Tests mit dem erläuterten Projekt anhand der CANape Beispielanwendungen konnten erfolgreich Kalibrierungen am Steuergerät durchgeführt und verschiedene Kanäle versuchsweise aufgezeichnet werden. Zunächst wurde versucht steuergerätinterne Parameter via CCP zu kalibrieren bzw. zu verändern. Dies konnte problemlos und einwandfrei durchgeführt werden. Im Anschluss wurden mehrere Kanäle während der Durchführung verschiedener Funktionsprüfungen am Prüfstand anhand der Beispielanwendung für Messungen aufgezeichnet. Auch diese Tests verliefen fehlerfrei. Die dabei ermittelten Latenzzeiten sind jedoch nicht aussagekräftig, da die Performance durch die manuelle Bedienung der Beispielanwendungen erheblich eingeschränkt wird und demzufolge die Verzögerungszeiten verfälscht werden. Verlässliche Ergebnisse in Bezug auf die Latenzzeit können lediglich mit einem Clientsystem, welches eine automatisierte Bedatung des Steuergeräts erlaubt, ermittelt werden. Der Grund für diese Tests lag jedoch nicht primär darin die jeweiligen Latenzzeiten festzustellen, sondern die grundlegenden Funktionen der Schnittstelle an einem realen Steuergerät zu überprüfen ohne dieses zu beschädigen. Darüber hinaus konnten durch diese Tests Informationen gesammelt werden, die bei der Erstellung einer Clientanwendung von Bedeutung sind. Der Zugriff auf ein CANape Projekt und der anschließende Datenverkehr mit einer ECU sind in der COM-Interface

Dokumentation nicht ausreichend beschrieben, da darin die jeweiligen Funktionen nur kurz erwähnt werden und keine detaillierten Informationen zu den verschiedenen Aufrufen vorhanden sind. Mit den genannten Excel-Beispielanwendungen konnten die einzelnen Kommunikationsschritte und Ladevorgänge jedoch nachvollzogen werden. Die geladenen Module werden in der aufgerufenen CANape Instanz im Write-Fenster angezeigt. Dabei wird auch die Zeitdauer mitprotokolliert. Dementsprechend konnte anhand dieses Write-Fensters nachverfolgt werden, welche Funktionen beispielsweise beim Start einer Datenaufzeichnung oder während der Durchführung einer Kalibrierung geladen werden.

Die folgende Abbildung zeigt den Ausschnitt eines Write-Fensters während der Durchführung einer Kalibrierung. Anhand dessen ist einerseits erkennbar, dass das COM-Interface auf den ASAM 3 Standards (vormals ASAP 3) aufbaut, da die von ASAM definierten Funktionsmodule verwendet werden und andererseits welche Funktionen im Zuge eines Kalibriervorgangs aufgerufen werden. Bei den ersten beiden Zeilen handelt es sich um das Auslesen des Objekts (Asap3ReadCalibrationObject) a0. Das Auslesen erfolgt dabei in zwei Schritten. Zuerst wird die X-Achse und anschließend die Y-Achse des Objekts ermittelt. Aus diesem Grund werden beim Auslesen eines Parameters stets zwei Zeilen ausgegeben. Der Inhalt wird dabei an das jeweilige Clientsystem übergeben, jedoch nicht im Write-Fenster angezeigt. Die anschließenden drei Zeilen werden beim Befehl den Inhalt von a0 zu überschreiben ausgegeben. Dabei wird das entsprechende Objekt zuerst ausgelesen und anschließend verändert. Der für die Durchführung einer Kalibrierung notwendige Aufruf lautet Asap3WriteCalibrationObject.

```
Asap3ReadCalibrationObject2(1, a0, 1,0)
Asap3ReadCalibrationObject2(1, a0, 1,0)
Asap3ReadCalibrationObject2(1, a0, 1,0)
Asap3ReadCalibrationObject2(1, a0, 1,0)
Asap3WriteCalibrationObject(1, a0, 1)
```

Abb. 39: Write-Fenster Ausschnitt eines Kalibriervorgangs, Quelle: Eigene Darstellung.

Inwiefern der Ablauf einer Messung im Write-Fenster protokolliert wird, ist in der nachkommenden Darstellung ersichtlich. Dabei wird jedoch lediglich beim Stoppen der Messung die aufgerufene Funktion (Asap3StopDataAcquistion) angezeigt. Bei den übrigen Zeilen handelt es sich um allgemeine CANape-Meldungen.

```
Messung starten
Echtzeitunterstützung: Synchronisation: Keine
1 Rekorder mit Aufzeichnung gestartet
Messung gestartet: 26.9.2016 9:43:21
Asap3StopDataAcquisition()
Messung gestoppt
Messung gestoppt (OK) am  26.09.2016    09:43:23
```

Abb. 40: Write-Fenster Ausschnitt einer Messung, Quelle: Eigene Darstellung.

5.4 Umsetzungsbeispiel

Um die automatisierte Bedatung eines Steuergeräts via LabVIEW testen und beurteilen zu können, ist die Erstellung eines Beispielprogramms erforderlich. Dazu soll eine eigenständige LabVIEW-Anwendung mit Zugriff auf das COM-Interface entworfen werden. Die für die Kommunikation mit dem Server erforderlichen Programmteile werden dabei wie bei den Excel-Beispielanwendungen den installierten CANape Bibliotheken entnommen und genutzt. Mit diesem Clientprogramm soll es möglich sein eine Verbindung zum Server herzustellen, CANape zu öffnen, Projekte bzw. Konfigurationsdateien (.cna) zu laden, Kalibrierungen durchzuführen, Messdateien zu erstellen, Skripte auszuführen und den CANape-Anzeigemodus anzupassen. Über eine übersichtliche Oberfläche soll es möglich sein, die dazu notwendigen Einstellungen vorzunehmen. Nach Fertigstellung des Programms und anschließender Inbetriebnahme werden verschiedene Testläufe durchgeführt, um die entsprechenden Funktionen beurteilen zu können.

Damit ein LabVIEW-Programm erstellt werden kann, muss zunächst ein leeres VI angelegt werden. Im Anschluss daran öffnet sich ein Frontpanel und ein Blockdiagramm. Im Blockdiagramm erfolgt die Programmierung der Anwendung, das Frontpanel dient als Benutzeroberfläche. Bei einem abermaligen Start des erstellten VIs öffnet sich lediglich das Frontpanel, das Blockdiagramm hingegen nicht. Dieses muss eigens geöffnet werden, um Änderungen am Programm vornehmen zu können. Nach dem Ausführen der LabVIEW-Anwendung können keine Anpassungen am Frontpanel bzw. im Blockdiagramm vorgenommen werden. Für den Entwurf des Programms werden die im Zuge der erwähnten Tests gesammelten Informationen und die Vector-Dokumentationsdatei der COM Schnittstelle genutzt. Die Hauptbereiche dieser Beschreibung sowie die Übersicht der verschiedenen Klassen sind im Anhang dieser Arbeit zu finden.

Zu Beginn der Programmerstellung müssen ActiveX-Referenzelemente (Referenz – ActiveX) am Frontpanel angelegt werden. Diese werden mit den jeweiligen Klassen der COM-Schnittstelle verknüpft, um auf die Inhalte der CANape COM Bibliothek (CANAPELib) zugreifen und in weiterer Folge mit dem CANapeAPI für die Verbindung mit CANape kommunizieren zu können. Diese Bibliothek beinhaltet die Klassen (z.B. IApplication2), die für den Datenaustausch und die Kommunikation zwischen COM-Client und CANape Server genutzt werden. Jede dieser Klassen umfasst dabei mehrere Funktionen (z.B. Open2) und Eigenschaften (z.B. IPAddress). Die erforderlichen Klassenparameter werden in der COM-Schnittstelle verarbeitet. In der Folge der Verarbeitung werden entsprechende Befehle an das CANapeAPI übergeben. Über das CANapeAPI wird schlussendlich die Verbindung zur CANape Anwendung über ASAP3 hergestellt. In LabVIEW müssen dazu lediglich die COM-Klassen angelegt, aufgerufen und entsprechende Parameter an die einzelnen Funktionen bzw. Eigenschaftsknoten dieser Klassen übergeben werden. Die weitere Verarbeitung in der Schnittstelle inklusive der Kommunikation mit dem CANapeAPI wird den installierten Bibliotheken entnommen. Folglich muss das CANapeAPI nicht in LabVIEW eingebunden werden. Es genügt dazu die erwähnte Installation und Registrierung der verschiedenen Bibliotheksdateien. Demnach wird anhand dieser Programmbibliotheken die Verbindung bzw. Kommunikation zwischen dem COM-Interface und dem CANape Server am Applikations-System-PC hergestellt.

Die folgende Abbildung zeigt inwiefern die Verknüpfung mit einer Klasse der CANape Bibliothek hergestellt wird. Dazu wurde bereits eine ActiveX-Referenz angelegt. Die Bezeichnung in LabVIEW dafür lautet ActiveX (Referenz). Im Anschluss muss dem ActiveX-Element am Frontpanel eine Klasse zugewiesen werden, indem nach dem Markieren mit der rechten Maustaste die entsprechende Klasse (z.B. CANAPELib.IApplication) ausgewählt wird. Die in der Darstellung ersichtlichen Klassen wurden bereits eingebunden und werden daher bereits vorgeschlagen. Beim erstmaligen Einbinden müssen die Klassen über den Reiter Suchen in der Bibliothek CANape 1.9 Type Library Version 1.9 ausgewählt werden. Nachdem die Klasse der ActiveX-Referenz zugewiesen wurde, kann auf diese und deren Funktionen im Blockdiagramm zugegriffen werden. Sofern es bei der Programmierung notwendig ist, kann mehrmals auf dieselbe Funktion zugegriffen werden. Dementsprechend muss für jede benötigte Klasse im LabVIEW-Programm eine ActiveX-Referenz am Frontpanel angelegt und verknüpft werden.

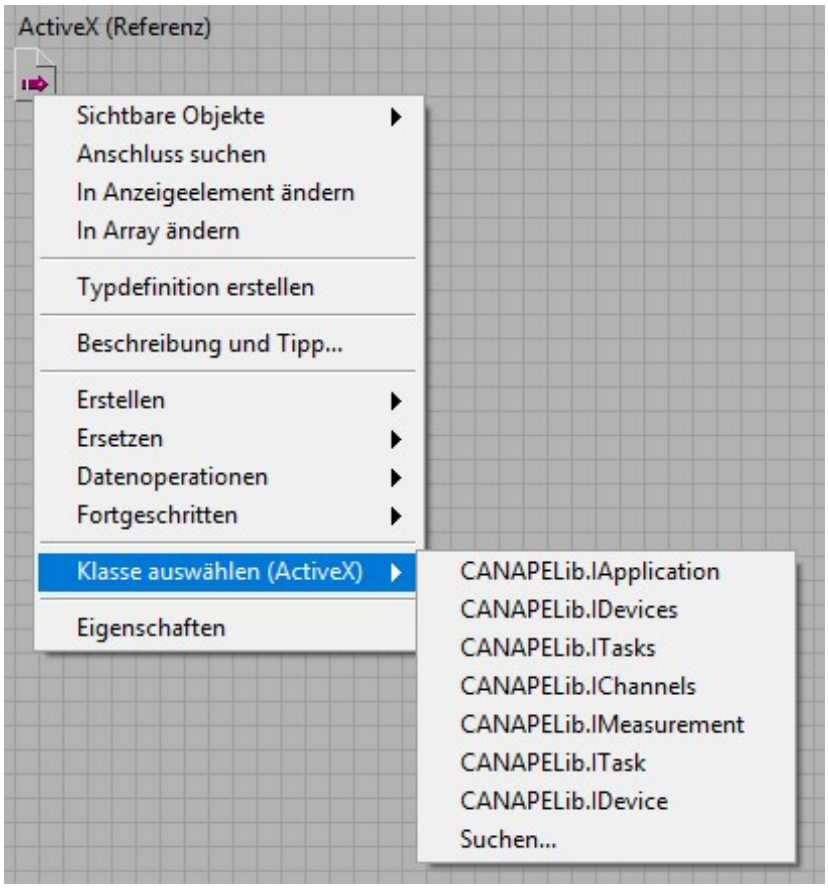


Abb. 41: ActiveX (Referenz), Quelle: Eigene Darstellung.

Welche Klassen für die Erstellung des Programms gemäß den Anforderungen notwendig sind, wurde der COM Schnittstellen Dokumentation entnommen. Demnach werden die Klassen IApplication, IApplication2, IDevice, IDevices2, ITasks, IChannels, IMeasurement und ICalibrationObject benötigt. Mit den in diesen Klassen inkludierten Funktionen sind alle notwendigen Aufrufe durchführbar. Damit kann im Anschluss an das Einrichten der ActiveX-Referenzen mit der Programmierung begonnen werden. Sollte im Zuge der Anwendungserstellung dennoch eine zusätzliche Klasse benötigt werden, muss diese nachträglich eingebunden werden. Ist hingegen eine bereits erstellte Klasse nicht erforderlich, muss die dazugehörige Referenz am Frontpanel gelöscht werden. Andernfalls wäre das Programm fehlerhaft und nicht ausführbar.

Nachdem die erforderlichen Klassen eingebunden wurden, konnte mit der Programmierung der Clientanwendung begonnen werden. In der nachfolgenden Beschreibung werden lediglich die wesentlichen Programmteile, Klassenaufrufe und die erforderlichen Übergabeparameter erläutert. Grundlegende Inhalte der Programmierung (z.B. Schleifen, Abbruchkriterien, Bedingungen etc.) werden nicht erwähnt. Zum Starten einer CANape Instanz wird die Funktion Open2 der Klasse IApplication2 benötigt. Derartige Funktionen werden in LabVIEW als Methoden aufgerufen. Damit die Funktion Open2 erfolgreich ausgeführt werden kann, müssen an diese unter anderem das CANape Arbeitsverzeichnis (workDir), der Debug-Modus (debug), das Timeout in ms (timeout) und der Verwendungsmodus (modalmode) übergeben werden. Sofern die COM-Schnittstelle lokal betrieben wird genügt dieser Aufruf inklusive der erwähnten Parameter. Soll das COM-Interface jedoch über eine Netzwerkverbindung (Ethernet) genutzt werden, muss vor dem Aufruf der Open2 Methode eine Verbindung zum Server hergestellt werden. Dazu sind die Eigenschaften IPAddress, CNPPort und CANapePort der IApplication2 Klasse erforderlich und können dabei in beliebiger Reihenfolge aufgerufen werden, da der Verbindungsaufbau erst gestartet wird, sobald alle erforderlichen Parameter übergeben wurden.

An die Eigenschaft IPAddress muss die IP-Adresse des Servers, an die Eigenschaft CNPPort der TCP/IP Port des Servers und an die Eigenschaft CANapePort der TCP/IP Port des Clients übergeben werden. Anhand der genannten Eigenschaftsknoten und Übergabeparameter kann die Kommunikation mit dem CANape Server aufgebaut werden. Voraussetzung dafür ist, dass die richtigen Parameter übergeben werden und die CNPControlServer Anwendung am Applikations-System-PC ausgeführt wird. Im Anschluss daran kann mit dem Aufruf der bereits erwähnten Funktion Open2 zum Start einer CANape Instanz fortgesetzt werden. Dazu wird auf den Inhalt der CANape.ini-Datei im angegebenen Arbeitsverzeichnis zugegriffen und die darin eingetragene letztverwendete Konfigurationsdatei im gewählten Fenstermodus geladen. Die dafür notwendigen Parameter werden abhängig vom geforderten Datentyp (z.B. Arbeitsverzeichnis = String) über entsprechende Eingabeobjekte am Frontpanel an die Funktion übergeben. Parameter falschen Datentyps können aufgrund der LabVIEW-Syntax ohnehin nicht übergeben werden.

Inwiefern die für den Verbindungsaufbau mit dem CANape Server notwendigen Eigenschaftsknoten und Methoden in LabVIEW aufgerufen werden, ist in der folgenden Darstellung (Abb. 42) ersichtlich. Angelegt werden die Methoden indem das entsprechende Klassenobjekt (wird durch das Anlegen am Frontpanel im Blockdiagramm erstellt) im Blockdiagramm mit der rechten Maustaste angewählt wird und die entsprechende Eigenschaft (z.B. IPAddress) bzw. Methode über Erstellen – Eigenschaft bzw. Methode für CANAPELib.IApplication2-Klasse zugewiesen wird. Damit die genannten Elemente auf die Bibliothekinhalte der IApplication2 Klasse zugreifen können, muss die ActiveX-Referenz (hier CANAPELib.IApplication2) mittels der abgebildeten LabVIEW-Funktion ActiveX-Objekt öffnen geladen werden. Der Ausgang dieser Funktion muss mit dem Eingang der ersten Methode und im Anschluss mit den weiteren Funktionen der Klasse verbunden werden. Damit werden die Inhalte der Klasse und die entsprechenden Übergabeparameter an die nächsten Elemente weitergegeben und von diesen bei der weiteren Verarbeitung genutzt.

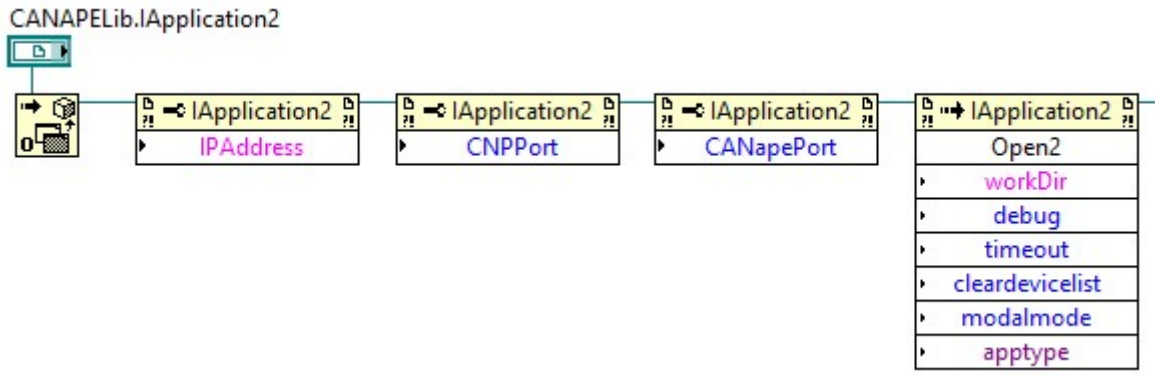


Abb. 42: IApplication2 Klasse, Quelle: Eigene Darstellung.

Im Anschluss an die Funktionsaufrufe zum Verbindungsaufbau zwischen Client und Server muss die Funktion Add2 der Klasse IDevices2 aufgerufen werden. Diese wird für den Zugriff auf ein bestimmtes Gerät (z.B. Steuergerät) der gewählten CANape Konfiguration benötigt. Dazu müssen der Gerätename (DeviceName), der Name der entsprechenden Beschreibungsdatei (DatabaseFilename), der Protokolltyp (driverType), der Kanal (channel) und der Modus (online) übergeben werden. Beim Modus wird zwischen online und offline unterschieden. Online bedeutet, dass beim Laden des Geräts zugleich eine Verbindung zu diesem hergestellt wird. Wird hingegen offline als Modus gewählt, werden lediglich die Gerätedaten der CANape.ini- sowie der A2L-Datei geladen. Für die Bedatung des Geräts über die COM-Schnittstelle muss dieses hinterher über die Funktion GoOnline der Klasse IDevice auf online geschaltet werden. Solange dies nicht geschieht, ist kein direkter Zugriff auf die Daten des Geräts möglich. Es können lediglich Werte im Zwischenspeicher von CANape angepasst werden. Dieser kann, nachdem das Gerät auf online geschaltet wurde, mit den aktuellen Werten im Gerät abgeglichen werden. Dennoch ist es empfehlenswert, das entsprechende Gerät bereits beim Laden auf online zu schalten, damit es zu keinem Datenverlust während dem Offlinemodus kommen kann. Dies könnte zu Fehlfunktionen bzw. Problemen am Steuergerät und damit an der Anlage während des Prüfstandsbetriebs führen und sollte demnach unbedingt vermieden werden, um etwaige Prüflaufausfälle zu verhindern. Die Funktion GoOnline kann auch bei einem späteren Verbindungsabbruch zum Gerät nach dem Ladevorgang, beispielsweise während der Durchführung eines Prüflaufs, für den erneuten Verbindungsaufbau aufgerufen werden.

Um die Funktion Add2 der Klasse IDevices2 nutzen zu können, muss vor dem Aufruf zunächst die Devices Schnittstelle anhand der Eigenschaft Devices der IApplication2 Klasse ausgelesen werden und diese anschließend der IDevices2 Klasse übergeben werden. Dabei wird der Ausgang der bereits angelegten Open2 Methode zum Öffnen der CANape Instanz an den Eingang der Devices Eigenschaft gelegt. Die dazu notwendige LabVIEW-Funktion lautet Variant nach Daten. Damit werden der IDevices2 Klasse alle konfigurierten und verfügbaren Geräte des gewählten CANape Projekts übergeben. Dementsprechend ist es unmöglich fälschlicherweise ein nicht konfiguriertes Gerät zu laden, da nur im Projekt angelegte Geräte gewählt werden können. Im Anschluss daran kann die erwähnte Methode Add2 aufgerufen und ein bestimmtes Gerät (z.B. Steuergerät) für die weitere Verwendung im Programm geladen und angelegt werden.

Das Auslesen der Schnittstelle Devices und die anschließende Übergabe an die Klasse IDevices2 sowie der Aufruf der Add2 Methode sind in der nachkommenden Abbildung ersichtlich. Darüber hinaus sind die dazu erforderlichen Verbindungen zwischen den einzelnen Elementen erkennbar. Derartige Aufrufe werden auf die gleiche Weise bei weiteren Programmabschnitten (z.B. für die Methode Add der Klasse IChannels) benötigt und werden daher in den folgenden Abschnitten nicht ein weiteres Mal näher erwähnt. Es werden lediglich die benötigten Klassen, Methoden und Eigenschaftsknoten zu den unterschiedlichen Programmteilen angeführt und kurz beschrieben. Im Anschluss daran wird der Test der Clientanwendung bzw. der COM-Schnittstelle anhand verschiedener Testläufe erläutert und in weiterer Folge beurteilt.

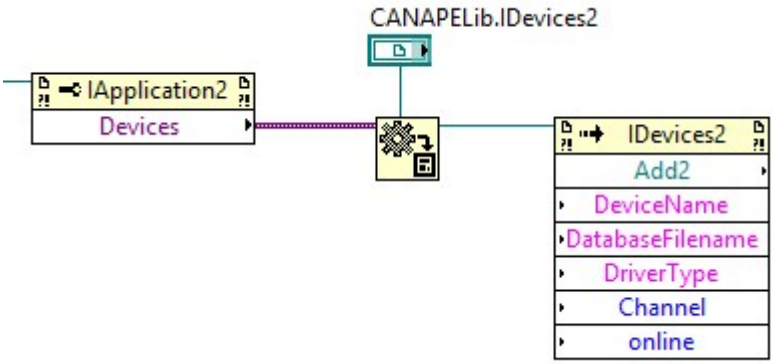


Abb. 43: IDevices2 Klasse, Quelle: Eigene Darstellung.

Neben dem Anlegen eines Geräts dient die Add2 Methode als Grundlage für die Durchführung von Datenaufzeichnungen, Kalibrierungen, Abfragen etc., da diese Methode die IDevice Schnittstelle an die entsprechenden Eigenschaftsknoten und Methoden übergibt. Folglich wird die Add2 Funktion für alle weiteren Programmteile dieses Umsetzungsbeispiels benötigt. Die Reihenfolge dieser Aufrufe ist dabei nebensächlich, da diese unabhängig voneinander genutzt werden und kein Aufruf gleichzeitig ausgeführt wird. Eine Ausnahme bildet dabei die Datenaufzeichnung. Beispielsweise soll es möglich sein eine Datenaufzeichnung zu starten und anschließend während der Messung eine Kalibrierung via CCP oder XCP am Steuergerät durchzuführen, um die dabei resultierenden Parameterverläufe aufnehmen zu können.

Bei der Datenaufzeichnung wird zwischen zwei Modi unterschieden. Einerseits ist es möglich eine Messung am Server direkt via CANape auszulösen sowie die Daten auf diesem abzuspeichern und andererseits die Messdaten an den COM-Client weiterzuleiten, um die Daten auf diesem speichern und zur weiteren Verarbeitung nutzen zu können. Im Zuge des Tests der Schnittstelle werden beide Varianten geprüft. Beide Messverfahren würden den Prüfstandsbetrieb erheblich vereinfachen, da das Starten einer Datenaufzeichnung dadurch keinen manuellen Eingriff erfordert. Allerdings ist es nicht erforderlich die steuergerätinternen Daten bei jeder Messung aufzuzeichnen. Dies ist nur bei bestimmten Funktionsmessungen zwingend notwendig. Für die genannten Messmodi sind verschiedene Methoden und Eigenschaftsknoten der Klassen IChannels, IMeasurement, IRecorder, IRecorders und ITasks erforderlich.

Kalibrierungen werden unter Verwendung der Klassen ICalibrationObjects sowie ICalibrationObject und den dazugehörigen Funktionen Add sowie Read und Write und den Eigenschaften Name, Value und Type durchgeführt. Der Zugriff erfolgt indem die IDevice Schnittstelle und deren Inhalte an die Klasse übergeben werden. Damit ist es möglich auf alle verfügbaren Parameter, der dem Gerät zugewiesenen A2L-Beschreibungsdatei, zugreifen zu können und diese zu verändern bzw. zu kalibrieren. Ob CCP oder XCP für die Kalibrierung genutzt wird, muss bereits bei der Add2 Methode angegeben werden.

Mit den erwähnten Klassen, Methoden und Eigenschaften sind die grundlegenden Funktionen, die für eine aussagekräftige Beurteilung der Schnittstelle notwendig sind, abgedeckt. Weitere Aufrufe wie Statusabfragen oder Funktionen zum Ausführen von CANape-Skripten erfordern das Einbinden von zusätzlichen Methoden sowie Eigenschaften. Diese werden im Folgenden jedoch nicht vollständig beschrieben. Die wichtigsten Funktionen sind NextSample der Klasse ITask zum Auslesen von Messdaten, RunScript der Klasse IApplication zum Ausführen eines Funktionsskripts, LoadCNAFile der Klasse IApplication2 um eine CANape-Konfigurationsdatei zu laden und GoOnline der Klasse IDevice um die Verbindung mit einem Gerät herzustellen. Neben diesen Funktionen werden unter anderem die Eigenschaftsknoten MDFFilename der Klasse IMeasurement zum Festlegen und Abfragen des aktuellen Dateinamens, Save2MDF der Klasse IChannel zum Speichern der gewählten Messdaten in einer MDF-Datei, IsOnline der Klasse IDevice zum Abfragen des Onlinestatus eines Geräts und SamplingTime der Klasse ITask zum Setzen der Abtastrate vor einer Messung und zum Abfragen dieser während einer Datenaufzeichnung. Anhand der genannten Funktionen sowie Eigenschaften ist eine flexible Gestaltung einer Clientanwendung möglich, die alle für den Test der Schnittstelle erforderlichen Bereiche abdeckt.

5.5 Test

Nachdem das LabVIEW-Beispielprogramm fertiggestellt wurde, konnte dieses und im Zuge dessen die COM-Schnittstelle getestet werden. Das Programm beinhaltet alle grundlegenden sowie spezifischen Funktionen, um die Einsatztauglichkeit des Interfaces in Kombination mit LabVIEW im Prüfstandsbetrieb beurteilen zu können. Da es sich dabei um eine Testanwendung handelt, wurde diese nicht in das vorhandene Automatisierungssystem eingebunden. Erst im Anschluss an die Versuchsreihen und der daraus resultierenden Bewertung kann entschieden werden, ob die Schnittstelle den Anforderungen entspricht und die Einbindung in das Vorgabesystem von Vorteil ist oder nicht.

Um die Durchführung der Testreihen zu vereinfachen wurde das LabVIEW-Programm mit einer übersichtlichen Benutzeroberfläche versehen. Über diese können die erforderlichen Übergabeparameter eingetragen werden und einzelne Programmteile beispielsweise zum Verbindungsaufbau oder zur Gerätinitialisierung gestartet werden. Ein Ausschnitt dieser LabVIEW-Oberfläche ist in der anschließenden Darstellung (Abb. 44) ersichtlich. Dabei sind die drei Gruppen Verbindung, Gerät und Messung ersichtlich. Innerhalb dieser Übergabefenster können die für die unterschiedlichen Programmabschnitte erforderlichen Parameter gesetzt werden. Dies geschieht entsprechend des Datentyps via String-Eingabefenster und numerische- bzw. boolesche-Eingabeelemente. Darüber hinaus dient die Benutzeroberfläche als Anzeige verschiedener Parameter. Demzufolge können unter anderem Messdaten des Steuergeräts sowie die Laufzeit der aktuellen Datenaufzeichnung angezeigt werden. Dazu müssen die jeweiligen Kanalbezeichnungen eingetragen und die Messung gestartet werden.

Abb. 44: LabVIEW-Benutzeroberfläche (Ausschnitt), Quelle: Eigene Darstellung.

Zu Beginn der Testläufe wurde der Verbindungsaufbau und das Öffnen der CANape-Instanz betrachtet. Dazu müssen unter anderem die IP-Adresse des Servers und die TCP/IP-Ports an die Clientanwendung übergeben werden. Dabei konnten keine Probleme festgestellt werden. Auch während des Prüfstandsbetriebs war es möglich CANape ohne zusätzliche Zeitverzögerung zu öffnen und das entsprechende Projekt zu laden. Dementsprechend hat der Datenverkehr während eines Prüflaufs keinerlei Auswirkungen auf den Lade- bzw. Startvorgang der CANape-Instanz. Beim Fenster- bzw. Debugmodus kann zwischen minimiert und normal gewählt werden. Beide Varianten wiesen beim Starten keine Fehler auf. Auch die Modalmodi modal und non-modal konnten erfolgreich getestet werden. Zum besseren Verständnis wurden die beiden Modi am Auswahlbutton als Client und Client & Server bezeichnet, da bei der Aktivierung des modal-mode die Bedienung von CANape am Server gesperrt wird. Dagegen ermöglicht der non-modal-mode die simultane (Client und Server) Nutzung der CANape-Instanz. Serverseitig sind jedoch nicht die gesamten Funktionen verfügbar. Das Starten einer Messung ist beispielsweise nicht möglich. Allerdings konnten in diesem Modus Parameter sowohl über die Clientanwendung als auch direkt über die CANape-Anwendung am Server verändert werden. Der Startvorgang über die COM-Schnittstelle konnte demnach erfolgreich und fehlerfrei in allen verfügbaren Kombinationen durchgeführt und getestet werden. Die Dauer des Ladevorgangs ist dabei unabhängig vom gewählten Debugmodus. Die Auswahl des Modalmodus wirkt sich jedoch, wenn auch nur minimal, auf die Dauer des Starts von CANape aus. So nimmt das Laden der CANape Anwendung im modal-mode ~11,5 s (max. Abweichung 1 %) in Anspruch während im non-modal-mode für den selben Vorgang ~11,8 s (max. Abweichung 1 %) benötigt werden. Innerhalb dieses Zeitfensters wird CANape gestartet, das entsprechende Projekt inklusive Konfigurationsdatei aufgerufen und die darin definierten Informationen, wie beispielsweise Rekordereinstellungen, geladen. Demnach ist die Zeitdauer des

Startvorgangs unter anderem abhängig von der Komplexität des entsprechenden CANape Projekts. Dagegen besteht keine Abhängigkeit der Ladedauer gegenüber dem zeitgleichen Ausführen des Prüfstandautomatisierungssystems. Die betrachteten Zeitfenster können prinzipiell als einmalig betrachtet werden, da es ausreicht, CANape vor dem Start einer Prüfung zu öffnen und am Ende wieder zu schließen. Es ist also kein mehrmaliges Öffnen bzw. Schließen der CANape Instanz während eines Prüflaufs erforderlich. Dementsprechend hat die Ladedauer keinen direkten Einfluss auf die Durchführung eines Prüflaufs.

Die folgende Abbildung zeigt den Ausschnitt eines Ladevorgangs im Write-Fenster der geöffneten CANape Instanz. Dabei ist ein Teil der aufgerufenen Funktionen (z.B. Asap3Init) und Module ersichtlich. Darüber hinaus kann der Verbindungsaufbau zwischen Client und Server über TCP nachvollzogen werden.

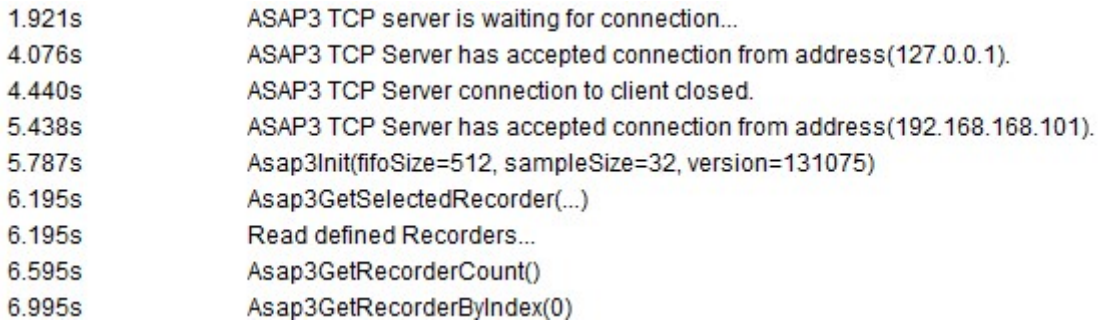


Abb. 45: Write-Fenster Ausschnitt Startvorgang, Quelle: Eigene Darstellung.

Nachdem das Starten einer CANape Instanz und das Laden des angegebenen Projekts mehrmals einwandfrei durchgeführt werden konnte, wurde das Anlegen und Aufrufen eines Geräts getestet. Dabei wurde versucht eine Verbindung mit dem für die Testreihen verwendeten Steuergerät herzustellen und auf dessen interne Daten zugreifen zu können. Dafür sind verschiedene Übergabeparameter wie der Gerätename und der Name der Beschreibungsdatei erforderlich. Entsprechend dieser Parameter wird das Gerät angelegt, geladen und initialisiert. Versuchsweise wurden fehlerhafte Bezeichnungen eingetragen. Dabei wurde eine nachvollziehbare Fehlermeldung ausgegeben und der Ladevorgang abgebrochen. Werden alle notwendigen Parameter fehlerfrei übergeben, wird die Verbindung zum angegeben Gerät hergestellt. Dies konnte mehrmals erfolgreich getestet werden. Dabei betrug die Ladedauer bei allen Wiederholungen ~1,6 s (max. Abweichung 2 %). Der Modalmodus wirkt sich auf diese Zeitdauer nicht aus. Die angegeben Werte gelten demnach gleichermaßen für beide auswählbaren Varianten. Auch der Debugmodus hat keinen Einfluss auf die genannten Zeitwerte. Ähnlich wie beim Öffnen von CANape ist die Zeitdauer des Ladevorgangs eines Geräts ebenfalls als einmalig anzusehen, da das entsprechende Gerät lediglich im Anschluss an den Startvorgang geladen wird und die Verbindung mit diesem bis zum Schließen der Anwendung aufrecht erhalten bleibt. Damit ist der Zugriff auf das Gerät über den gesamten Zeitraum möglich. Ein Verbindungsabbruch konnte bei keinem der unterschiedlichen Tests festgestellt werden. Auch bei längeren Tests über mehrere Stunden blieb die Verbindung zum Steuergerät aufrecht. Aus diesem Grund wurde ein Verbindungsabbruch provoziert indem die Busleitung unterbrochen wurde, um feststellen zu können inwiefern das System reagiert. Daraufhin wurde eine Fehlermeldung ausgegeben. Nach der Wiederherstellung der physikalischen Verbindung konnte das Gerät erneut problemlos und ohne zusätzlichen Zeitverzug geladen werden. Dies

wurde mit den beiden unterschiedlichen Funktionen Add2 sowie GoOnline getestet. Die CANape Anwendung musste dazu nicht neu gestartet werden. Im Clientprogramm konnte die Verbindung zum entsprechenden Gerät über den Eigenschaftsknoten IsOnline abgefragt werden. Folglich kann dieser Status (Bool, True = Online) als Überwachungsparameter im Prüfstands-Automatisierungssystem genutzt werden, um eine Prüfung im Fehlerfall zu stoppen und damit einen möglichen Schaden am Steuergerät oder an der Prüfstandsanlage zu verhindern. Demnach kann sichergestellt werden, dass ein etwaiger Verbindungsabbruch sofort detektiert wird.

Die anschließende Abbildung zeigt den Ladevorgang eines Geräts im Write-Fenster. Dabei sind die einzelnen Übergabeparameter wie Gerätename und Beschreibungsdateiname ersichtlich. Das gewählte Gerät wird entsprechend dieser Parameter angelegt. Im Fehlerfall wird nach der ersten Zeile eine Fehlermeldung ausgegeben und der Ladevorgang abgebrochen. Darüber hinaus kann anhand dieser Darstellung die Zeitdauer des Ladevorgangs nachvollzogen werden. Dementsprechend nimmt das Laden in diesem Fall ab dem ersten Modul (Asap3CreateModule) 1,203 s in Anspruch. Das Modul selbst benötigt 0,396 s zum Laden. Folglich ergeben sich die genannten 1,6 s (exakt 1,599 s).

35.512s	Asap3CreateModule('ECU', 'ECU.a2l', 1, 1, online, undefined)
35.513s	Existing device 'ECU[0]' assigned
35.914s	Asap3GetNetWorkName(0)
36.314s	Asap3GetNetWorkName(0)
36.715s	Asap3GetNetWorkName(0)

Abb. 46: Write-Fenster Ausschnitt Gerät laden, Quelle: Eigene Darstellung.

Das Starten der CANape Instanz inklusive dem Aufrufen des gewählten Projekts sowie das anschließende Laden des notwendigen Geräts dienen als Grundlage für die weiteren Funktionen der COM-Schnittstelle. Demnach wurde im nächsten Schritt die Durchführung von Datenaufzeichnungen getestet. Dabei wurde die Einsatztauglichkeit der zwei erläuterten Varianten, Speichern am CANape Server bzw. Datenübertragung an den Client geprüft. Je nach Anforderung können diese Funktionen zeitgleich oder zumindest die Aufzeichnung am Applikations-System-PC gesondert ausgeführt werden. Dies konnte mehrmals erfolgreich getestet werden. Die beiden Funktionen beeinflussen sich dabei nicht. Beim Speichern via CANape am Server werden die gesamten Einstellungen des CANape-Projekts übernommen und der darin parametrierte Rekorder gestartet bzw. nach Abschluss der Messung gestoppt. Demgegenüber ist es auch möglich, einen Rekorder vollständig über die COM-Schnittstelle anhand verschiedener Funktionen zu parametrieren und anschließend diesen zu verwenden. Dies konnte ebenfalls wiederholt erfolgreich getestet werden. Für die weiteren Tests wurde jedoch der zuvor im Zuge der Projekterstellung parametrierte Rekorder verwendet, da dieser bereits entsprechend der Anforderungen eingestellt wurde. Zunächst wird die Datenaufzeichnung am CANape Server erläutert. Das Starten einer Messung erfolgt durch den Aufruf der Funktion Start. Dabei sind keine Übergabeparameter erforderlich. Nach dem Start der Messung ist es möglich die Datenaufzeichnung zu pausieren (Funktion Pause). In diesem Modus werden die gemessenen Daten lediglich in den voreingestellten Messfenstern in CANape angezeigt. Im Anschluss kann die Datenaufzeichnung anhand der genannten Funktion wieder fortgesetzt werden. Dies konnte mehrfach während einer Messung durchgeführt und erfolgreich getestet werden. Um die Messung schlussendlich zu stoppen, muss die

Funktion Stop aufgerufen werden. Auch hier sind keine Übergabeparameter erforderlich. Nachdem die Messung gestoppt wurde, wird die Messdatei entsprechend der Rekordereinstellungen erstellt und abgelegt. Im Anschluss kann erneut eine Datenaufzeichnung gestartet werden. Das Starten einer Messung nimmt ~1,2 s (max. Abweichung 5 %) in Anspruch. Das anschließende Stoppen der Datenaufzeichnung benötigt ~0,01 s (max. Abweichung 1 %). Nachdem die Messung gestoppt wurde, wird eine Messdatei aus den Daten im Messpuffer erstellt und abgelegt. Die Dauer dieses Schritts ist dabei abhängig von der Größe der Datei. Bevor die nächste Messung gestartet wird, sollte die Verarbeitung der vorherigen Messung abgeschlossen sein. Die verschiedenen Tests ergaben, dass dies jedoch nicht unbedingt notwendig ist, da die Verarbeitung nach Abschluss der nächsten Messung fortgesetzt wird. Allerdings ist es von Vorteil, eine ausreichend lange Stillstandzeit einzuplanen um sicherstellen zu können, dass die Messdatei nach dem Beenden einer Datenaufzeichnung sofort erstellt sowie abgelegt wird und es auf keinen Fall zu einem Datenverlust kommen kann.

Der folgende Write-Fenster Ausschnitt zeigt die Vorgänge nachdem eine Messung gestoppt wurde. Dabei ist die Zeitdauer, die für das Erstellen (0,176 s) und Ablegen (1,074 s) der Messdatei benötigt wird, erkennbar. Die genannten Zeiten gelten jedoch lediglich für diese Beispielmessung. Bei den weiteren Testmessungen ergaben sich abweichende Verarbeitungszeiten.

1m 2.729s	Asap3StopDataAcquisition()
1m 2.729s	Messung gestoppt
1m 2.739s	Messung gestoppt (OK) am 14.10.2016 14:56:22
1m 2.915s	Rekorder 'e3k_3128297449': Messdatei 'D:\Projekte\e3k\Daten\0608_
1m 3.989s	Datei 'D:\Projekte\e3k\Daten\0608_e3k_3128297449_14_14_55.MDF

Abb. 47: Write-Fenster Ausschnitt Messung Stopp, Quelle: Eigene Darstellung.

Gegenüber der Datenaufzeichnung am CANape Server ist die Datenübermittlung an den Client um ein Vielfaches komplizierter, da dazu mehrere Übergabeparameter, Funktionen und Eigenschaftsknoten notwendig sind. Darüber hinaus wird bei diesem Modus die Datenaufzeichnung via CANape als Grundlage für die Datenübertragung an den Client benötigt. Diese muss zumindest im Pause-Modus, sprich ohne Datenspeicherung, betrieben werden, da andernfalls keine Daten vom Gerät an den CANape Server übertragen werden und somit auch keine Datenübermittlung an den Client möglich ist. Des Weiteren können bei dieser Variante lediglich Daten eines zuvor angelegten Geräts aufgezeichnet werden, da die dafür notwendige Schnittstelle Tasks über die Add2 Funktion der IDevices2 Klasse übergeben wird. Folglich muss jedes Gerät, welches bei der Datenübermittlung berücksichtigt werden soll, angelegt und geladen werden. Dies ist bei der Datenaufzeichnung über CANape nicht erforderlich, da bei dieser Variante die benötigten Einstellungen der CANape.ini-Datei entnommen werden und auf die Daten aller darin parametrierten Geräte zugegriffen werden kann. Damit Daten an den Client übermittelt werden, müssen der gewünschte Task (Polling, 2 ms, 10 ms etc.) sowie die entsprechenden Kanalbezeichnungen übergeben werden. Die LabVIEW-Beispielanwendung ist so gestaltet, dass die symbolischen Namen von vier Kanälen eingetragen werden können. Dabei ist es jedoch nicht möglich über die COM-Schnittstelle direkt auf den Inhalt der A2L-Datei am Server zuzugreifen und ein Objekt zur Aufzeichnung auszuwählen, da das COM-Interface auf dem ASAP 3 v2.0 Standard basiert, der dies nicht vorsieht. Folglich müssen die Kanalbezeichnungen am Applikations-System-PC ausgelesen und anhand der Clientanwendung übergeben werden. Sofern die Objektnamen fehlerfrei eingetragen wurden, konnte

die Datenübermittlung an den Client mehrmals erfolgreich durchgeführt werden. Dazu wurden sämtliche verfügbare Tasks (Messtakte) verwendet und getestet. Die auswählbaren Tasks der CCP-Daten sind in der A2L-Datei definiert. Nachdem die Daten an den Client übertragen wurden, konnten diese problemlos weiterverarbeitet und aufgezeichnet werden. Das Starten der Datenübertragung an den Client inklusive dem Startvorgang der Datenaufzeichnung am CANape Server nimmt ~2,6 s (max. Abweichung 5 %) in Anspruch. Im Anschluss konnte die Datenaufzeichnung am Applikations-System-PC über die COM-Schnittstelle testweise pausiert werden. Die Übermittlung der Daten an den COM-Client blieb dabei dennoch aufrecht. Zum Stoppen der Datenübertragung musste zunächst die Datenaufzeichnung am CANape Server und im Anschluss die einzelnen aufgerufenen Funktionen beendet werden.

Wichtig dabei ist auf den FIFO-Puffer (First In – First Out) zu achten. Dieser muss entsprechend der Menge an zu messenden Daten gewählt werden und wird für die Messpufferung benötigt, damit die Daten bei Verzögerungen vorrübergehend bis zur weiteren Verarbeitung zwischengespeichert werden können und es zu keinem Datenverlust kommen kann. Während der Datenübermittlung kann der aktuelle Auslastungsgrad des FIFO-Puffers abgefragt und die Größe des Puffers dementsprechend angepasst werden. Folglich kann dieser Wert auch als Überwachungsparameter genutzt werden. Bei den verschiedenen Tests wurden die Einstellungen des FIFO-Puffers jedoch nicht angepasst, da die Datenübermittlung vom Server an den Client mit den voreingestellten FIFO-Parametern, FifoSize = 512 und sampleSize = 32, fehlerfrei und problemlos durchgeführt werden konnte. Es konnte bei keinem der absolvierten Tests ein Datenverlust festgestellt werden. Sollte es zum Verlust von Messdaten kommen, wird eine Fehlermeldung im Write-Fenster ausgegeben. Auch während der Durchführung eines Prüflaufs konnten keine Beeinträchtigungen bei der Datenübertragung festgestellt werden. Das Prüfstands-Automatisierungssystem und die übrigen Systeme der Prüfstandsanlage wurden durch die Übermittlung der Messdaten zwischen Server und Client ebenfalls nicht gestört. Dementsprechend konnten die Systeme völlig unabhängig voneinander genutzt werden. Dies ist ein wesentlicher Punkt für die mögliche Implementierung der COM-Schnittstelle im bestehenden Vorgabesystem, da ein fehlerloser Betrieb der gesamten Prüfstandsanlage unbedingt gewährleistet werden muss.

Folglich konnte durch diese Tests festgestellt werden, dass die Übermittlung von Messdaten eines Steuergeräts zum CANape Server und anschließend zum COM-Client zur weiteren Verarbeitung (Anzeige, Aufzeichnung, Überwachung etc.) problemlos durchgeführt werden kann. Darüber hinaus konnten während der verschiedenen Versuche keine Störungen an den übrigen Systemen der Prüfstandsanlage erkannt werden.

Die folgende Abbildung zeigt einen Ausschnitt des Write-Fensters während dem Startvorgang einer Datenaufzeichnung am CANape Server. Im Anschluss daran wurde die Speicherung der Daten für den erwähnten Test angehalten. Die Datenübertragung an den Client blieb dabei jedoch aufrecht.

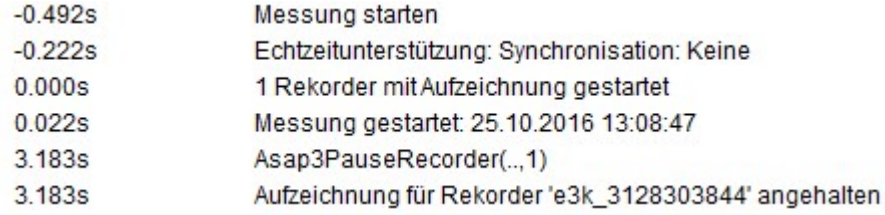


Abb. 48: Write-Fenster Ausschnitt Messung Start/Pause, Quelle: Eigene Darstellung.

Neben den Start- bzw. Initialisierungsvorgängen sowie der Durchführung von Datenaufzeichnungen wurde die Kalibrierung von steuergerätinternen Parametern via CCP mehrfach und anhand verschiedener Objekte erfolgreich getestet. Dazu müssen die symbolische Bezeichnung sowie der Zielwert des entsprechenden Objekts übergeben werden. Wird das Objekt lediglich ausgelesen, genügt die Eingabe des symbolischen Namens. Vor diesen Eingaben muss jedoch CANape gestartet und das entsprechende Gerät geladen und aufgerufen werden. Erst nach Abschluss dieser beiden Schritte konnte eine Kalibrierung durchgeführt werden. Testweise wurde versucht vor diesen Vorgängen eine Kalibrierung zu starten. Diese wurde sofort nach dem Funktionsaufruf abgebrochen, da keine Kommunikation mit dem Steuergerät möglich war. Zusätzlich wurde eine Fehlermeldung nach dem Abbruch ausgegeben.

Sofern das CANape-Projekt somit korrekt geladen und das Gerät erfolgreich initialisiert wurde, konnten problemlos verschiedene Objekte des Steuergeräts kalibriert werden. Sollte der Typ eines Objekts nicht bekannt sein, kann dieser abgefragt und ausgelesen werden. Handelt es sich dabei um ein Messobjekt, so kann dieses nicht kalibriert bzw. verändert werden. Wird dies dennoch versucht, erscheint eine Fehlermeldung und der Kalibriervorgang wird abgebrochen. Testweise wurde versucht derartige Kalibrierungen durchzuführen. Dabei konnten keine fehlerhaften Einträge im Steuergerät generiert werden, da die Vorgänge sofort abgebrochen wurden. Dieser Umstand ist wesentlich für einen sicheren Betrieb des Prüfstand-Automatisierungsystems in Kombination mit der COM-Schnittstelle und muss daher unter allen Umständen garantiert werden. Andernfalls könnte es zu einem Defekt am Steuergerät und damit zu einem Schaden am Prüfobjekt bzw. an der Prüfstandsanlage kommen.

Nachdem die Kalibrierung verschiedener Objekte einwandfrei durchgeführt werden konnte, wurde die Zeitdauer für das Auslesen sowie Verändern eines Kalibrierobjekts untersucht. Dabei wurde zuerst die Dauer einer einzelnen Abfrage bzw. Kalibrierung untersucht. Das Auslesen eines Kalibrierobjekts konnte durchschnittlich innerhalb von ~0,8 s (max. Abweichung 3 %) durchgeführt werden. Bei diesen Tests wurden keine mehrdimensionalen Objekte ausgelesen. Dazu bestand kein Untersuchungsinteresse, da derartige Objekte bei Bedarf ohnehin über entsprechende Skripte angepasst werden. Das direkte Verändern eines Kalibrierobjekts nimmt durchschnittlich ~0,4 s (max. Abweichung 5 %) in Anspruch. Auch hier wurden im Zuge der verschiedenen Versuche keine mehrdimensionalen Kalibrierobjekte des Steuergeräts verwendet.

Der anschließende Write-Fenster Ausschnitt zeigt zwei aufeinanderfolgende Abfragen und die anschließende Kalibrierung des ECU-Objekts P_trqSetPointMax via CCP. Demnach ist erkennbar, dass das Auslesen des genannten Parameters im Verlauf dieses Versuchs innerhalb von 0,78 s abgehandelt wurde. Demgegenüber hat die Durchführung einer Kalibrierung desselben Objekts 0,39 s in Anspruch genommen.

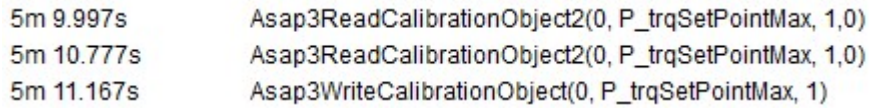


Abb. 49: Write-Fenster Ausschnitt Kalibrierung, Quelle: Eigene Darstellung.

Im Anschluss daran wurde geprüft inwiefern die benötigten Befehle sowie in welcher Zeitdauer mehrfach aufeinanderfolgende Abfragen bzw. Kalibrierungen über die COM-Schnittstelle abgehandelt werden können. Dabei wurde im Zuge dieser Versuchstests festgestellt, dass die Clientanwendung das Auslesen bzw. Verändern verschiedener Kalibrierobjekte (keine mehrdimensionalen) zur gleichen Zeit problemlos übergeben werden können. Beispielsweise konnte die zeitgleiche Kalibrierung mehrerer Objekte über das Clientprogramm ausgeführt werden. Die anschließende Abarbeitung in der COM-Schnittstelle und folglich die Abhandlung am Applikations-System-PC konnte jedoch nicht gleichzeitig durchgeführt werden. Demnach wurden die einzelnen Funktionen in CANape hintereinander aufgerufen und demzufolge die Abfragen bzw. Kalibrierungen einzeln ausgeführt. Dabei konnte kein Ausfall eines Funktionsaufrufs festgestellt werden. Die gesamten, an die Clientanwendung übergebenen, Befehle wurden folglich an CANape übergeben. Nach dem Abschluss aller aufgerufenen Funktionen war es möglich erneut Befehle über die COM-Schnittstelle auszuführen.

Darüber hinaus konnte während dieser Tests festgestellt werden, dass die genannten Vorgänge mit unterschiedlichen Kalibrierobjekten sowie einem einzigen Objekt, welches mehrmals dazu abgefragt bzw. verändert wurde, im selben Zeitrahmen durchgeführt werden konnten. Dementsprechend hat die Auswahl des Kalibrierobjekts, sofern es sich dabei um kein mehrdimensionales Objekt handelt, keinen Einfluss auf die Zeitdauer der Abarbeitung eines Aufrufs.

Anhand des folgenden Write-Fenster Ausschnitts ist die Abarbeitung zeitgleich ausgeführter Abfragen eines Kalibrierobjekts (hier P_trqSetPointMax) ersichtlich. Dabei ist erkennbar, dass das Auslesen des Kalibrierobjekts im Zuge dieses Tests stets ~0,82 s (max. Abweichung 2 %) in Anspruch nahm und aufeinanderfolgend durchgeführt wurde. Auch bei weiteren steuergerätinternen Parametern verliefen die einzelnen Abfragen innerhalb der gleichen genannten Zeitdauer.

2m 25.047s	Asap3ReadCalibrationObject2(0, P_trqSetPointMax, 1,0)
2m 25.867s	Asap3ReadCalibrationObject2(0, P_trqSetPointMax, 1,0)
2m 26.687s	Asap3ReadCalibrationObject2(0, P_trqSetPointMax, 1,0)
2m 27.507s	Asap3ReadCalibrationObject2(0, P_trqSetPointMax, 1,0)
2m 28.337s	Asap3ReadCalibrationObject2(0, P_trqSetPointMax, 1,0)
2m 29.157s	Asap3ReadCalibrationObject2(0, P_trqSetPointMax, 1,0)

Abb. 50: Write-Fenster Ausschnitt Abfragetest, Quelle: Eigene Darstellung.

Der anschließende Write-Fenster Ausschnitt zeigt im Grunde die Abarbeitung desselben Versuchs jedoch mit dem Ziel ein Kalibrierobjekt (ebenfalls P_trqSetPointMax) zeitgleich mehrmals zu verändern. Dazu wurden die benötigten Funktionen nacheinander aufgerufen, da ein zeitgleicher Aufruf nicht möglich ist. Die Zeitdauer für einen einzelnen Aufruf betrug bei diesem Test durchgehend 0,42 s.

34.334s	Asap3WriteCalibrationObject(0, P_trqSetPointMax, 1)
34.754s	Asap3WriteCalibrationObject(0, P_trqSetPointMax, 1)
35.174s	Asap3WriteCalibrationObject(0, P_trqSetPointMax, 1)
35.594s	Asap3WriteCalibrationObject(0, P_trqSetPointMax, 1)
36.014s	Asap3WriteCalibrationObject(0, P_trqSetPointMax, 1)
36.434s	Asap3WriteCalibrationObject(0, P_trqSetPointMax, 1)
36.854s	Asap3WriteCalibrationObject(0, P_trqSetPointMax, 1)

Abb. 51: Write-Fenster Ausschnitt Kalibriertest, Quelle: Eigene Darstellung.

Im Anschluss an die genannten Tests wurde versucht CANape-Skripte am Applikations-System-PC anhand der Clientanwendung auszuführen. Dies konnte mit unterschiedlichen Funktionsskripten erfolgreich durchgeführt werden. Dazu musste der Pfad inklusive dem Namen der Skriptdatei als String übergeben werden (z.B. D:\Projekte\Test\Test1.cns). Dabei ist zu beachten, dass die entsprechenden Skriptdateien zuvor mithilfe des Funktionseditors in CANape manuell kompiliert werden müssen. Andernfalls können diese nicht ausgeführt werden. Dieser Vorgang kann jedoch nicht über das COM-Interface durchgeführt werden. Sollte ein nicht-kompiliertes Skript aufgerufen werden, wird der Vorgang abgebrochen und eine Fehlermeldung ausgegeben. Dies wurde testweise durchgeführt, um feststellen zu können, wie das System auf einen derartigen Aufruf reagiert und ob es dabei möglicherweise zu einem Verbindungsabbruch kommen kann. Die Zeitdauer für das Ausführen eines Skripts ist dabei abhängig vom Umfang der entsprechenden Skriptdatei. Demnach wirkt sich die Anzahl und der Typ der zu veränderten Parameter sowie die Komplexität des Codes auf die benötigte Zeit zur Abarbeitung des Skripts aus.

Nachdem verschiedene Funktionsskripte ebenfalls erfolgreich ausgeführt werden konnten, wurde schlussendlich versucht unterschiedliche CANape-Konfigurationsdateien zu laden. Dies ist bei bestimmten Funktionsmessungen notwendig, die eine spezielle Konfiguration erfordern. Dabei wurde festgestellt, dass eine Konfigurationsdatei problemlos anhand der Clientanwendung über die COM-Schnittstelle geladen werden kann. Jedoch kann dieser Vorgang nicht vollständig automatisiert durchgeführt werden, da beim Laden einer Konfigurationsdatei die bereits geöffnete Konfiguration geschlossen werden muss. Dies geschieht zwar automatisch jedoch wird im Zuge dessen ein Fenster mit einer Abfrage geöffnet, welche beantwortet werden muss. Andernfalls ist es nicht möglich eine alternative Konfigurationsdatei zu laden. Dieses Abfragefenster dient zur etwaigen Speicherung der zu schließenden Konfigurations- sowie Projektdateien. Bei den durchgeführten Tests konnte nicht festgestellt werden, ob es eine Möglichkeit gibt diese Abfrage zu deaktivieren. Darüber hinaus wäre es ohnehin nicht sinnvoll diesen Speichervorgang zu umgehen, da dadurch gegebenenfalls Änderungen an der Konfiguration bzw. dem Projekt verloren gehen würden. Auch eine automatische Speicherung wäre nicht optimal, da es vorkommt, dass temporäre Änderungen an einer Konfiguration in CANape durchgeführt werden, die beispielsweise nach Abschluss einer Messung wieder verworfen werden können. Dementsprechend ist das Laden einer Konfigurationsdatei über die COM-Schnittstelle nicht unter den gewünschten Bedingungen automatisiert durchführbar.

Die nachkommende Abbildung zeigt die beschriebene Abfrage zur Speicherung der Konfigurations- sowie Projektdatei. Solange diese nicht entsprechend beantwortet wird, kann der anschließende Vorgang nicht durchgeführt werden.

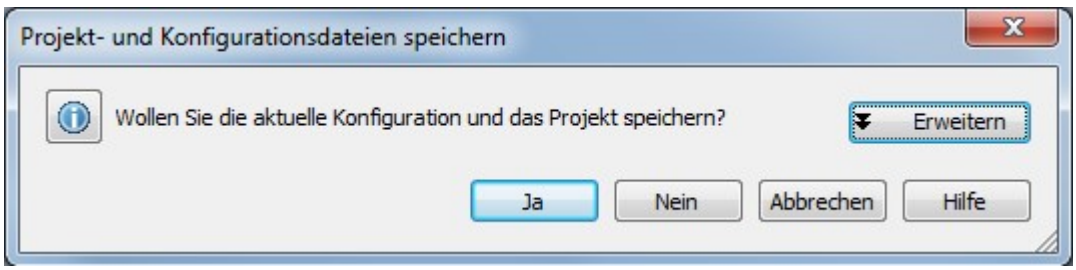


Abb. 52: Konfiguration speichern, Quelle: Eigene Darstellung.

5.6 Beurteilung

In Bezug auf die durchgeführten Funktionstests folgt in diesem Abschnitt die Beurteilung der Einsatztauglichkeit des COM-Interfaces. Dazu werden die Hauptbereiche der Schnittstelle einzeln herangezogen und deren Einsatztauglichkeit für den Prüfstandsbetrieb bewertet. Dementsprechend wird im folgenden Teil geklärt inwiefern die gesetzten Anforderungen an die Automatisierungsschnittstelle durch das gewählte Interface abgedeckt werden können und die automatisierte Bedatung automobiler Steuergeräte durchgeführt werden kann, indem die benötigten Teile (z.B. Startvorgang, Initialisierung, Datenaufzeichnung etc.) des COM-Interfaces in das bestehende Prüfstands-Automatisierungssystem implementiert werden.

Zu Beginn wird der Start- bzw. Ladevorgang eines CANape Projekts inklusive dem Verbindungsaufbau via TCP/IP zwischen Client und CANape Server betrachtet. Nachdem das Starten von CANape und im Zuge dessen das Laden eines Projekts lediglich einmalig vor der Durchführung von Funktionstests am Prüfstand ausgeführt werden muss, ist die dafür benötigte Zeitdauer im Grunde nebensächlich. Dennoch liegt diese in einem angemessenen Rahmen von durchschnittlich 11,8 s (max. Abweichung 1 %). Dementsprechend muss vor dem erstmaligen Start einer Testreihe bzw. einem Prüflauf eine ausreichende Verarbeitungsdauer für den Start von CANape und die Übergabe der erforderlichen Parameter bedacht werden. Der aktive Prüfstandsbetrieb hat dabei keinerlei Auswirkungen auf den Ladevorgang von CANape. Folglich ist dieser Teil der Schnittstelle ohne Beschränkungen für den Einsatz am Prüfstand geeignet, da darüber hinaus bei mehreren Funktionstests keine Fehler bzw. Probleme festgestellt wurden. Der Verbindungsaufbau zwischen Client und CANape Server sowie der anschließende Ladevorgang am Applikations-System-PC konnten stets erfolgreich und innerhalb der genannten Zeitdauer durchgeführt werden. Sollte dennoch ein Problem auftreten, wird eine Fehlermeldung ausgegeben und der Vorgang abgebrochen. Dies ist wesentlich für einen sicheren Prüfstandsbetrieb. Demnach kann dieser Teil der Schnittstelle ohne Bedenken eingebunden und verwendet werden.

Im Anschluss an das Starten von CANape wird der Initialisierungsvorgang eines Geräts bewertet. Dabei handelt es sich ebenfalls um einen einmaligen Vorgang bevor etwaige Tests am Prüfstand durchgeführt werden. Demnach ist die dafür benötigte Zeitdauer (max. 1,6 s) lediglich beim Verbindungsaufbau zu einem bestimmten Steuergerät vor dem Start eines Prüflaufs einzuplanen. Eine wiederholte Initialisierung eines Steuergeräts ist ausschließlich bei einem möglichen Verbindungsabbruch erforderlich. Bei den durchgeführten Tests konnte jedoch zu keiner Zeit ein Kommunikationsabbruch zur ECU festgestellt werden. Folglich ist auch bei diesem Bereich die Einsatztauglichkeit für einen einwandfreien Prüfstandsbetrieb ohne Einschränkungen gegeben. Darüber hinaus liefert dieser Teil der Schnittstelle die Möglichkeit, die Verbindung zum Steuergerät laufend zu überwachen und gegebenenfalls die Kommunikation ohne einen Neustart von CANape neu herzustellen. Diese Überwachung ist entscheidend für die Gewährleistung eines sicheren Prüfstandsbetriebs. Dementsprechend ist bei der Implementierung des Initialisierungsvorgangs in das Vorgabesystem aufgrund der genannten Funktionen keine zusätzliche Überwachung erforderlich. Es ist ausreichend die Parameter der COM-Schnittstelle zu verwenden.

Einer der wichtigsten Bereiche der COM-Schnittstelle ist die Datenaufzeichnung steuergerätinterner Messobjekte. Diese ist bei der Durchführung und der anschließenden Auswertung verschiedener Funktionsmessungen erforderlich. Sowohl die Speicherung am CANape Server sowie die Datenübermittlung an den Client würden den Prüfstandsbetrieb erheblich vereinfachen. Diese beiden Varianten werden im Folgenden betrachtet und dementsprechend eine eventuelle Implementierung in das vorhandene Prüfstands-Automatisierungssystem in Erwägung gezogen bzw. die Einsatztauglichkeit beurteilt. Zunächst wird die Speicherung via CANape und im Anschluss die Übertragung der Messdaten an den Client erläutert.

Bei der Datenaufzeichnung am Applikations-System-PC handelt es sich um die weniger aufwendige Variante in Bezug auf die Implementierung in das Vorgabesystem, da dazu lediglich die drei Funktionen Start, Stop und Pause benötigt werden. Wichtig dabei zu beachten ist, dass der Startvorgang innerhalb von durchschnittlich 1,2 s (max. Abweichung 5 %) durchgeführt wird. Erst nach Ablauf dieser Zeitspanne erfolgt der Start einer Messung. Dementsprechend muss diese Zeitdauer vor dem Beginn einer Prüfung eingeplant werden. Während der unterschiedlichen Funktionstests konnten keine Probleme bzw. Ausfälle bei der Datenaufzeichnung am CANape Server festgestellt werden. Eine etwaige Parametrierung des Rekorders über das COM-Interface ist nicht erforderlich, da diese der erstellten Konfiguration entnommen wird. Folglich ist diese Variante aufgrund der einfachen Implementierung und verlässlichen sowie einwandfreien Funktionsweise ohne Beschränkung für den Einsatz am Prüfstand geeignet. Durch die Implementierung der genannten Funktionen in das aktuelle Vorgabesystem wäre das manuelle Eingreifen zum Start einer Messung via CANape somit nicht mehr erforderlich. Demnach wäre die Anforderung nach einer automatisierten Datenaufzeichnung ECU-interner Daten (hier CCP) anhand dieser Variante vollständig abgedeckt.

Demgegenüber ist die Datenübermittlung an den Client weitaus komplexer, da dazu mehrere Funktionen aufgerufen, Parameter übergeben sowie Eigenschaften ausgelesen werden müssen. Dementsprechend gestaltet sich die Implementierung aufwendiger als die vorhin genannte Variante. Beispielsweise müssen die entsprechenden Objektnamen übergeben werden, da diese nicht direkt anhand der COM-Schnittstelle ausgelesen werden können. Auch Parameter wie die Abtastrate werden von den entsprechenden Funktionen benötigt. Darüber hinaus muss die passende Größe des FIFO-Puffers bedacht werden. Diese ist abhängig von der Menge an zu übermittelten Daten und wesentlich für einen ausfallsicheren Betrieb. Neben den bereits genannten Bedingungen für diese Variante der Datenaufzeichnung muss ohnehin die Messung am CANape Server zumindest im sogenannten Pause-Modus aktiviert sein. Andernfalls können keine Daten an den Client übertragen werden. Dementsprechend muss eine Zeitdauer von 2,6 s (max. Abweichung 5 %) vor dem Start einer Prüfung eingeplant werden. Dennoch konnten bei den durchgeführten Funktionstests keine Ausfälle bzw. Fehler bei der Datenübertragung zwischen CANape Server und Client festgestellt werden. Auch der aktive Prüfstandsbetrieb hatte keinen Einfluss auf die Datenverbindung. Daher ist auch bei dieser Variante eine Einsatztauglichkeit am Prüfstandssystem ohne Einschränkung gegeben. Darüber hinaus können in diesem Modus steuergerätinterne Messobjekte (z.B. Fehlercodes) als Überwachungsparameter im Vorgabesystem genutzt werden um Störungen an der ECU frühzeitig erkennen und mögliche Schäden vermeiden zu können. Folglich sollten beide Varianten implementiert werden und je nach Anforderung auf diese zugegriffen werden.

Einer der Hauptgründe für den Einsatz einer Automatisierungsschnittstelle war der Wunsch nach einer automatisierten Kalibrierung steuergerätinterner Objekte. Dementsprechend wird im Folgenden die Kalibrierung (hier CCP) verschiedener Elemente über die COM-Schnittstelle und eine mögliche Implementierung der dazu erforderlichen Funktionen betrachtet sowie deren möglichen Einsatz im aktuellen Prüfstands-Automatisierungssystem beurteilt. Im Zuge dessen werden das Auslesen bzw. Verändern eines Kalibrierobjekts untersucht. Beide Funktionen setzen voraus, dass CANape gestartet, die entsprechende Konfiguration geladen, das Steuergerät initialisiert sowie eine Verbindung zu diesem hergestellt wurde. Diese genannten Schritte werden im Folgenden nicht mehr beschrieben sowie beurteilt.

Für das Auslesen eines Kalibrierobjekts wird lediglich eine Funktion und ein Übergabeparameter, der symbolische Name des entsprechenden Elements, benötigt. Dementsprechend kann eine derartige Abfrage mit geringem Aufwand am Client implementiert werden. Darüber hinaus handelt es sich dabei um einen verlässlichen Funktionsaufruf, da bei den durchgeführten Tests kein Problem bzw. Abbruch detektiert werden konnte. Die Zeitdauer für das Auslesen eines einzelnen eindimensionalen Kalibrierobjekts über CCP lag im Zuge dieser Funktionstests durchschnittlich bei 0,8 s (max. Abweichung 3 %). Diese Zeitdauer muss bei einer etwaigen Abfrage vor oder während der Durchführung von Prüfläufen bedacht werden. Beim zeitgleichen Auslesen mehrerer Kalibrierobjekte summiert sich die Zeitdauer eines einzelnen Aufrufs je nach Anzahl der auszulesenden Elemente, da die Abarbeitung über die COM-Schnittstelle nicht zeitgleich erfolgt. Nachdem die Abfrage trotz der etwas hohen Zeitdauer schneller über die COM-Schnittstelle als durch einen manuellen Eingriff am Applikations-System-PC durchgeführt werden kann, ist auch dieser Teil des Interfaces für den Einsatz am Prüfstand geeignet. Insbesondere, da es sich bei diesem Vorgang um einen zuverlässigen Funktionsaufruf handelt, der auch zeitgleich mehrfach aufgerufen sowie ohne großen Aufwand in das bestehende Vorgabesystem anhand einer einzelnen Funktion (ICalibrationObject – Read) eingebunden werden kann.

Zum Verändern eines Kalibrierobjekts ist ebenfalls lediglich ein Funktionsaufruf inklusive der Übergabe des symbolischen Objektnamens und des Zielwerts erforderlich. Folglich ist die Implementierung dieses Teils der Schnittstelle ebenso unkompliziert. Die Abarbeitung verläuft durchschnittlich innerhalb einer Zeitdauer von 0,4 s (max. Abweichung 5 %) und liegt damit zwar nicht im gewünschten jedoch in einem akzeptablen Zeitrahmen. Zudem ist dieser Teil der Schnittstelle ebenso durchaus verlässlich, da bei den verschiedenen Tests zu keiner Zeit ein Ausfall im Zuge einer Kalibrierung festgestellt werden konnte. Auch bei mehrfach zeitgleich ausgeführten Kalibrieraufrufen kam es zu keinen Problemen bei der Abarbeitung. Nachdem die einzelnen Funktionsaufrufe bei gleichzeitiger Anforderung wie beim Auslesen eines Objekts hintereinander in gleichen Zeitabständen ausgeführt werden, summiert sich auch hier die Zeitdauer entsprechend der zu veränderten Kalibrierobjekte. Demnach muss die resultierende Zeitdauer bei der Durchführung von Funktionsmessungen bedacht und eingeplant werden. Die Einsatztauglichkeit für den Prüfstandsbetrieb ist demzufolge auch bei diesem Teil der Schnittstelle gegeben, da es sich dabei um einen ausfallsicheren Vorgang handelt, der mit geringem Aufwand in das aktuelle Prüfstands-Automatisierungssystem integriert werden kann. Die einzige Einschränkung bei der Verwendung dieser Funktion ist die erforderliche Zeitdauer für die Durchführung einer Kalibrierung, die für einen flexiblen Prüfstandsbetrieb geringer ausfallen sollte.

Schlussendlich werden in den folgenden Abschnitten die Ausführung von Funktionsskripten sowie der Ladevorgang einer CANape-Konfigurationsdatei betrachtet. Dazu werden die Integrierung der erforderlichen Funktionen in das Vorgabesystem sowie deren Einsatztauglichkeit in Bezug auf die durchgeführten Tests beurteilt. Auch bei diesen beiden Teilen der Schnittstelle wird vorausgesetzt, dass zumindest CANape bereits gestartet und ein Projekt geöffnet wurde. Darüber hinaus setzt das Ausführen eines Funktionsskripts voraus, dass eine Konfiguration geladen sowie die Kommunikation mit dem Steuergerät hergestellt wurde. Andernfalls können keine direkten Änderungen an den steuergerätinternen Objekten vorgenommen werden, da in diesem Fall lediglich auf den Zwischenspeicher des Projekts zugegriffen wird. Darüber hinaus müssen die benötigten Skriptdateien im Voraus via CANape kompiliert werden.

Bei der Durchführung bestimmter Funktionsmessungen ist das Laden verschiedener Funktionsskripte erforderlich. Dementsprechend könnte durch das automatisierte Ausführen derartiger Skripte die Auslastung der Prüfanlagen erheblich erhöht werden. Die Implementierung und Bedienung dazu gestaltet sich unkompliziert. Es wird lediglich ein Funktionsaufruf sowie ein Übergabeparameter, der Pfad inklusive dem Dateinamen des gewählten Funktionsskripts, zur weiteren Verarbeitung benötigt. Die Zeitdauer für die Ausführung eines Skripts hängt vom Inhalt (Anzahl der Parameter, Komplexität des Programmcodes etc.) der Datei ab und muss daher einmalig anhand einer manuellen Ausführung festgestellt werden. Ein direkter Einfluss auf die Durchführung einer Prüfung ist jedoch nicht gegeben, da die entsprechenden Skripte stets vor dem Start einer Messung ausgeführt werden müssen. Im Zuge des Tests der COM-Schnittstelle wurden mehrfach unterschiedliche Funktionsskripte aufgerufen. Diese wurden vollständig und problemlos ausgeführt. Es wurde zu keiner Zeit ein etwaiger Verbindungsabbruch oder Fehler detektiert. Folglich handelt es sich auch bei diesem Teil der Schnittstelle um einen verlässlichen sowie ausfallsicheren Funktionsaufruf, welcher mit geringem Programmieraufwand implementiert werden kann. Die Einsatztauglichkeit ist damit grundsätzlich ohne Einschränkungen gegeben. Es ist lediglich darauf zu achten, dass die Kompilierung der entsprechenden Skripte nicht über das COM-Interface durchgeführt werden kann und demnach einen manuellen Eingriff am Applikations-System-PC (CANape – Funktionseditor) erfordert.

Neben dem Ausführen unterschiedlicher Funktionsskripte erfordern bestimmte Funktionsmessungen zusätzlich das Laden spezieller Konfigurationen. Dementsprechend wurde im Laufe der durchgeführten Tests versucht unter verschiedenen Konfigurationsdateien zu wechseln. Dabei wurde jedoch festgestellt, dass dieser Vorgang nicht vollständig automatisiert ausgeführt werden kann, da eine Abfrage nach dem Schließen der geladenen CANape-Konfiguration beantwortet werden muss. Wird diese über die Benutzeroberfläche nicht entsprechend quittiert, kann der Vorgang nicht fortgesetzt werden. Folglich entspricht dieser Bereich der Schnittstelle nicht den gesetzten Anforderungen. Somit ist die Einsatztauglichkeit am Prüfstand zwar unter der genannten Einschränkung grundsätzlich gegeben, jedoch ist die Flexibilität für die Verwendung dieses Funktionsaufrufs im Prüfstandsbetrieb nicht ausreichend. Demnach muss das Wechseln einer Konfigurationsdatei weiterhin manuell durchgeführt werden. Nachdem derartige Ladevorgänge jedoch ohnehin selten durchgeführt werden müssen, ist diese Bedienungseinschränkung nicht ausschlaggebend für die Entscheidung über die mögliche Implementierung der COM-Schnittstelle im vorhandenen Prüfstands-Automatisierungssystem.

Nachdem die einzelnen Bereiche der gewählten Automatisierungsschnittstelle betrachtet und im Zuge dessen beurteilt wurden, folgt nun eine kurze Zusammenfassung der erforderlichen Teilbereiche für den Einsatz am Prüfstand sowie eine Einschätzung zur möglichen Implementierung der COM-Schnittstelle im vorhandenen Vorgabesystem. Dazu wird kurz auf die Möglichkeiten eingegangen, die sich durch den Einsatz des Interfaces ergeben und anschließend die Integration betrachtet.

Anhand der durchgeführten Tests sowie der anschließenden Beurteilung wurde festgestellt, dass die Inhalte der COM-Schnittstelle eine automatisierte Bedatung eines Steuergeräts erlauben und beinahe die gesamten Anforderungen abgedeckt werden. Demnach ermöglicht der Einsatz des COM-Interfaces das Starten der CANape-Anwendung inklusive dem Laden eines zuvor erstellten Projekts, das Initialisieren eines Geräts (z.B. Steuergerät), das Speichern von Messdaten am CANape Server sowie das Übertragen dieser an den Client zur weiteren Verarbeitung, das Durchführen von Kalibrierungen über CCP (Auslesen und Verändern) und das Laden von Funktionsskripten. Diese genannten Vorgänge können innerhalb einer angemessenen Zeitdauer durchgeführt werden, die bei der Verwendung entsprechend eingeplant und bedacht werden muss. Lediglich das Wechseln einer geladenen Konfigurationsdatei ist nicht unter den gewünschten Anforderungen durchführbar, da dieser Vorgang einen manuellen Eingriff am Applikations-System-PC erfordert und demnach nicht zur Gänze automatisiert abgehandelt werden kann. Entsprechend der genannten Eigenschaften ist eine massive Steigerung der Prüfstandsauslastung durch die Implementierung der COM-Schnittstelle zu erwarten, da dadurch manuelle Eingriffe grundsätzlich nicht mehr erforderlich sind und damit ein durchgängiger 24-Stundenbetrieb ermöglicht wird. Bei etwaigen Verbindungsabbrüchen sowie sonstigen Fehlerfällen wird jedoch weiterhin ein manueller Eingriff notwendig sein, um beispielsweise die Server-Anwendung neu zu starten.

Die Implementierung der erforderlichen Teile der Schnittstelle gestaltet sich unterschiedlich komplex. Jedoch sind bei der Programmierung der einzelnen Bereiche keine Schwierigkeiten aufgetreten. Dementsprechend können die erforderlichen Funktionen sowie Eigenschaften der COM-Schnittstelle via LabVIEW problemlos eingebunden und verwendet werden. Auswirkungen auf den übrigen Datenverkehr innerhalb des Prüfstandssystems sind durch den Einsatz der Automatisierungsschnittstelle zu keiner Zeit gegeben. Folglich ist die Einsatztauglichkeit für die genannten Teilbereiche, bis auf das Laden einer Konfigurationsdatei, gegeben. Es ist lediglich auf wenige Einschränkungen und auf die Zeitdauer der einzelnen Funktionen zu achten.

6 MÖGLICHKEITEN DURCH CAN FD

Die Weiterentwicklung des klassischen CAN zum optimierten CAN mit flexibler Datenrate, kurz CAN FD, hat eine Vielzahl an Erneuerungen mit sich gebracht. Kernbereiche des Protokolls wie die Nutzdatenlänge, die Datenrate sowie die Fehlererkennung wurden aufgrund des steigenden Datenverkehrs in der Automobilelektronik überarbeitet und verbessert. Damit wurden in jenen Bereichen, in denen sich durch die begrenzte Datenrate von CAN und die verhältnismäßig hohe Komplexität von FlexRay Engpässe ergaben, neue kosteneffiziente Vernetzungslösungen geschaffen. Folglich wurde auf die veränderten Anforderungen an Bussysteme in der Fahrzeugtechnik reagiert.

Nun stellt sich einerseits die Frage was für den Einsatz von CAN FD spricht und andererseits inwiefern sich ein Umstieg auf dieses Protokoll lohnt. Dies soll in diesem Kapitel anhand der erarbeiteten Informationen im Theorieteil erläutert werden. Dazu werden die Bussysteme CAN, CAN FD und FlexRay untereinander verglichen sowie deren Vor- und Nachteile in Bezug auf verschiedene Anwendungsgebiete dargestellt. Darüber hinaus soll die mögliche Ablöse des komplizierten FlexRay-Systems durch CAN FD betrachtet werden. Zusätzlich wird kurz die benötigte Hardware und der Einsatz von CAN FD in der Prüfstandstechnik erläutert.

6.1 Gegenüberstellung

Zur Gegenüberstellung der drei Bussysteme CAN, CAN FD und FlexRay werden deren Hauptmerkmale Botschaftsaufbau, Zugriffsverfahren, Fehlererkennung bzw. -behandlung sowie Kommunikationstechnik und Realisierung verglichen. Diese Informationen werden für die nachfolgenden Abschnitte genutzt um die Vor- bzw. Nachteile der Bussysteme darstellen und entsprechende Anwendungsgebiete beurteilen zu können.

6.1.1 Botschaftsaufbau

In Bezug auf den Botschaftsaufbau unterscheiden sich CAN und CAN FD nur geringfügig. Das Botschaftsformat bei FlexRay ist zu den beiden CAN Protokollen hingegen völlig verschieden. Lediglich die grundsätzliche Einteilung in Header, Payload und Trailer ist identisch. Bei CAN ist das Nutzdatenfeld mit maximal 8 Byte beschränkt. Gesichert werden die Daten mit einem 15 Bit CRC. Bei CAN FD können dagegen bis zu 64 Byte übertragen werden. Je nach Nutzdatengröße wird ein CRC mit maximal 21 Bit eingesetzt. Neben wenigen zusätzlichen Steuerbits und dem größeren Nutzdatenfeld ist der größte Unterschied zwischen CAN und CAN FD, dass die Bitrate während der Übertragung der Nutzdaten und der CRC-Prüfsumme verändert werden kann. Damit sind Datenraten von bis zu 4 MBit/s statt 1 MBit/s bei CAN möglich. Daneben ergeben sich durch die Möglichkeit der Bitratenumschaltung bei CAN FD markant höhere Nutzdatenraten als beim gewöhnlichen CAN-Protokoll.

Die folgende Abbildung (Abb. 53) veranschaulicht die Unterschiede zwischen CAN und CAN FD in Bezug auf die erhöhte Bitrate sowie auf das vergrößerte Datenfeld. Dazu sind die Anpassung der Bitrate zwischen den Arbitrierungsphasen sowie die Erweiterung des Nutzdatenfeldes von 8 auf 64 Byte in vereinfachter Weise veranschaulicht.

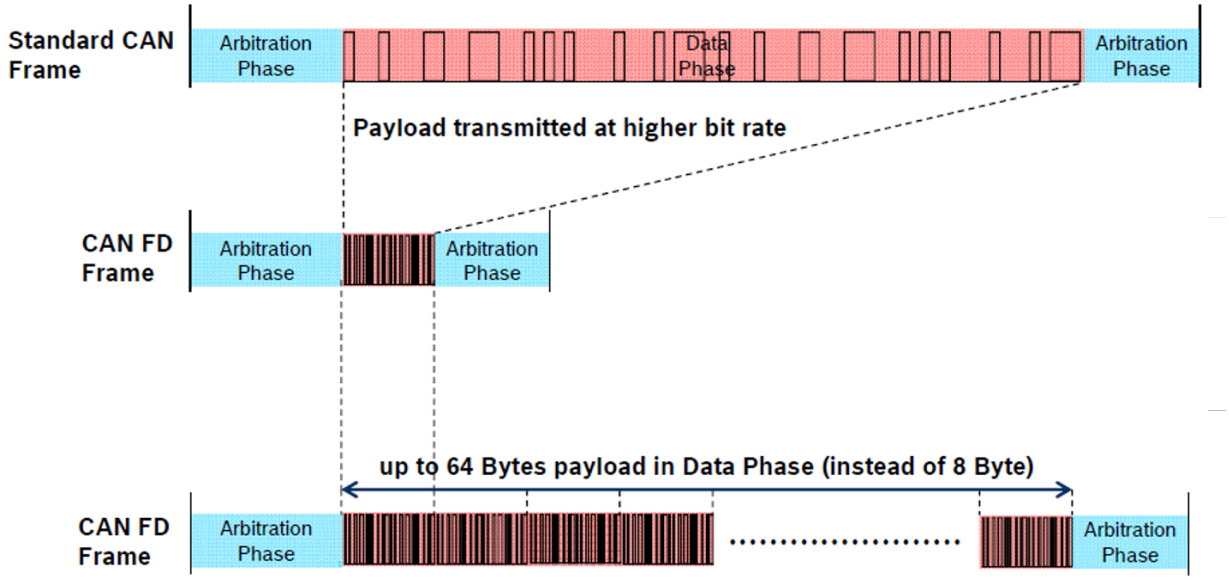


Abb. 53: CAN/CAN FD Vergleich, Quelle: Lindenkreuz (2012), Online-Quelle [29.April.2016], S. 5 (leicht modifiziert).

Demgegenüber kann eine FlexRay-Nachricht bis zu 254 Byte an Nutzdaten beinhalten. Demnach ist FlexRay das geeignetere Bussystem sofern hohe Mengen an Nutzdaten innerhalb einer Botschaft versendet werden müssen. Dabei sind Übertragungsraten von bis zu 10 Mbit/s möglich. Zum Schutz der Daten wird ein 24 Bit CRC eingesetzt. Die Identifikation der Botschaften erfolgt wie auch bei den CAN-Systemen über sogenannte Message Identifier.

Ein weiterer wichtiger Aspekt des Botschaftformats ist die Protokolleffizienz. Diese hängt einerseits vom Overhead des Protokolls und andererseits von den zu übertragenden Nutzdaten ab. Dementsprechend ergibt sich die Protokolleffizienz indem ein Verhältnis von Nutzdaten zu Gesamtdaten (Nutzdaten + Overhead) gebildet wird. Bei 8 Bit Nutzdaten ergeben sich für CAN 14,5% (47 Bit Overhead), für CAN FD 13,8% (50 Bit Overhead) und für FlexRay 11,1% (64 Bit Overhead) Protokolleffizienz. Bei CAN und CAN FD wurde das Standardformat als Grundlage gewählt (Standard Identifier) und das Bitstuffing nicht berücksichtigt.

Entsprechend dieser Resultate ist erkennbar, dass die Protokolleffizienz der drei Bussysteme CAN, CAN FD und FlexRay bei 8 Bit Nutzdaten minimal variiert. Bei größeren Datenmengen steigt die Effizienz, da der Overhead grundsätzlich konstant bleibt. Eine Ausnahme dabei bilden die CAN-Protokolle aufgrund der beiden unterschiedlichen Identifier. Bei FlexRay steigt die Protokolleffizienz auf 96,9 % bei 254 Byte Nutzdaten. Demgegenüber kann bei CAN FD eine Effizienz von maximal 91,1 % bei der Übertragung von 64 Byte an Nutzdaten erreicht werden, da das Nutzdatenfeld mit einer Größe von 64 Byte begrenzt ist.

Folglich kann anhand dieses Indikators, dass am ehesten geeignete Bussystem für verschiedene Anwendungen bestimmt werden. Dazu sind lediglich die zu erwartenden durchschnittlichen Paketgrößen erforderlich. Bei den beiden CAN-Protokollen muss zusätzlich die Größe der notwendigen Identifier, Standard oder Extended, bedacht werden.

6.1.2 Zugriffsverfahren

Beim Zugriffsverfahren unterscheiden sich CAN und CAN FD nicht. Beide Protokolle nutzen das CSMA/CR Verfahren mit Busarbitrierung. Dieses Verfahren ermöglicht das Vermeiden von möglichen Kollisionen indem die Busteilnehmer je nach Priorität die Erlaubnis zum Starten einer Datenübertragung bekommen. Zeitgleich kann lediglich ein Teilnehmer senden. Dementsprechend erhält lediglich der Teilnehmer mit der aktuell höchsten Priorität eine Sendeerlaubnis. Die übrigen Busteilnehmer wechseln dabei sofort in den reinen Empfangsmodus. Folglich garantieren die Protokolle CAN sowie CAN FD eine verlustfreie Datenübertragung.

Demgegenüber werden bei FlexRay die Verfahren TDMA im statischen bzw. FTDMA im dynamischen Teil des Kommunikationszyklus genutzt. Dabei handelt es sich im Gegensatz zum zufälligen Buszugriff mit CSMA/CR bei den CAN-Protokollen um zeitgesteuerte Verfahren. Demnach kann im statischen Teil eine deterministische Datenübertragung garantiert werden. Der dynamische Teil dient dagegen vorrangig zur ereignisgesteuerten Datenübertragung von Botschaften mit geringer Anforderung an die vorhersagbare Latenzzeit. Bei beiden Teilen kann keine Kollision auftreten, da die Datenübertragung der einzelnen Busteilnehmer über Sendeberechtigungen abgehandelt wird. Dabei wird sichergestellt, dass stets ausschließlich ein aktiver Teilnehmer am Bus die Berechtigung zum Senden einer Nachricht erhält. Demnach kann es bei FlexRay-Systemen niemals zu einem zeitgleichen Sendevorgang mehrerer Busteilnehmer kommen.

Entsprechend der verwendeten Verfahren kann lediglich bei FlexRay eine deterministische Datenübertragung garantiert werden. Kollisionen können jedoch bei keinem der drei Bussysteme auftreten, da verschiedene Verfahren zur Kollisionsvermeidung genutzt werden und damit ein kollisionsfreier Betrieb garantiert wird.

Die folgende Abbildung zeigt eine beispielhafte Abarbeitung des Buszugriffsverfahrens CSMA/CR, welches bei den Protokollen CAN und CAN FD eingesetzt wird. Die ersten beiden Zeilen stellen Steuergeräte, die darunterliegende die resultierenden Signale am Bus. Dabei ist erkennbar, dass das zweite Steuergerät (ECU 2) seinen Sendewunsch aufgrund einer Kollision zurückzieht und in den Empfangsmodus wechselt.

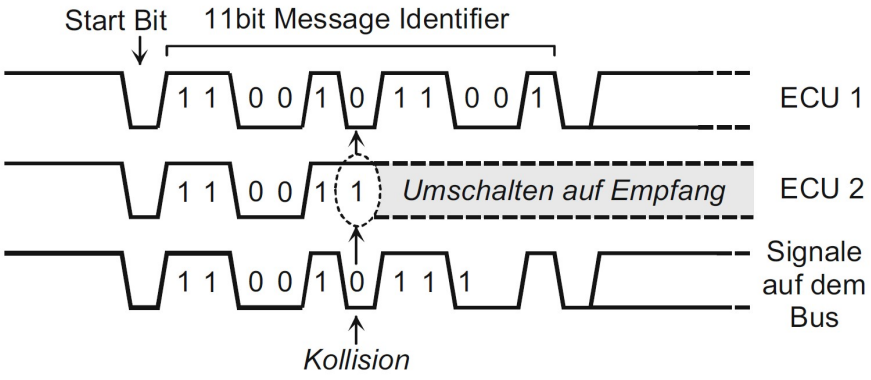


Abb. 54: Kollisionserkennung bei CAN, Quelle: Zimmermann/Schmidgall (2014), S. 62.

6.1.3 Fehlererkennung und Fehlerbehandlung

Das CAN-Protokoll nutzt fünf unterschiedliche Mechanismen zur Fehlererkennung. Dementsprechend wird eine hohe Erkennungsrate erreicht. Bei CAN FD werden ebenfalls diese fünf Mechanismen verwendet. Der CRC-Mechanismus wird jedoch aufgrund des größeren Nutzdatenbereichs angepasst (max. 21 Bit). Darüber hinaus nutzt CAN FD drei zusätzliche Verfahren, um die Fehlerkennungsrate im Vergleich zum CAN-Protokoll noch weiter zu erhöhen. Demnach werden bei CAN FD unter anderem mögliche Stuffbitfehler durch Bit-Flips detektiert, die bei CAN nicht erkannt werden können. Die Fehlerbehandlung wiederum erfolgt bei beiden Protokollen auf die gleiche Weise. Bei Erkennung eines Übertragungsfehlers sendet der Busteilnehmer, der den Fehler erkennt, eine Error-Message aus, um die Botschaft am Bus zu zerstören. Im Anschluss übermittelt der Sender seine Nachricht automatisch erneut. Auch beim Fehlen der Empfangsbestätigung (Acknowledgement) eines Teilnehmers wird die entsprechende Botschaft wiederholt gesendet. Auf diese Weise wird bei den beiden Bussystemen sichergestellt, dass eine Nachricht bei den gesamten Empfängern garantiert ankommt und verarbeitet werden kann.

Die anschließende Darstellung zeigt einen möglichen Fehler in einer CAN-Nachricht durch das Auftreten eines Bit-Flips während einer Datenübertragung. Bei CAN-Systemen würde dieser Fehler nicht erkannt werden. Dagegen wird bei CAN FD aufgrund der zusätzlichen Mechanismen zur Fehlererkennung ein derartiger Fehler detektiert und im Anschluss eine Error-Message ausgelöst sowie eine Sendewiederholung gestartet.

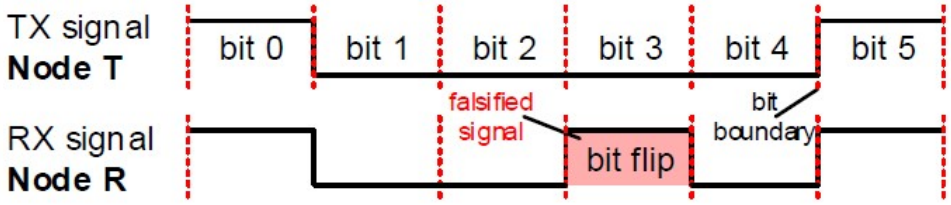


Abb. 55: Bit-Flip, Quelle: Mutter/Hartwich (2015), Online-Quelle [4.August.2016], S. 4.

FlexRay sieht dagegen, aufgrund des deterministischen Systembetriebs keine automatische Sendewiederholung bei Erkennung eines Fehlers vor. Zu einer derartigen Fehlererkennung werden wie bei den CAN-Protokollen fünf unterschiedliche Mechanismen eingesetzt. Ein erkannter Fehler wird jedoch lediglich der darüber liegenden Software-Ebene übergeben. Darüber hinaus wechselt der jeweilige Busteilnehmer im Gegensatz zu CAN bzw. CAN FD in einen passiven Modus oder stoppt seinen Betrieb. Überdies bietet FlexRay die Möglichkeit einen Buswächter einzusetzen, der Datenübertragungsfehler aufgrund eines fehlerhaften Busteilnehmers verhindern kann, indem er diesen abschaltet und damit vom Bussystem trennt.

Folglich bieten die beiden CAN-Protokolle neben einer hohen Fehlererkennungsrate eine automatische Sendewiederholung bei Datenübertragungsfehlern. Dagegen ermöglicht FlexRay ebenfalls einen hohen Fehlerkerkennungsgrad jedoch keine automatische Sendewiederholung, da andernfalls das deterministische Verhalten des Protokolls im statischen Teils des Kommunikationszyklus verletzt werden würde.

6.1.4 Kommunikationstechnik und Realisierung

Die CAN-Protokolle unterscheiden sich in Bezug auf die Kommunikationstechnik in wenigen Punkten. Es handelt sich bei beiden um bitstrom-orientierte Protokolle mit bidirektionaler Zweidrahtleitung. Als Topologie wird zumeist die Linientopologie gewählt. Für den Betrieb werden an den Enden der Busleitung 120 Ohm Widerstände benötigt. Die Unterschiede liegen lediglich in der maximalen Übertragungsrate und der für den Systembetrieb benötigten Hardware. Das gewöhnliche CAN-Protokoll kann bei maximal 1 MBit/s betrieben werden. CAN FD hingegen deckt Datenübertragungsraten von bis zu 4 MBit/s ab und eröffnet damit neue Möglichkeiten in Bezug auf die Vernetzung in der modernen Automobiltechnik.

Bei der Hardware liegt der Unterschied darin, dass CAN-Businterfaces CAN FD-Nachrichten nicht entschlüsseln können. Demgegenüber sind CAN FD-Interfaces grundsätzlich abwärtskompatibel und sind somit in der Lage Nachrichten beider Protokolle empfangen bzw. übertragen zu können. Dementsprechend kann nach einem Umstieg von CAN auf CAN FD weiterhin die Kompatibilität zum Standard CAN-Protokoll gewährleistet werden. Ferner ermöglicht der Umstieg von einem CAN- auf ein CAN FD-Businterface den Einsatz beider Protokolle in einem Bussystem. Dazu muss lediglich zwischen den Protokollen umgeschaltet werden.

Die nachkommende Abbildung zeigt den prinzipiellen Aufbau eines CAN FD-Controllers von Bosch. Derartige Controller werden bereits in der Automobiltechnik als Teil eines CAN-Gateways verbaut und können Botschaften mit bis zu 64 Byte an Nutzdaten bei einer theoretischen Bitrate von maximal 10 MBit/s verarbeiten.

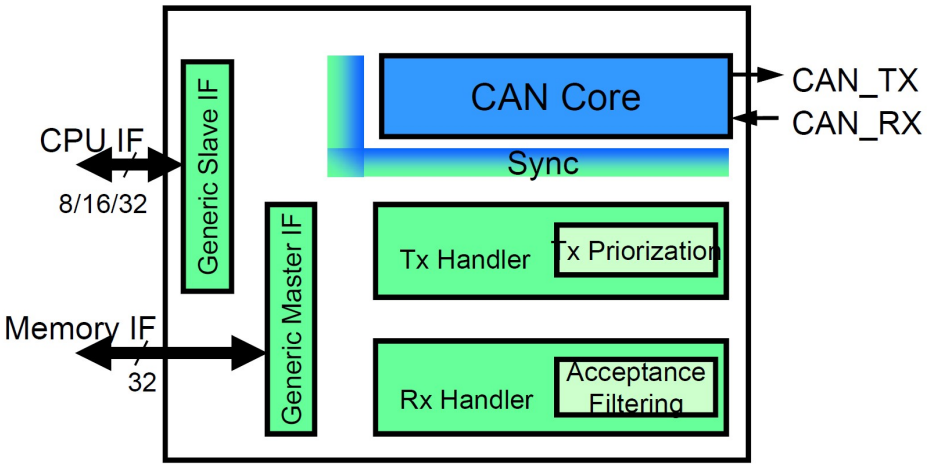


Abb. 56: Bosch CAN FD-Controller, Quelle: Lindenkreuz (2012), Online-Quelle [29.April.2016], S. 13.

Bei FlexRay handelt es sich ebenfalls um ein bitstrom-orientiertes Protokoll mit bidirektionaler Zweidrahtleitung. Als Topologie wird hingegen sowohl die Linien- als auch die Sterntopologie eingesetzt. Für den Betrieb wird ein FlexRay-Controller benötigt. Die maximal erreichbare Datenrate liegt im Gegensatz zu den CAN-Protokollen bei bis zu 10 MBit/s. Darüber hinaus bietet FlexRay die Möglichkeit im zweikanaligen Betrieb eine redundante Verbindung einzurichten oder die Datenrate zu verdoppeln. Der wesentliche Unterschied zu CAN sowie CAN FD liegt neben genannten Eigenschaften im deterministischen bzw. zeitgesteuerten Betrieb.

Dementsprechend wird FlexRay bei Systemen mit Anforderungen nach einer hohen Datenrate von bis zu 10 MBit/s bzw. nach einem deterministischen Verhalten eingesetzt. Sind hingegen Datenraten von bis zu 4 MBit/s ausreichend und ein deterministischer Betrieb nicht erforderlich, ist das CAN FD-Protokoll eine geeignete Alternative. CAN wiederum bietet eine maximale Datenrate von lediglich 1 MBit/s und muss folglich bei einem möglichen Einsatz bedacht werden.

Die folgende Tabelle beinhaltet die wesentlichen Eigenschaften der drei Bussysteme. Anhand dieser übersichtlichen Darstellung sind die Unterschiede als auch die Gemeinsamkeiten der genannten Protokolle ersichtlich.

Protokoll	CAN	CAN FD	FlexRay
Max. Nutzdatengröße pro Botschaft	8 Byte	64 Byte	254 Byte
Message Identifier	Ja	Ja	Ja
Overhead	47 Bit	50 Bit	64 Bit
CRC	Ja	Ja	Ja
Bitratenumschaltung	Nein	Ja	Nein
Buszugriffsverfahren	CSMA/CR	CSMA/CR	TDMA bzw. FTDMA
Deterministisch	Nein	Nein	Ja
Zufälliger Buszugriff	Ja	Ja	Nein
Zeitgesteuert	Nein	Nein	Ja
Kollisionsfreie Kommunikation	Ja	Ja	Ja
Fehlererkennung	Ja	Ja	Ja
Automatische Sendewiederholung	Ja	Ja	Nein
Fehlerbehandlung	Aktiv	Aktiv	Passiv
Buswächter möglich	Nein	Nein	Ja
Abschlusswiderstände	120 Ohm	120 Ohm	100 Ohm
Übertragungsart	Bitstrom-basiert	Bitstrom-basiert	Bitstrom-basiert
Medium	Zweidrahtleitung	Zweidrahtleitung	Zweidrahtleitung
Bevorzugte Topologie	Linie	Linie	Linie bzw. Stern
Max. Datenrate	1 MBit/s	4 MBit/s	10 MBit/s
Knotenkosten z.B. Vector (relativ)	1	1	~2

Tab. 2: Gegenüberstellung CAN, CAN FD und FlexRay, Quelle: Eigene Darstellung.

6.2 Vor- bzw. Nachteile

Nachdem die drei Bussysteme, CAN, CAN FD sowie FlexRay untereinander verglichen wurden, folgen nun kurz zusammengefasst die grundsätzlichen Vor- bzw. Nachteile des neu entwickelten Protokolls gegenüber CAN und FlexRay. Dazu werden wie bei der Gegenüberstellung die vier Hauptbereiche Botschaftsaufbau, Zugriffsverfahren, Fehlererkennung und Fehlerbehandlung sowie Kommunikationstechnik und Realisierung des Bussystems betrachtet. Anhand dieser erläuterten Informationen werden die wesentlichen Eigenschaften des Protokolls und ein etwaiger Einsatz im nachkommenden Abschnitt beurteilt.

Die Vorteile des CAN FD-Protokolls liegen unter anderem im vergrößerten Nutzdatenfeld, welches bis zu 64 Byte umfassen kann. Die dabei resultierende Protokolleffizienz ist beispielsweise bei 8 Bit Nutzdaten vergleichbar mit CAN und FlexRay. Daneben eröffnet die maximale Datenrate von 4 MBit/s neue Möglichkeiten in der Fahrzeugelektronik, da sich durch die maximale Bitrate von 1 MBit/s bei CAN Einschränkungen ergaben. Das eingesetzte Zugriffsverfahren (CSMA/CR mit Busarbitrierung) kann zu den Vor- als auch zu den Nachteilen gezählt werden, da es sich dabei um ein kollisionsfreies jedoch nicht-deterministisches Verfahren handelt. Als weitere Vorteile können somit die Kollisionserkennung sowie die verlustfreie Datenübertragung angeführt werden. Ein weiterer Vorteil des CAN FD-Protokolls ist die aufwendige und komplexe Fehlererkennung bzw. Fehlerbehandlung, die gegenüber CAN noch weiter verbessert wurde. Fehler, die beim CAN-Protokoll nicht erkannt wurden, werden nun durch die neu eingeführten Fehlererkennungsmechanismen erkannt. Auch die Abwärtskompatibilität der CAN FD-Controller zählt zu den Vorteilen dieses Bussystems, da dadurch der parallele Einsatz der beiden CAN-Protokolle in einem Bussystem ermöglicht wird.

Demgegenüber muss jedoch festgehalten werden, dass bei größeren Datenpaketen der Einsatz von CAN FD eher nachteilig ist. Darüber hinaus ist bei gewünschten Datenraten von über 4 MBit/s ebenfalls FlexRay die geeignetere Alternative. Zusätzlich kann je nach Anforderung das eingesetzte Zugriffsverfahren ebenfalls zu den Nachteilen des Protokolls gezählt werden, da bei CAN FD aufgrund der automatischen Sendewiederholung keine deterministische Datenübertragung garantiert werden kann.

Die genannten Vor- und Nachteile sind zur besseren Übersicht in der anschließenden Tabelle dargestellt.

Vorteile	Nachteile
Vergrößertes Nutzdatenfeld	Nutzdatenfeld kleiner als bei FlexRay
Erhöhte Datenrate	Niedrigere Datenrate als FlexRay
Verlustfreie Datenübertragung	Nicht deterministisch
Aktive Fehlerbehandlung	
Abwärtskompatibilität	

Tab. 3: Vor- und Nachteile des CAN FD-Protokolls im Vergleich zu CAN und FlexRay, Quelle: Eigene Darstellung.

6.3 Beurteilung

Anhand der erarbeiteten Informationen in Bezug auf CAN FD werden im Anschluss die Eigenschaften der wesentlichen Bereiche des Protokolls beurteilt und ein etwaiger Einsatz des neu entwickelten CAN-Bussystems in der allgemeinen Prüfstandstechnik bzw. in der vorhandenen Prüfstandsanlage betrachtet sowie erläutert. Dazu werden unter anderem jene Bereiche in Betracht gezogen, in denen das CAN-Protokoll an seine Grenzen gestoßen ist und sich durch den Umstieg auf CAN FD neue Möglichkeiten ergeben. Schlussendlich wird aufgrund der gesammelten Informationen ein Fazit zum möglichen Einsatz von CAN FD gebildet.

Nachdem das Nutzdatenfeld vergrößert (von 8 auf 64 Byte) wurde, können nun beispielsweise mehr Signale in einer Botschaft als bei CAN übertragen werden. Dementsprechend müssen bei CAN FD im Vergleich zu CAN-Systemen wesentlich weniger Botschaften bei gleicher Anzahl an Signalen übertragen werden. Dies würde die Parametrierung des Vorgabesystems erheblich vereinfachen, da jede Botschaft einzeln eingetragen werden muss. Darüber hinaus gestaltet sich eine Beschreibungsdatei bei weniger Nachrichten übersichtlicher. Des Weiteren werden aufgrund des größeren Payloadbereichs weniger Message Identifier benötigt, da ein Identifier eine größere Anzahl an Signalen repräsentiert als bei CAN-Bussystemen. Dies würde die Konfiguration am Prüfstands-Automatisierungssystem ebenfalls erleichtern. Hinsichtlich des vergrößerten Nutzdatenbereichs, wurde der Cyclic Redundancy Check entsprechend angepasst. Demzufolge kann auch weiterhin eine hohe Fehlererkennungsrate und somit ein sicherer Prüfbetrieb garantiert werden. Ohne diese Anpassung würde die Hamming-Distanz sinken und müsste bei einem Einsatz am Prüfstand bedacht werden. Die wenigen zusätzlichen Bits (3 Bits) bei CAN FD haben keinen wesentlichen Einfluss auf den Betrieb verglichen mit dem CAN-Protokoll. Folglich ist bei diesem neuentwickelten Botschaftsformat die Einsatztauglichkeit für die Prüfstandsanlage ohne Einschränkungen gegeben.

Das Zugriffsverfahren unterscheidet sich bei CAN FD und CAN nicht. Bei beiden wird das Verfahren CSMA/CR mit Busarbitrierung eingesetzt. Dementsprechend ist bei einem Umstieg auf das weiterentwickelte FD-Protokoll in Bezug auf den Buszugriff keine Anpassung der Prüfstandssysteme erforderlich. Nachdem dieses Verfahren einen kollisionsfreien und ausfallsicheren Betrieb garantiert, ist auch dieser Bereich für den Einsatz am Prüfstand geeignet. Einzig das nicht-deterministische Verhalten dieses Prinzips ist wie auch bei der Verwendung des CAN-Protokolls bei der Parametrierung des Vorgabesystems zu bedenken.

Da die Fehlererkennung im Zuge der Entwicklung von CAN FD noch weiter verbessert wurde, können nun Fehler erkannt werden, die beim CAN-Protokoll nicht detektiert wurden. Demzufolge wird die Ausfallsicherheit am Prüfstand bei Verwendung des CAN FD-Protokolls wesentlich erhöht. Daneben wurde die Fehlerbehandlung nicht verändert. Demnach wird nach der Erkennung eines Fehlers bei der Übertragung einer Botschaft weiterhin automatisch eine Sendewiederholung gestartet. Dies ist wesentlich für die verlässliche Funktion der Prüfstandsanlage. Darüber hinaus kann dadurch eine verlustfreie Datenübertragung garantiert werden. Folglich ergibt sich in diesem Bereich eine klare Verbesserung beim Umstieg von CAN auf CAN FD. Die Einsatztauglichkeit für den Prüfstandsbetrieb ist damit ebenfalls ohne Einschränkungen gegeben.

Im Bereich der Kommunikationstechnik unterscheiden sich die beiden CAN-Protokolle nur minimal. Die Busleitungen, die Abschlusswiderstände sowie die bestehende Topologie können bei einem Umstieg von CAN zu CAN FD beibehalten werden. Lediglich das Businterface bzw. das I/O-System müssen umgerüstet und das Prüfstands-Automatisierungssystem angepasst werden, um eine Kompatibilität zum CAN FD-Protokoll garantieren zu können. Diese CAN FD-Controller können grundsätzlich Botschaften beider Protokolle verarbeiten. Dementsprechend ist es nach der Umrüstung bzw. Anpassung der genannten Systeme möglich beide Protokolle einzusetzen. Dies würde die Flexibilität der Prüfstandsanlage erheblich steigern. Daneben wird die Performance des Prüfstandssystems durch die erhöhte Datenrate des CAN FD-Protokolls von maximal 4 MBit/s und der dadurch merklich gesteigerten Nutzdatenrate ebenfalls wesentlich erhöht. Aufgrund dessen ist bei diesem Teil des Protokolls die Einsatztauglichkeit für den dauerhaften Prüfbetrieb ebenfalls vollends und ohne Einschränkungen gegeben.

Anhand der nachfolgenden Darstellung wird die massive Steigerung der Nutzdatenrate bei einem Umstieg vom klassischen CAN- zum neuentwickelten CAN FD-Protokoll deutlich. Demnach wird die Nutzdatenrate bei einer Übertragung von 8 Byte und im Falle der Bitratenumschaltung auf 4 MBit/s mehr als verdoppelt (von 29 auf 79 KB/s). Berechnet wurden die angegebenen Übertragungsraten mit einem 11 Bit Message Identifier. Das Bit Stuffing wurde bei den Berechnungen berücksichtigt. Dabei wurde der ungünstigste Fall angenommen.

n_{Data}	Klassischer CAN 2.0 $f_{\text{bit}} = 500$ kbit/s	CAN FD ohne Bitratenumschaltung $f_{\text{bit,N}} = f_{\text{bit,H}} = 500$ kbit/s	CAN FD mit Bitratenumschaltung $f_{\text{bit,N}} = 500$ kbit/s, $f_{\text{bit,H}} = 4$ Mbit/s
8 Byte	Nicht möglich	29 KB/s	79 KB/s
16 Byte		35 KB/s	131 KB/s
32 Byte		40 KB/s	195 KB/s
64 Byte		44 KB/s	260 KB/s

Abb. 57: CAN/CAN FD Nutzdatenraten, Quelle: Zimmermann/Schmidgall (2014), S. 78.

Hinsichtlich der erläuterten Teile ist erkennbar, dass sich der Umstieg von CAN auf das neuentwickelte CAN FD-Protokoll zweifellos lohnt, da sich dadurch wesentliche Verbesserungen (z.B. höhere Datenrate, gesteigerte Fehlererkennungsrate, vergrößertes Nutzdatenfeld etc.) ergeben. Der dabei resultierende Aufwand fällt verhältnismäßig gering aus, da lediglich auf die erhöhte Datenrate sowie das vergrößerte Nutzdatenfeld softwareseitig geachtet und die Hardware (Businterface) entsprechend umgerüstet werden muss. Wesentliche Bereiche wie das Buszugriffsverfahren oder die Fehlerbehandlung wurden nicht verändert. Busleitungen sowie Abschlusswiderstände können beibehalten werden. Folglich ist es möglich, einen Wechsel vom gewöhnlichen CAN-Protokoll auf das neue CAN FD-Bussystem inklusive der dazu erforderlichen Tätigkeiten bzw. Anpassungen rasch und problemlos durchzuführen und damit die Performance sowie Flexibilität des Prüfstandssystems wesentlich zu erhöhen. Demnach würde ein derartiger Systemumstieg während der Umrüstung bzw. Anpassung keine langen Stillstandzeiten der Prüfstandsanlage verursachen.

Darüber hinaus stellt sich anhand der Eigenschaften des CAN FD-Protokolls heraus, dass es sich dabei um eine brauchbare Alternative zu FlexRay handelt. Der direkte Umstieg von CAN auf FlexRay aufgrund der höheren Datenrate ist somit nicht mehr zwingend, da das neuentwickelte Bussystem Datenraten von bis zu 4 MBit/s unterstützt. Dementsprechend wird in Zukunft, sofern die Übertragungsrate ausreichend ist und im Falle, dass das CAN-Protokoll den Anforderungen nicht entspricht, vermehrt auf das CAN FD-Protokoll zurückgegriffen werden, da der Wechsel von CAN auf FlexRay mit einem weitaus höheren Aufwand verbunden ist. Bei einem derartigen Umstieg müssten sowohl softwareseitig als auch in Bezug auf die Hardware (z.B. Businterface, Abschlusswiderstände etc.) aufwendige Anpassungen bzw. Änderungen vorgenommen werden. Folglich ergibt sich durch den Einsatz des CAN FD-Protokolls eine Möglichkeit dem komplexen Wechsel von CAN auf FlexRay und dem damit resultierenden hohen Aufwand zu entgehen, da es beispielsweise nicht erforderlich ist Anwendungen vollständig neu zu programmieren oder Teile der Übertragungsleitungen anzupassen. Sind jedoch Übertragungsraten von über 4 MBit/s sowie ein deterministisches Verhalten des Bussystems erforderlich, ist weiterhin das FlexRay-System die bestmögliche Wahl.

7 RESÜMEE & AUSSICHT

Das Ziel dieser Arbeit war die Erstellung eines Konzepts zur Echtzeit-Kopplung automobiler Steuergeräte an ein Prüfstands-Automatisierungssystem, das den Einsatz einer Schnittstelle zur automatisierten Bedatung von automobilen Steuergeräten und den möglichen Umstieg von CAN auf CAN FD behandelt, um die Flexibilität sowie Performance der Anlage erhöhen zu können. Dazu wurden zunächst die entsprechenden Grundlagen der erforderlichen Themenbereiche, wie allgemeine Bustechnik, Bussysteme, ASAM Standardisierungen etc., erarbeitet und im anschließenden Praxisteil genutzt, um die zuvor gewählte Automatisierungsschnittstelle zu testen bzw. deren Einsatztauglichkeit zu bewerten sowie den möglichen Umstieg des neuentwickelten CAN FD-Protokolls anstatt des gewöhnlichen CAN-Systems zu erläutern und schlussendlich zu beurteilen.

Demnach wurden in Bezug auf die erarbeiteten Themen des Theorieteils im Umfang des Praxisteils zahlreiche Funktionstests durchgeführt sowie mögliche Einsatzszenarien betrachtet bzw. bewertet. Entsprechend der daraus resultierenden Ergebnisse wurde die Einsatztauglichkeit der einzelnen Bereiche abgeleitet. Im Zuge dessen konnte festgestellt werden, dass die Implementierung der COM-Schnittstelle von Vector in Kombination mit der Systemdesignsoftware LabVIEW die automatisierte Bedatung automobiler Steuergeräte gemäß den gesetzten Anforderungen ermöglicht. Einzig eine Teilfunktion der Schnittstelle, der Wechsel einer CANape-Konfigurationsdatei, erfordert weiterhin einen manuellen Eingriff und erfüllt somit nicht die Ansprüche nach einer vollautomatisierten Abwicklung des Vorgangs. Dabei handelt es sich jedoch lediglich um einen nebensächlichen Teil, der keine gravierende Auswirkung auf den gewünschten Prüfbetrieb hat. Darüber hinaus hat sich im Laufe des zweiten Abschnitts des Praxisteils herausgestellt, dass das moderne CAN FD-Protokoll eine Vielzahl an Verbesserungen im Vergleich zum herkömmlichen CAN-System bietet, ohne Einschränkungen für den dauerhaften Einsatz im bestehenden Prüfstandssystem geeignet ist und aufgrund der erhöhten Datenrate eine brauchbare Alternative zu FlexRay darstellt.

Somit entspricht sowohl die gewählte Automatisierungsschnittstelle als auch das CAN FD-Bussystem den Anforderungen für einen verbesserten und weiterhin ausfallsicheren Prüfbetrieb ohne bedeutende Einschränkungen. Demzufolge bildet dieses Konzept die Grundlage für die Implementierung der COM-Schnittstelle einschließlich des CAN FD-Bussystems, um einen automatisierten sowie optimierten Prüfbetrieb sicherstellen zu können. Dazu sind die Anpassung des Prüfstand-Automatisierungssystems, die hardwareseitige Umrüstung des I/O-Systems bzw. des Businterfaces und eine Adaptierung am Applikations-System-PC erforderlich. Infolge dieser Tätigkeiten ergibt sich eine wesentliche Steigerung der Performance bzw. Flexibilität des Prüfstandsystems, da unter anderem die Auslastung (24-Stundenbetrieb möglich) der Anlage aufgrund der automatisierten Bedatung der Steuergeräte massiv steigt sowie der Umstieg auf das moderne CAN FD-Protokoll einen zukunftsorientierten sowie flexiblen Prüfbetrieb erlaubt. In Zukunft wird in der Automobilbranche vermehrt auf das CAN-Protokoll mit flexibler Datenrate zurückgegriffen werden. Folglich ist es von Vorteil, das für diese Arbeit betrachtete Prüfstandsystem sowie die übrigen Prüfstandsanlagen im Unternehmen frühzeitig anzupassen, um für zukünftige Projekte, in denen CAN FD-Kompatibilität gefordert wird, gerüstet zu sein.

Gemäß dem Vorrangegangen wurde das Ziel dieser Arbeit, ein Konzept zu entwerfen, welches die Optimierung der bestehenden Prüfstandsanlage ermöglicht, erreicht. Zur besseren Veranschaulichung folgt im Anschluss eine vereinfachte Darstellung des genannten Konzepts, dass den Einsatz der COM-Schnittstelle und des CAN FD-Protokolls am bestehenden Prüfstandsystem (beide Systeme sind in Rot gehalten) vorsieht. Folglich sind nun verglichen zur Ausgangslage die entsprechende Automatisierungsschnittstelle zwischen Prüfstands-Automatisierungs-PC und Applikations-System-PC sowie das CAN FD-Bussystem zur Kommunikation mit dem Steuergerät der Prüfkomponente angeführt. Aufgrund der Abwärtskompatibilität der CAN FD-Controller sind beide CAN-Protokolle angegeben, da das klassische CAN-System neben dem modernen FD-Protokoll angesichts der sukzessiven Umstellung im automobilen Sektor zunächst weiterhin verwendet wird.

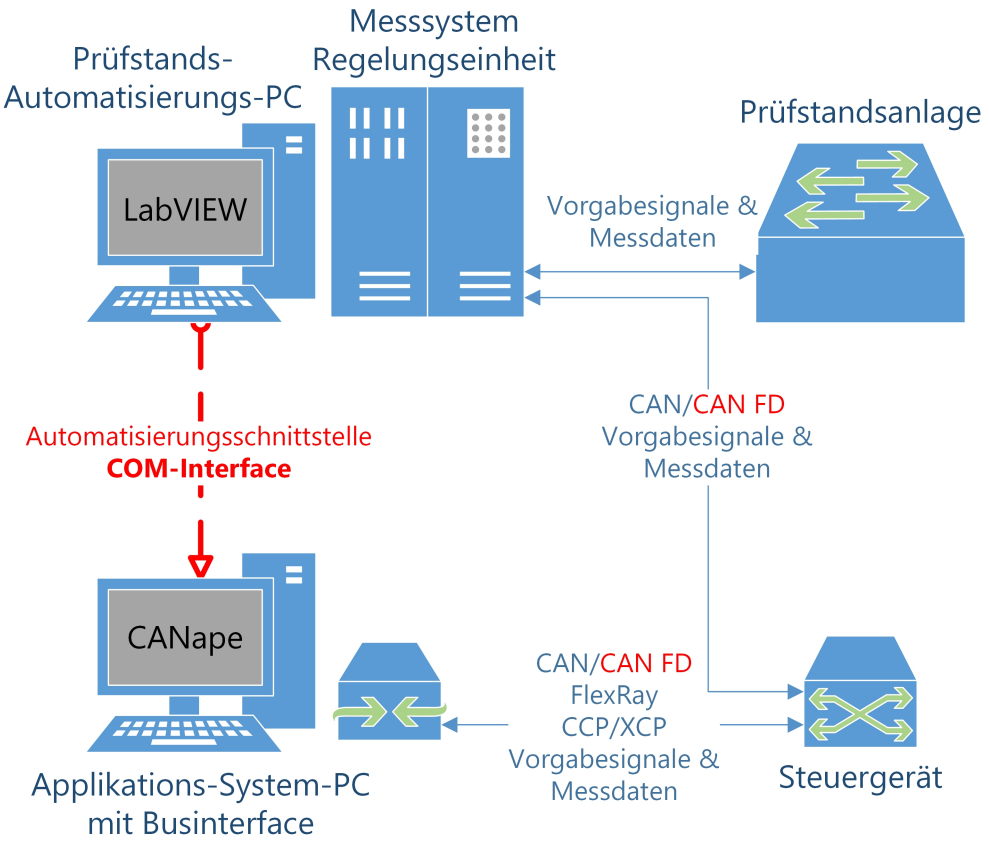


Abb. 58: Konzeptschema vereinfacht, Quelle: Eigene Darstellung.

Schlussendlich ergeben sich anhand der erarbeiteten Bereiche folgende Empfehlungen an das Unternehmen. Einerseits ist es ratsam die COM-Schnittstelle von Vector in das bestehende Prüfstands-Automatisierungssystem zu implementieren und die dazu notwendigen Anpassungen am Applikations-System-PC durchzuführen. Dies würde den Auslastungsgrad aufgrund der vollautomatisierten Bedienung erheblich steigern. Darüber hinaus ist es empfehlenswert das I/O-System sowie das Businterface auf CAN FD-Kompatibilität umzurüsten, um für zukünftige Projekte frühzeitig vorbereitet zu sein und damit die Flexibilität der Anlage weiter zu erhöhen. Diese Tätigkeiten auf Basis der vorliegenden Arbeit sind somit notwendig, um den geforderten Prüfbetrieb zu erreichen und die Anzahl der manuellen Eingriffe während eines Prüflaufs auf ein Minimum zu reduzieren. Folglich bildet die vorliegende Arbeit die Grundlage bzw. den Ausgangspunkt für die zukünftige Optimierung der bestehenden Prüfstandsanlage.

LITERATURVERZEICHNIS

Gedruckte Werke (9)

Reif, Konrad (Hrsg.) (2011): *Bosch Autoelektrik und Autoelektronik: Bordnetze, Sensoren und elektronische Systeme*, 6. Auflage, Vieweg+Teubner Verlag, Wiesbaden

Borgeest, Kai (2014): *Elektronik in der Fahrzeugtechnik: Hardware, Software, Systeme und Projektmanagement*, 3. Auflage, Springer Vieweg, Wiesbaden

Knoll, Steffen; Hendratno, Gerry; Herzog, Stephan (2015): *How to efficiently extend test benches and HIL systems: Integrating CANape via an automation interface*, 1. Auflage, Vector Informatik GmbH, Stuttgart

Patzer, Andreas; Herzog, Stephan (2016): *CANape 14.0: New Features Overview*, 1.03. Auflage, Vector Informatik GmbH, Stuttgart

Patzer, Andreas; Zaiser, Rainer (2014): *XCP – Das Standardprotokoll für die Steuergeräte-Entwicklung: Grundlagen und Einsatzgebiete*, 1. Auflage, Vector Informatik GmbH, Stuttgart

Paulweber, Michael; Lebert, Klaus (2014): *Mess- und Prüfstandstechnik: Antriebsstrangentwicklung, Hybridisierung, Elektrifizierung*, 1. Auflage, Springer Vieweg, Wiesbaden

Reif, Konrad (2014): *Automobilelektronik: Eine Einführung für Ingenieure*, 5. Auflage, Springer Vieweg, Wiesbaden

Streichert, Thilo; Traub, Matthias (2012): *Elektrik/Elektronik-Architekturen im Kraftfahrzeug: Modellierung und Bewertung von Echtzeitsystemen*, 1. Auflage, Springer Vieweg, Berlin Heidelberg

Zimmermann, Werner; Schmidgall, Ralf (2014): *Bussysteme in der Fahrzeugtechnik: Protokolle, Standards und Softwarearchitektur*, 5. Auflage, Springer Vieweg, Wiesbaden

Online-Quellen (10)

Magna Steyr Engineering (2016): *corporate.intranet.magnasteyr.com*
http://corporate.intranet.magnasteyr.com/de/web/ms-intranet/intranet-graz#http://cms.intranet.magnasteyr.com/ms/EU/AT/graz/5825_DEU_PIN.php?ts=1462784312044 [Stand: 9.Mai.2016]

Vector Informatik GmbH (2014): *Kompendium ausgewählter Fachartikel zur Elektronik-Entwicklung in verteilten Systemen*
http://vector.com/portal/medien/cmc/marketing_items/web/91110.pdf [Stand: 29.April.2016]

CAN in Automation (2016): *CAN FD - The basic idea*
http://www.can-cia.org/can-knowledge/can/can-fd/ [Stand: 30.April.2016]

Magna International Inc. (2016): *Magna: Home*
http://www.magna.com/ [Stand: 9.Mai.2016]

ASAM e.V. (2015): *About ASAM*
http://www.asam.net/home/about-asam.html [Stand: 20.August.2016]

Hartwich, Florian (2012): *CAN with Flexible Data-Rate*
http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/ipmodules_1/can_fd/icc13_2012_paper_Hartwich.pdf [Stand: 29.April.2016]

Kless, Alfred (2013): *Use of ASAM MC-Standards in Vector products: History and success stories*
https://www.asam.net/fileadmin/documents/News/10_MC-Systems_-_Vector_01.pdf [Stand: 29.April.2016]

Lindenkreuz, Thomas (2012): *CAN FD – CAN with Flexible Data Rate*
https://vector.com/portal/medien/cmc/events/commercial_events/VectorCongress_2012/VeCo12_8_NewBusSystems_3_Lindenkreuz_Lecture.pdf [Stand: 29.April.2016]

Mutter, Arthur; Hartwich, Florian (2015): *Advantages of CAN FD Error detection mechanisms compared to Classical CAN*
http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/icc_2015_mutter.pdf [Stand: 4.August.2016]

Zeltwanger, Holger (2014): *CAN FD - schnell und zuverlässig*
http://www.elektroniknet.de/automotive/bussysteme/artikel/105590/ [Stand: 29.April.2016]

Elektronische Quellen (1)

Vector Informatik GmbH (2016): *Interface Description: CANape COM Interface 14.0.30: Interface Description*, 14.0.30. Auflage, Vector Informatik GmbH, Stuttgart

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

ACI	Automatic Calibration Interface
ACK	Acknowledge
AE	Automotive Electronics
AML	ASAM2 Meta Language
API	Application Programming Interface
ASAM	Association for Standardisation of Automation and Measuring Systems
ASAP	Arbeitskreis zur Standardisierung von Applikationssystemen
BRS	Bit Rate Switch
CA	Collision Avoidance
CAN	Controller Area Network
CAS	Collision Avoidance System
CAT	Computer Aided Testing
CCP	CAN Calibration Protocol
CD	Collision Detection
CDF	Calibration Data Format
CEA	Components for Evaluation and Analysis
CR	Collision Resolution
CRC	Cyclic Redundancy Check
CRO	Command Receive Objects
CSMA	Carrier Sense Multiple Access
CTO	Command Transfer Objects
DAQ	Data Acquisition
DBC	Data Base CAN
DEL	Delimiter
DLC	Data Length Code
DLL	Dynamic Link Library
DTO	Data Transmission Objects
DVP	Design Validation Plan
ECU	Electronic Control Unit

EDL	Extended Data Length
EEPROM	Electrical Erasable Programmable Read Only Memory
EOF	End of File/End of Frame
EPROM	Erasable Programmable Read Only Memory
ESI	Error State Indicator
ETK	Emulationstastkopf
FD	Flexible Data Rate
FIBEX	Field Bus Exchange Format
FIFO	First In – First Out
FTDMA	Flexible Time Division Multiple Access
F&E	Forschung und Entwicklung
GDI	Generic Device Interface
ICB	Inter Character Break
IDE	Identifier Extension
IFB	Inter Frame Break
IFS	Interframe Space
ISO	International Organization for Standardization
LDF	LIN Description File
LIN	Local Interconnect Network
LLC	Logic Link Control
LSB	Least Significant Bit
LXF	Layout Exchange Format
MAC	Medium Access Control
MAC	Media Access Controller
MC	Measurement, Calibration
MCD	Measurement, Calibration, Diagnosis bzw. Measure, Calibrate, Diagnose
MDF	Measurement Data Format
MDI	Medium Dependant Interface
MDX	Model Data Exchange Format
MOST	Media Oriented System Transport
MSE	Magna Steyr Engineering

MTS	Media Access Test
NCF	Node Capability File
ND	Nutzdaten
NIT	Network-Idle-Time
NRZ	Non-Return-to-Zero
OBD	On-Board Diagnostic
ODS	Open Data Service
ODX	Open Data Exchange
OSI	Open Systems Interconnection
OTX	Open Test Sequence Exchange
PCI	Protocol Control Information
PDU	Protocol Data Unit
PID	Packet Identifier
PLS	Physical Signalling
PMA	Physical Medium Attachment
RAM	Random Access Memory
RTR	Remote Transmission Request
SAE	Society of Automotive Engineers
SDU	Service Data Unit
SOF	Start of Frame
SRR	Substitute Remote Request
TDMA	Time Division Multiple Access
UART	Universal Asynchronous Receiver and Transmitter
UDS	Unified Diagnostic Services
UDP	User Datagram Protocol
USB	Universal Serial Bus
XCP	Universal Measurement and Calibration Protocol
XML	Extensible Markup Language

ANHANG: CANAPE COM INTERFACE DESCRIPTION

Im Anschluss folgen die wesentlichen Abschnitte der Schnittstellenbeschreibung zum COM-Interface von Vector. Diese Beschreibungsdatei mit dem Namen CANapeCOM.chm ist in der Installation von CANape sowie der COM-Client Anwendung inkludiert und nicht online verfügbar. Das anschließende Zitat bezieht sich auf alle nachkommenden Abschnitte der Beschreibung, nicht jedoch auf diesen Text.[158]

CANape COM Interface 14.0.30

Interface Description

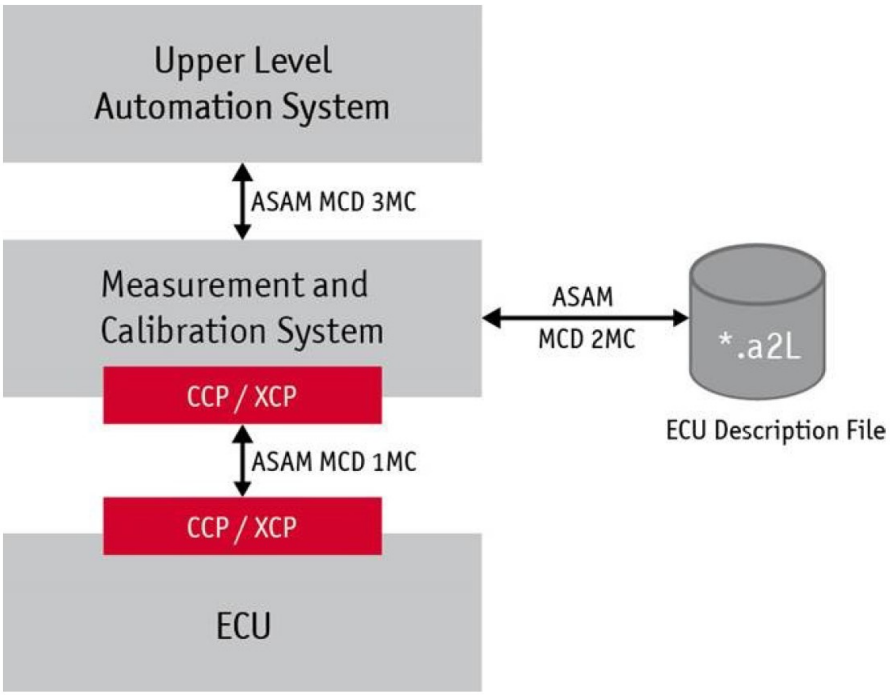
Main Page | Modules | Classes

Start Page

Application Development with CANape COM interface

In test benches, HIL systems and other test fixtures, it is important to access all necessary input and output variables simply and quickly
The CANape COM interface provides a client connection to CANape.

It is easy to use in VB, VBA/VBS, C#, Python, LabView and many further script-based languages



It's possible to use the CANape COM interface in two different ways.
The first way is to use it on your local machine. CANape and your client appliction runs on the same machine. The second way is to use it on a network over TCP/IP. CANape runs on another machine than your client application do. See **tcpdevenv TCP/IP Development Environment** .

vector

[158] Vector Informatik GmbH (2016), Elektronische Quelle.

CANape COM Interface 14.0.30

Interface Description

Main Page | Modules | Classes

Available driver types

General

driver types

types

The CANape COM interface is able most in CANape available drivers
Here is a list of all supported drivertypes of the COM interface.

Driver typ	Description
CCP	CCP Driver
XCP	XCP Driver
CAN	CAN monitoring Driver
KWPONCAN	KWP on CAN Driver
CANOPEN	CANOpen protocol Driver
ANALOG	Anaglog Driver
HEXEDIT	Virtueller Driver
CANDELA	Diagnose Driver
ENVIRONMENT	Internal Driver to access CANape internal objects
LIN	LIN monitoring Driver
FLEXRAY	FlexRay monitoring Driver
NIDAQMX	National Instruments DAQMX Driver
ETHERNET	ETHERNET monitoring Driver

See Also:
IDevices::Add, **IDevices2::Add2** **IDevices2::Add3**

CANape COM Interface 14.0.30

Interface Description

Main Page | Modules | Classes

Available hardware channels

General

hardware channels

Channels

The CANape Com interface to connect to most of CANape sported drivers
Here is a list of supported Hardware channels

Bustyp	Harware Channel	Value
CAN	CAN 1-CAN20	1-20
FelxRay	FLX1-FLX8	31-38
Lin	LIN1-LIN8	61-68
VX Device CAN	VX_CAN1 -VX_CAN4	81-84
VX Device TCP	VX_TCP	85
SXI	SXI1-SXI8	91-98
USB		110
TCP		255
UDP		256
Userdfined		261

See Also:
IDevices::Add, **IDevices2::Add2** **IDevices2::Add3**

CANape COM Interface 14.0.30

Interface Description

Main Page | Modules | Classes

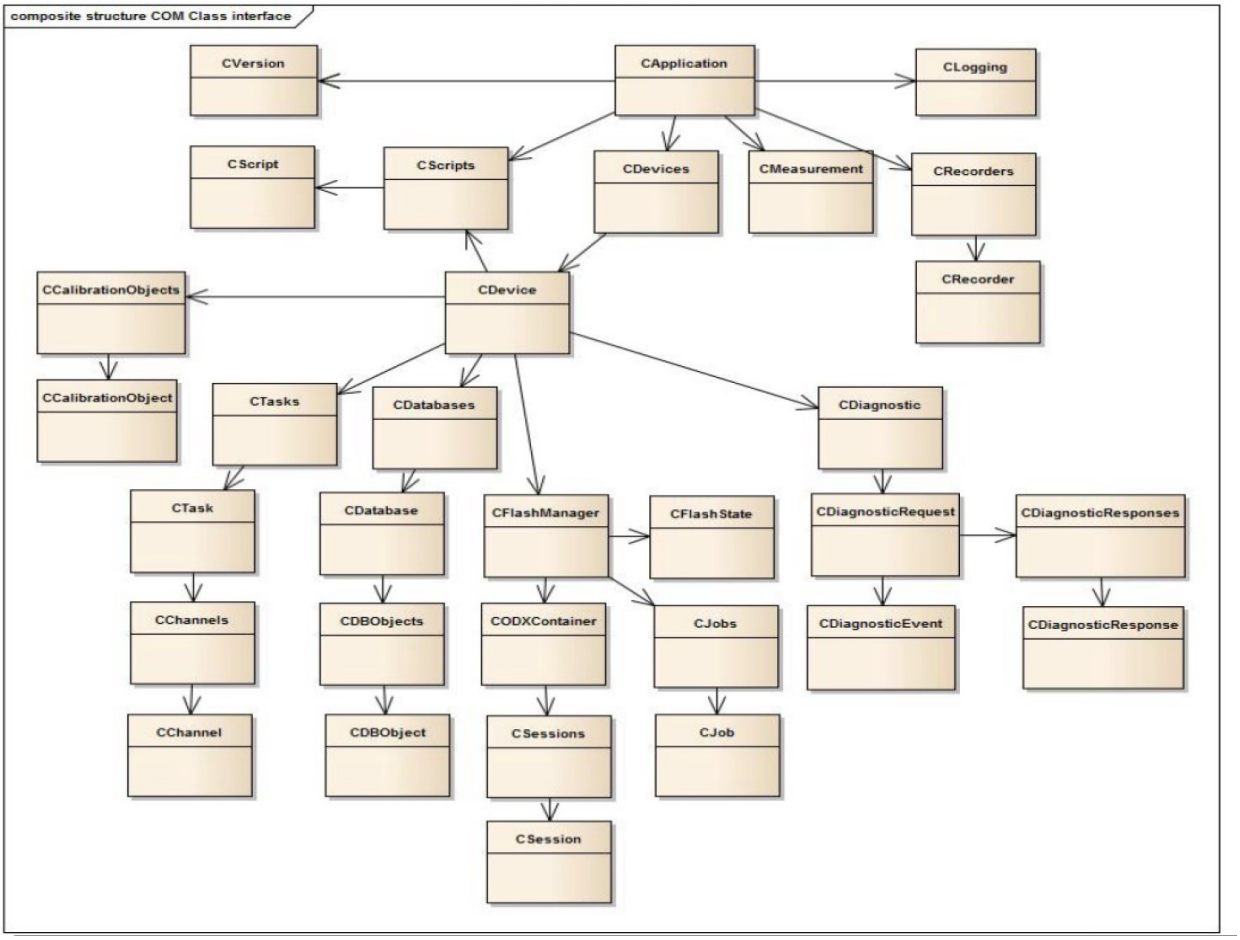
Inheritance Diagram

General

Object hierarchy diagramm.

hierarchy

Object Diagram



CANape COM Interface 14.0.30

Interface Description

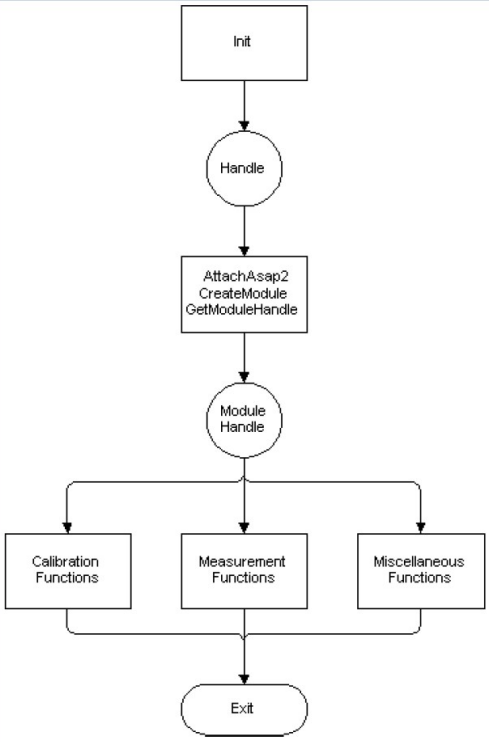
Main Page | Modules | Classes

Init & Exit Sequence

General

Establish connection to CANape and create/get module handle for further access.

Init & Exit Sequence



To establish a connection between your application and CANape over the ASAP 3 Interface, you have to call one of the **init function** . The init function starts CANape and configures some settings(e.g responseTimeout, fifoSize). The return value of the Init function indicates, if the connection with CANape has been established or has been failed (after timeout).

Attention

The CANapeApi send a request to CANape. Normally CANape send a response. The CANapeAPI waits for a duration of the given parameter "responseTimeout". Pay attention to set them not too high or low.

If the connection is established, a module has to be selected. A module corresponds to a device in CANape(see device list). There are 3 different ways to select a module:

CreateModule
Create a new module, select a driver and attach a description file (A2L,DB,DBC).

Attention

The functions **IDevices::Add()** and **IDevices2::Add2()** expect existing modules. Be aware, that some existing modules are not deleted in the **IDevices::Add()** function. If the parameter *clearDeviceList* is true the list will be deleted.

To close the connection call the **exit function** or **exit function**.
If the ASAP3 Interface is called by the COM interface, CANape will terminate.

See Also:

IApplication::Open()
IApplication2::Open2()
IDevices::Add()
IDevices2::Add2()
IApplication::Quit()
IApplication::QuitNonModal() .

CANape COM Interface 14.0.30

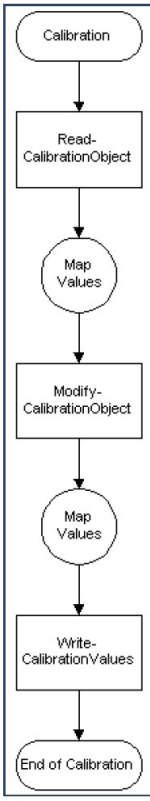
Interface Description

Main Page | Modules | Classes

Calibration Sequence

General

Calibration Sequence



To calibrate an Object you have to get some object information, first. The ReadCalibrationObject function **ICalibrationObject::Read** returns specific type data from a given CalibrationObjectName. It will return a value, type specific information and data by calling **ICalibrationObject::Value**.

To modify some values change them in the union TCalibrationObjectValue and call the **ICalibrationObject::Write()**

VBA Calibration Example

```
Dim CANapeApplication
Dim CANapeDevice
Dim calValue As Double

Call CANapeApplication.Open1(WorkingDir.Text, 0, 30000, 1)

Set CANapeDevice1 = CANapeApplication.Devices.Add(Devicename.Text, ASAP2.Text, Driver.Text, Val(Channel.Text))


CANapeDevice.CalibrationObjects.Clear
CANapeDevice.CalibrationObjects.Add (CalibrationObject.Text)
Set o = CANapeDevice.CalibrationObjects(1)
o.RepresentationType = RepresentationTypeFlag
o.Read
calValue=o.Value


Call CANapeApplication.Quit
Set CANapeDevice = Nothing
Set CANapeApplication = Nothing
```

See Also:
ICalibrationObject::Write(), **ICalibrationObject::Read**

CANape COM Interface 14.0.30

Interface Description

Main Page | Modules | Classes

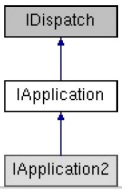
Class List | Class Index | Class Hierarchy | Class Members

Public Member Functions | Properties | List of all members

IApplication Interface Reference

Mesurement and Calibration

Inheritance diagram for IApplication:



Public Member Functions

HRESULT	**Open** ([in] BSTR workDir,[in] long debug)
HRESULT	**Quit** ()
HRESULT	**RunScript** ([in] BSTR script)
HRESULT	**Convert2Matlab** ([in] BSTR MDFFile,[in] BSTR MatlabFile,[in, defaultvalue(FALSE)] BOOL asyncron)
HRESULT	**DebugWindow** ()
HRESULT	**QuitNonModal** ()
HRESULT	**Open1** ([in] BSTR workDir,[in] long debug,[in] long timeout,[in] BOOL modalmode)

Properties

BSTR	**Name** [get]
IDispatch	**Devices** [get]
IDispatch	**Measurement** [get]
IDispatch	**Exception** [get]
IDispatch	**Version** [get]
BSTR	**AppName** [set]
IDispatch	**APPVersion** [get]
IDispatch	**Recorders** [get]

CANape COM Interface 14.0.30

Interface Description

Main Page | Modules | Classes

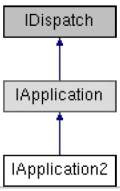
Class List | Class Index | Class Hierarchy | Class Members

Public Member Functions | Properties | List of all members

IApplication2 Interface Reference

Mesurement and Calibration

Inheritance diagram for IApplication2:



Public Member Functions

HRESULT	**Open2** ([in] BSTR workDir,[in] long debug,[in] long timeout,[in] BOOL cleardevicelist,[in] BOOL modalmode,[in] VARIANT apptype)
HRESULT	**SaveDebugWindow** ([in] BSTR FileName)
HRESULT	**LoadCNAFile** ([in] BSTR FileName)
HRESULT	**SelectLabelList** ([in] BSTR FileName,[in, defaultvalue(FALSE)] BOOL includeMeaMode,[in, defaultvalue(1)]int mode)

▸ Public Member Functions inherited from **IApplication**

Properties

BSTR	**AppName** [get]
BOOL	**USVersion** [get]
BSTR	**CNAFilename** [get]
BSTR	**WorkingDirectory** [get]
IDispatch	**Scripts** [get]
BSTR	**IPAddress** [get, set]
short	**CANapePort** [get, set]
short	**CNPPort** [get, set]
IDispatch	**Logging** [get]
IDispatch	**MDFConverters** [get]
IDispatch	**NetWorks** [get]

▸ Properties inherited from **IApplication**

CANape COM Interface 14.0.30

Interface Description

Main Page | Modules | Classes

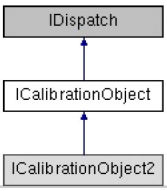
Class List | Class Index | Class Hierarchy | Class Members

Public Member Functions | Properties | List of all members

ICalibrationObject Interface Reference

Calibration Classes

Inheritance diagram for ICalibrationObject:



Public Member Functions

HRESULT **Read** ([out, retval] VARIANT *pVal)

HRESULT **Write** ()

Properties

BSTR **Name** [get]

long **XDim** [get]

long **YDim** [get]

long **Type** [get]

VARIANT **Value** [get, set]

eRepresentationType **RepresentationType** [get, set]

CANape COM Interface 14.0.30

Interface Description

Main Page | Modules | Classes

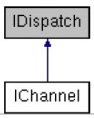
Class List | Class Index | Class Hierarchy | Class Members

Properties | List of all members

IChannel Interface Reference

Measurement Classes

Inheritance diagram for IChannel:



Properties

BSTR	**Name** [get]
BOOL	**Save2MDF** [get, set]
long	**TimeStamp** [get]
VARIANT	**Value** [get]
VARIANT	**xDim** [get]
VARIANT	**yDim** [get]

CANape COM Interface 14.0.30

Interface Description

Main Page | Modules | Classes

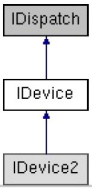
Class List | Class Index | Class Hierarchy | Class Members

Public Member Functions | Properties | List of all members

IDevice Interface Reference

Mesurement and Calibration

Inheritance diagram for IDevice:



Public Member Functions

HRESULT	**WriteMemory** ([in] long addr,[in] VARIANT newVal)
HRESULT	**ReadMemory** ([in] long addr,[in] long size,[out, retval] VARIANT *pVal)
HRESULT	**Telegramm** ([in] BSTR request,[in] long responseTime,[out, retval] VARIANT *pVal)
HRESULT	**RunScript** ([in] BSTR script)
HRESULT	**Upload** ([in] BSTR filename)
HRESULT	**Download** ([in] BSTR filename)
HRESULT	**GoOnline** ([in] BOOL download)
HRESULT	**GoOffline** ()

Properties

BSTR	**Name** [get]
BSTR	**DatabaseFilename** [get]
BSTR	**DriverType** [get]
long	**Channel** [get]
IDispatch	**CalibrationObjects** [get]
IDispatch	**Tasks** [get]
BOOL	**IsOnline** [get]
INT	**DriverTypeByID** [get]
IDatabases	**Databases** [get]
IDispatch	**Diagnostic** [get]
IDispatch	**FlashManager** [get] \par Description: Returns the FlashManager object More...

CANape COM Interface 14.0.30

Interface Description

Main Page | Modules | Classes

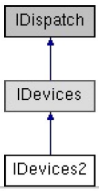
Class List | Class Index | Class Hierarchy | Class Members

Public Member Functions | Properties | List of all members

IDevices2 Interface Reference

Mesurement and Calibration

Inheritance diagram for IDevices2:



Public Member Functions

HRESULT **Add2** ([in] BSTR DeviceName,[in] BSTR DatabaseFilename,[in] BSTR driverType,[in] long channel,[in] BOOL online,[out, retval] **IDevice** **pVal)

HRESULT **Add3** ([in] BSTR DeviceName,[in] BSTR DatabaseFilename,[in] BSTR driverType,[in] long channel,[in] BOOL online,[in]BOOL enablecache,[out, retval] **IDevice** **pVal)

▸ Public Member Functions inherited from **IDevices**

Properties

IDispatch **NetWorks** [get]

▸ Properties inherited from **IDevices**

CANape COM Interface 14.0.30

Interface Description

Main Page | Modules | Classes

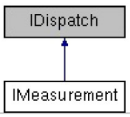
Class List | Class Index | Class Hierarchy | Class Members

Public Member Functions | Properties | List of all members

IMeasurement Interface Reference

Measurement Classes

Inheritance diagram for IMeasurement:



Public Member Functions

HRESULT **Start** ()

HRESULT **Stop** ()

Properties

BOOL **Running** [get]

BSTR **MDFFilename** [get, set]

long **FifoSize** [get, set]

long **SampleSize** [get, set]

BOOL **SyncMode** [get, set]

BOOL **ResumeMode** [get]

eMeasurementState **MeasurementState** [get]

BOOL **UseNAN** [get, set]

CANape COM Interface 14.0.30

Interface Description

Main Page | Modules | Classes

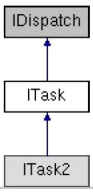
Class List | Class Index | Class Hierarchy | Class Members

Public Member Functions | Properties | List of all members

ITask Interface Reference

Measurement Classes

Inheritance diagram for ITask:



Public Member Functions

HRESULT	**NextSample** ([out] LONG *pTimeStamp,[out, retval] VARIANT *pVal)
HRESULT	**CurrentValues** ([out] LONG *pTimeStamp,[out, retval] VARIANT *pVal)

Properties

BSTR	**Name** [get]
long	**ID** [get]
IDispatch	**Channels** [get]
long	**SamplingTime** [get, set]
VARIANT	**FifoLevel** [get]