Bachelorarbeit

Holger Buchfink

Grafische Busanalyse auf unterschiedlichen Protokollebenen
am Beispiel von IPMB und I2C

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Holger Buchfink

Grafische Busanalyse auf unterschiedlichen Protokollebenen am Beispiel von IPMB und I2C

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Franz Korf
Zweitgutachter : Prof. Dr. rer. nat. Stephan Pareigis

Abgegeben am 4. Januar 2010

Holger Buchfink

Thema der Bachelorarbeit

Grafische Busanalyse auf unterschiedlichen Protokollebenen am Beispiel von IPMB und I2C

Stichworte

Busanalyse, Protokollanalyse, Bussysteme, Protokolle, Protokolldekodierung, IPMB, I2C

Kurzzusammenfassung

Gegenstand dieser Arbeit ist der Entwurf und die Implementierung vom softwarebasierten Analysemodule eines Busanalysewerkzeugs am Beispiel vom IPMB und I2C Protokoll. Das Analysemodul ermöglicht die grafische Busanalyse der Protokollebenen eines Protokollstacks. Die Protokollebenen werden auf einer gemeinsamen Zeitachse dargestellt, um einen nahtlosen Übergang zwischen den Protokollebenen zu ermöglichen. Dadurch können die einzelnen Protokollebenen und deren Zusammenspiel untereinander sichtbar gemacht werden. Durch Zoom- und Scrollfunktionen können die Protokolldaten navigiert und mit unterschiedlichen Detailstufen angezeigt werden. Der Entwurf erfolgt allgemein, so dass eine Anpassung für weitere Protokolle begünstigt wird. Das entwickelte Analysemodul setzt sich u. a. aus den Funktionsbereichen Protokolldatenimport, Protokolldekodierung, Protokolldatenhaltung, Protokollfilter, Protokolldatenmapping und Protokolldatenausgabe zusammen.

Holger Buchfink

Title of the paper

Graphical Bus Analysis of Different Protocol Layers Exemplified with IPMB and I2C

Keywords

bus analysis, protocol analysis, bus system, protocols, protocol decode, IPMB, I2C

Abstract

The object of this thesis is the design and implementation of a software based analysis module that is part of a bus analysis tool and is exemplified with the IPMB and I2C protocol. The module makes it possible to graphically analyze the protocol layers of a protocol stack. The same timeline is used to display the protocol layers to allow a seamless transition between layers. This approach allows the protocol layers and their interactions to become visible. The protocol data can be navigated with zoom and scroll functions and displayed with different levels of detail. The design is kept general to allow further adaptation of other protocols. The developed analysis module consists of the functional blocks protocol data import, protocol decode, protocol data storage, protocol filter, protocol data mapping and protocol data output.

Inhaltsverzeichnis

1. Einleitung

1.1 Hintergrund

Der Einsatz von Systemen mit Mikrocontroller basierten Steuer- und Regelfunktionen nimmt stetig zu. Serielle Bussysteme wie IPMB (Intelligent Platform Management Bus, [ipmb]), I2C (Inter-Integrated Circuit, [i2c]) und SPI (Serial Peripheral Interface, [spi]) werden u. a. eingesetzt, um die Kommunikation zwischen den Modulen solcher Systeme zu ermöglichen. Die Konfiguration der Systeme reicht von der Übertragung einfacher Steuersignale bis hin zu komplexen verteilten Systemen mit vielen Teilnehmern. Die Kommunikation der Teilnehmer wird dabei durch ein Protokoll bzw. Protokollstack bestimmt. Ein Protokollstack besteht aus Protokollebenen, die aufeinander aufbauen und dabei Abstraktionsebenen bilden. Jede Protokollebene übernimmt eine bestimmte Funktion. Die oberste Protokollebene z.B. stellt den Busteilnehmern ein Interface zur Verfügung, das die direkte Adressierung anderer Busteilnehmer und den objektbasierten Austausch von Informationen durch logische Kommunikationskanäle ermöglicht. Die darunterliegende Protokollebene stellt z.B. die Korrektheit der Übertragungen durch den Einsatz von Prüfsummen sicher. Die unterste Protokollebene überträgt die Informationen bitseriell auf der physikalischen Busebene. Die Protokollebenen kodieren die Übertragung beim Senden und dekodieren sie beim Empfang entsprechend. Dadurch können physikalische Kommunikationskanäle durch logische ersetzt werden. Durch diese Arbeitsweise werden einerseits leistungsfähige Kommunikationsschnittstellen für die Busteilnehmer ermöglicht, andererseits können die Bussysteme mit einer minimalen Anzahl an physikalischen Komponenten hergestellt werden. Es führt jedoch auch dazu, dass die Komplexität der Übertragungsprotokolle entsprechend ansteigt. Dadurch entsteht ein Bedarf an leistungsfähigen Analysewerkzeugen, mit denen die Übertragungsprotokolle solcher Systeme analysiert werden können.

1.2 Motivation

Ein Protokollstack wie IPMB besteht aus den Protokollebenen I2C und IPMB. Die IPMB Ebene stellt den Busteilnehmern u. a. Nachrichtenklassen und Kommunikationsobjekte zur Verfügung. Die IPMB Ebene baut auf der I2C Protokollebene auf. Die I2C Protokollebene ist u. a. für die physikalische Datenübertragung auf dem Bussystem zuständig. Die Kommunikation auf solchen Bussystemen setzt das Zusammenspiel der Protokollebenen voraus. Die I2C Implementierung kann z.B. durch einen Mikrocontroller bereitgestellt werden. Die IPMB Implementierung kann durch eine Softwarebibliothek auf dem Mikrocontroller umgesetzt werden. Die korrespondierenden Protokollebenen der einzelnen Busteilnehmer können Implementierungen unterschiedlicher Hersteller verwenden. Die Mikrocontroller können stark integriert sein und nur eingeschränkte oder keine Ausgabemöglichkeiten zur Fehleranalyse bieten. Die Übertragung auf dem Bussystem ist durch den Einsatz der Protokollstacks entsprechend kodiert. Um die Validierung und Fehleranalyse bei solchen Systemen effektiv durchführen zu können, ist es von großem Vorteil, wenn die Möglichkeit besteht, die einzelnen Protokollebenen und deren Zusammenspiel sichtbar zu machen. Die zurzeit am Markt erhältlichen Lösungen bieten diese Möglichkeiten nicht oder nur unzureichend.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist es, dass softwarebasierte Analysemodul eines Busanalysewerkzeugs für das IPMB und I2C Protokoll zu entwickeln. Die Software soll es ermöglichen, die einzelnen Protokollebenen grafisch zu analysieren. Die Darstellung der Protokollebenen auf einer gemeinsamen Zeitachse soll einen nahtlosen Übergang zwischen den Protokollebenen ermöglichen. Durch den Einsatz von Zoom- und Scrollfunktionen soll das Navigieren der Protokollebenen mit unterschiedlichen Detailstufen ermöglicht werden. Das Bild 1.3.b zeigt ein Busanalysewerkzeug aufgeteilt in zwei Module, die Hardware zur Erfassung der Protokolldaten einer Übertragung auf einem Bussystem und das durch diese Arbeit zu entwickelnde Softwaremodul, das für die Analyse der Protokolldaten zuständig ist. Die Lösung soll dabei so entworfen werden, dass eine Anpassung für weitere Protokolle bzw. Protokollebenen begünstigt wird.

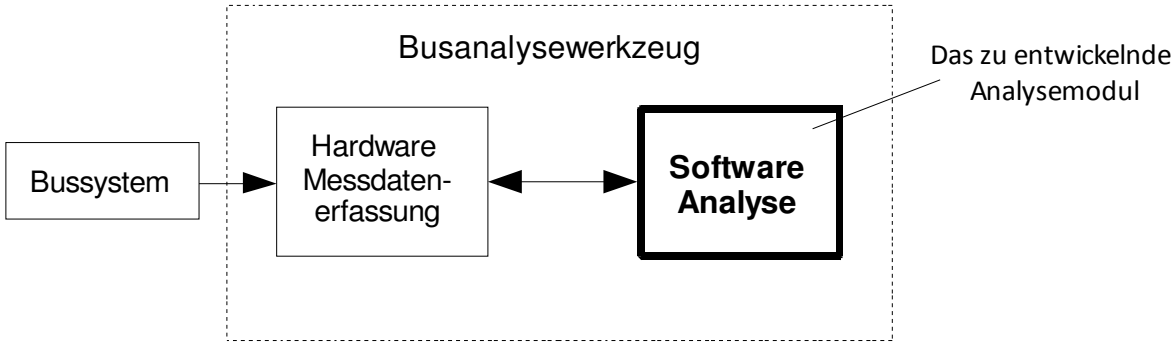

Bild 1.3.b: Module des Busanalysewerkzeugs

2. Analyse

In diesem Kapitel wird zunächst auf die Eigenschaften von Bussystemen und deren Protokolle eingegangen. Es werden allgemeine Eigenschaften von Busanalysewerkzeugen vorgestellt, sowie die Messdatenformate, die verwendet werden, um die Protokolldaten zu erfassen. Danach werden die Fähigkeiten aktueller Busanalysewerkzeuge untersucht und in Form einer Marktübersicht vorgestellt. Anschließend werden Themengebiete mit ähnlichen Problemstellungen und deren Lösungswege untersucht.

Das Kapitel ist folgendermaßen aufgebaut:

- Bussysteme und Protokolle
- Busanalysewerkzeuge
- Messdatenformate Protokolldaten
- Marktübersicht Busanalysewerkzeuge
- Angrenzende Themengebiete

2.1 Bussysteme und Protokolle

2.1.1 Bussysteme

Bei einem Bussystem kommunizieren alle Teilnehmer über eine gemeinsame Busleitung, wie Bild 2.1.1.a zeigt. Jeder Teilnehmer empfängt das gleiche, beim Senden müssen sich die Teilnehmer den Bus teilen. (vgl. [schnell], S.3f)

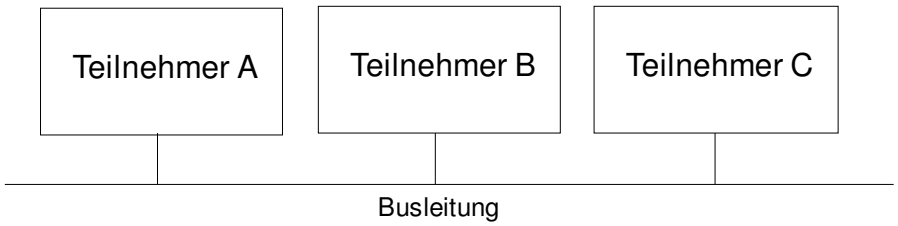

Bild 2.1.1.a: Bussystem mit gemeinsamer Busleitung

2.1.2 Protokollstack

Der Ablauf der Kommunikation der Teilnehmer untereinander, wird durch ein Protokoll bestimmt. Ein Protokoll lässt sich i.d.R. in mehrere Ebenen einteilen, diese Ebenen erfüllen bestimmte Aufgaben. Das ISO OSI Modell (Bild 2.1.2.b) z.B., definiert sieben Schichten bzw. Ebenen und entsprechende Aufgaben. (vgl. [osi] s.32f]) Das Modell dient nur als Referenz. Die tatsächliche Implementierung eines Protokolls und die entsprechende Einteilung und Aufgaben der Schichten sind nicht festgelegt. Es wird von einem Protokollstack gesprochen, da die Protokolle bzw. Protokollebenen aufeinander aufbauen.

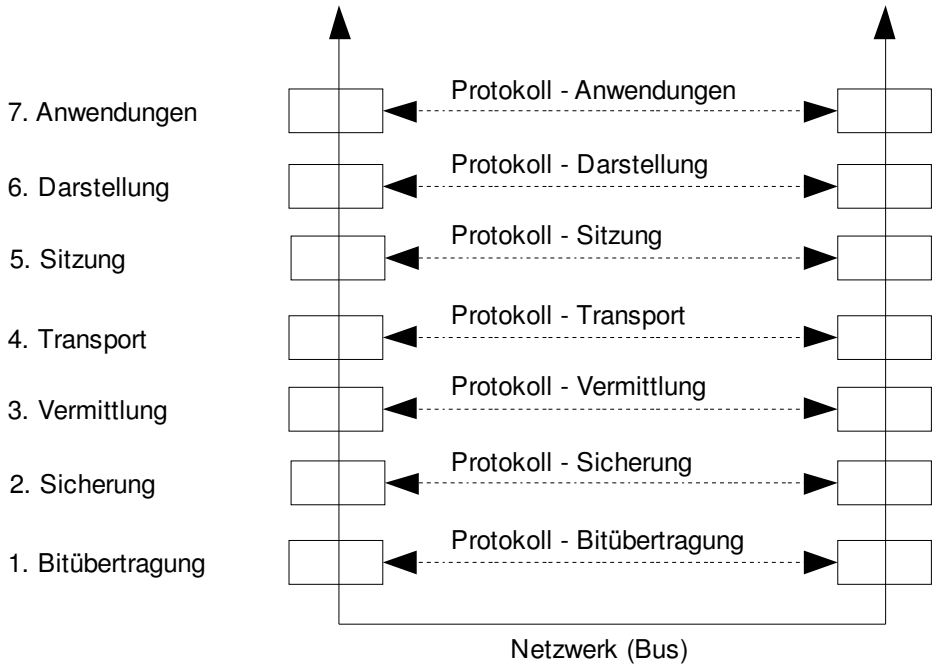

Bild 2.1.2.b: ISO OSI Modell

Die Protokollebenen kommunizieren miteinander. Die 3. Ebene von Teilnehmer A z.B. kommuniziert mit der 3. Ebene von Teilnehmer B. Praktisch werden die Daten aber beim Senden an die darunterliegende Ebene weitergereicht und beim Empfang von der darunterliegenden Ebene entgegengenommen. Auf der untersten Ebene findet die tatsächliche Übertragung statt.

Eine Ebene kodiert ihre Daten beim Senden so, dass die gleiche Ebene beim Empfang die Daten dekodieren bzw. wiederherstellen kann. Dazu können die Ebenen die zu übertragenen Daten um Steuerinformationen wie Header und Trailer ergänzen. Ein Header z.B. kann u. a. die Anzahl der übertragenen Bytes enthalten. Bild 2.1.2.c zeigt die tatsächlich übertragenden Bits auf der untersten Ebene beim Übertragen einer Nachricht. (vgl. [tanenbaum_ds] S. 60f)

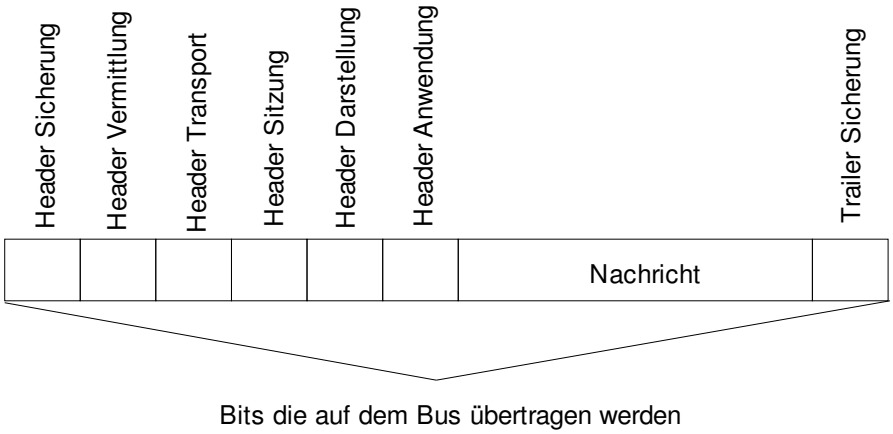

Bild 2.1.2.c: Kodierung einer Nachricht bei der Übertragung

2.1.3 I2C Protokoll

Der I2C Bus (Inter-IC Bus)[i2c] ist ein von NXP entwickelter serieller Datenbus. Der I2C Bus benötigt zwei Busleitungen, Serial Data Line (SDA) und Serial Clock Line (SCL). Die Datenübertragung erfolgt seriell, 8-Bit basiert. Jeder Busteilnehmer besitzt eine eindeutige durch Software adressierbare Adresse. Der Bus ist als Master-Slave-Bus konzipiert, die Arbitrierung ist per Spezifikation geregelt. Bild 2.1.3.a zeigt eine I2C Bus Beispielkonfiguration.

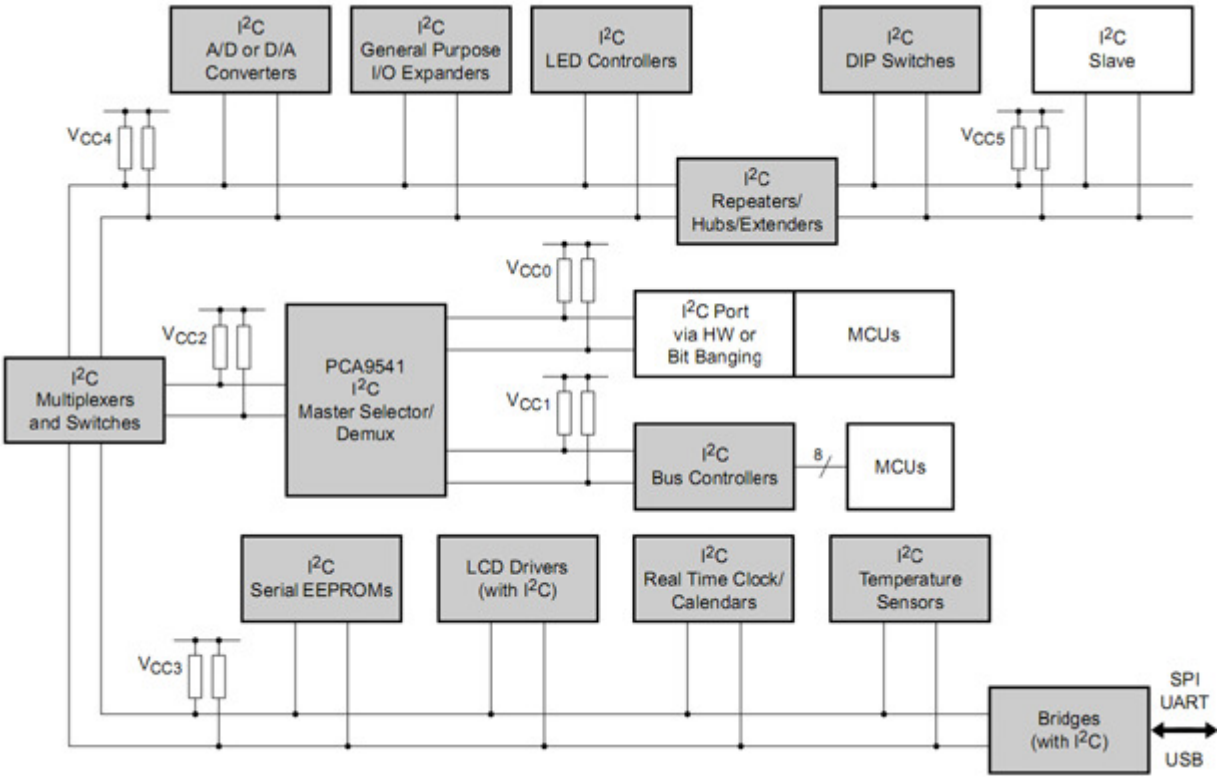

Bild 2.1.3.a: I2C Bus Beispielkonfiguration [i2c]

Adressierung und Datenübertragung

Bild 2.1.3.b zeigt eine Datenübertragung auf dem I2C Bus. Die Übertragung erfolgt als Lese- oder Schreibvorgang. Der Master sendet eine 7-Bit Slave Adresse, gefolgt von einem R/W Bit, das einen Lese- oder Schreibvorgang bestimmt. Nach jedem Byte folgt ein Acknowledge Bit. Das Acknowledge Bit erlaubt es dem Empfänger, das empfangene Byte gegenüber dem Sender zu bestätigen. Anschließend folgen beliebig viele Bytes, die gelesen bzw. geschrieben werden entsprechend dem R/W Bit. Um eine Übertragung zu starten bzw. zu stoppen, wird ein Start- bzw. Stopsignal vom Master übertragen (Bild 2.1.3.c).

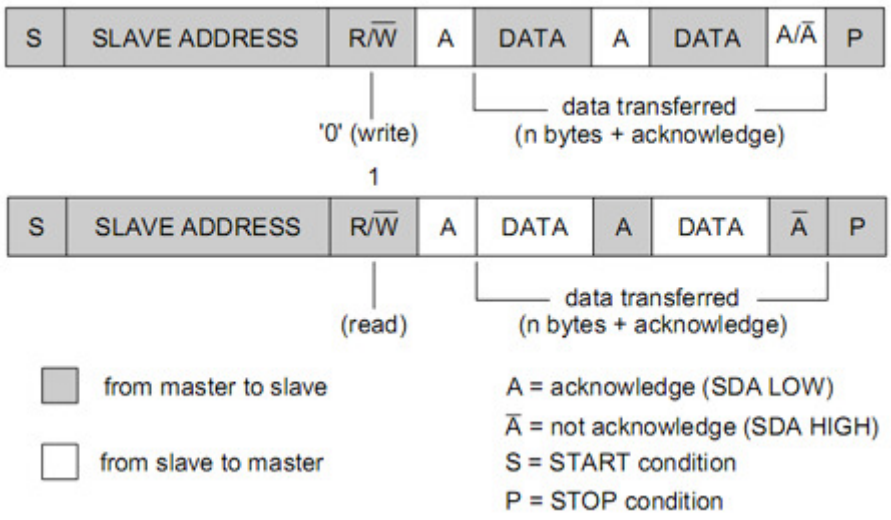

Bild 2.1.3.b: I2C Bus Adressierung und Datenübertragung [i2c]

Physikalische Protokollebene

Bild 2.1.3.c zeigt ein Start- und ein Stopsignal. Mit dem Startsignal wird eine Datenübertragung eingeleitet, mit dem Stopsignal entsprechend beendet. Beim Startsignal wird die SDA Leitung LOW gezogen, während SCL High ist. Beim Stopsignal wird SDA HIGH gezogen, während SCL HIGH ist.

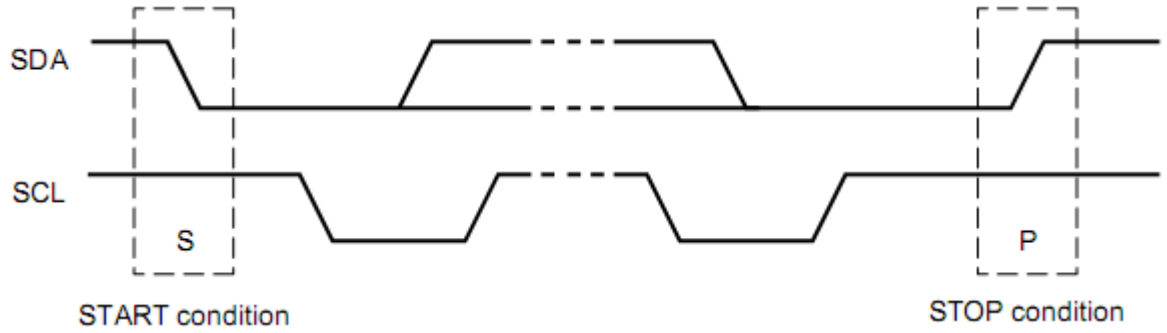

Bild 2.1.3.c: I2C Bus Start und Stop Ereignis [i2c]

Bild 2.1.3.d zeigt die Zustände der Busleitungen SDA (Data)und SCL (Clock) bei der Bit Übertragung. Um ein Bit zu übertragen, muss die SDA (Data) Leitung das zu übertragene Bit unverändert halten für die Dauer einer Clock HIGH Phase der SCL (Clock) Leitung. Während die SCL Leitung LOW ist, kann der SDA Zustand für das als nächstes zu übertragende Bit verändert werden.

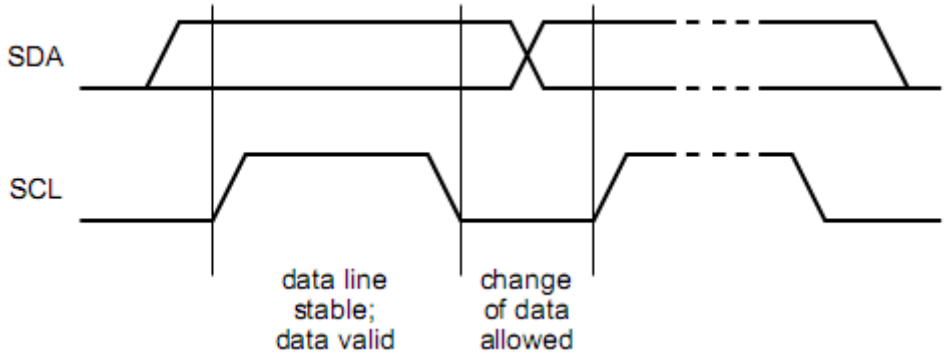

Bild 2.1.3.d: I2C Bus Bit Kodierung [i2c]

Bild 2.1.3.e zeigt die Zustände der Busleitungen SDA und SCL bei einer vollständigen Übertragung.

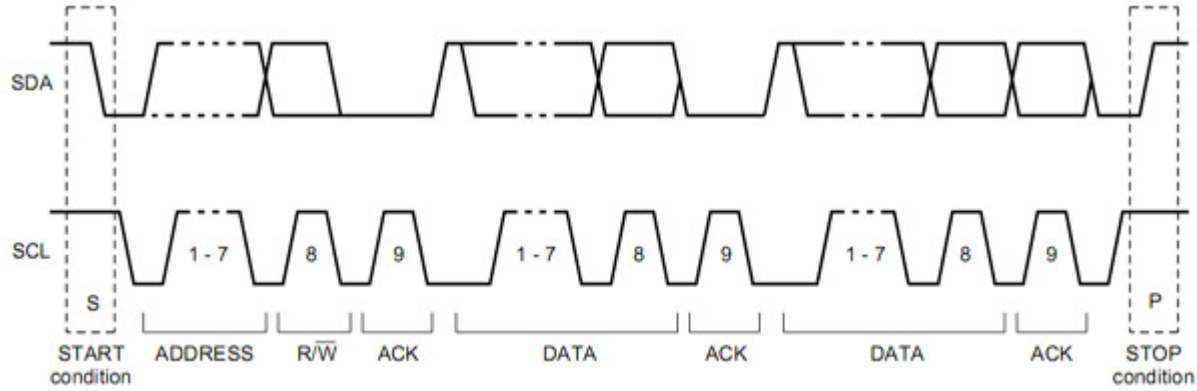

Bild 2.1.3.e: I2C Bus vollständige Datenübertragung [i2c]

Die Pegel der logischen Zustände HIGH und LOW sind nicht festgelegt und von der Übertragungshardware abhängig. Es können u. a. CMOS, NMOS und Bipolar I/Os verwendet werden. Der LOW Zustand gilt ab 30%, der HIGH Zustand ab 70% vom Maximalpegel.

Der I2C Bus arbeitet mit bis zu 100 kbit/s im Standard-mode, bis zu 400 kbits/s im Fast-mode, bis zu 1 Mbit/s in Fast-mode Plus und bis zu 3.4 Mbit/s im High-speed mode.

Die I2C Bus Spezifikation [i2c], beschreibt u. a. die folgenden weiteren Eigenschaften, auf die hier nicht näher eingegangen wird:

- Repeated start: Erneutes Einleiten einer Übertragung ohne Stopsignal.
- Clock streching: Verzögern der Übertragung durch das Halten von SCL auf LOW.
- 10-Bit Adressing: Erweiterter Adressraum von 10-Bit für Slaves.
- Multi-Master: Mehrere Master auf einem Bus.
- Arbitration: Regelt das Zusammenspiel mehrerer Master auf einem Bus.

2.1.4 IPMB Protokoll

IPMB (Intelligent Platform Management Bus, [ipmb]) spezifiziert eine Byte basierte Übertragung für IPMI (Intelligent Platform Management Interface, [ipmi]) Nachrichten durch I2C Geräte. Das IPMB Protokoll definiert einen Protokollstack Bild 2.1.4.a, der aus einer IPMB Protokollebene besteht, die auf dem 100 kbit/s I2C Protokoll aufbaut.

IPMI stellt standardisierte Schnittstellen für das Platform Management bereit. Es wird z.B. in Serversystemen eingesetzt zur Steuerung und Überwachung der Systemkomponenten wie Hauptplatine, Lüfter, Netzteile, Festplatten, usw.

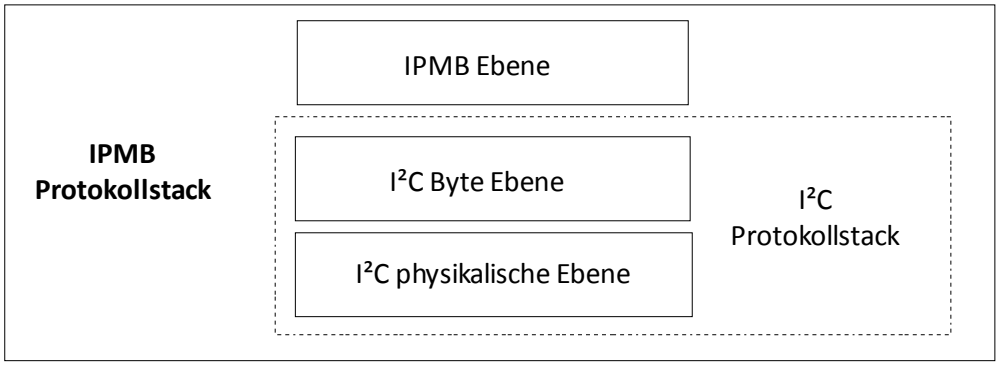

Bild 2.1.4.a: IPMB Protokollstack

Eine IPMB Transaktion besteht aus einem Request und Response, wie Bild 2.1.4.b zeigt.

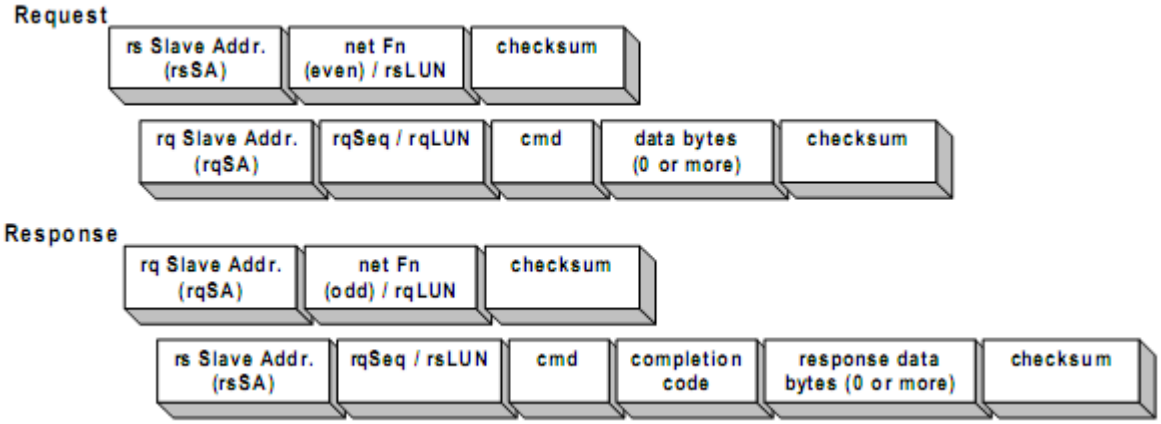

Bild 2.1.4.b: IPMB Request/Response Format [ipmb]

Die Felder in Bild 2.1.4.b und 2.1.4.c haben folgende Bedeutung:

- rs Slave Addr.: Responder's Slave Address
- net FN: Net Function – Funktionsgruppe
- checksum: Checksum für Header bzw. Data

- rq Slave Addr.: Requester's Slave Address
- rqSeq: Request sequence counter
- LUN: Logical Unit Number – Erweiterung vom Adressraum
- cmd: Command – Auszuführende Funktion
- data bytes: Nutzdaten

Bei einem Request werden u. a. eine Funktionsgruppe und eine Funktion übertragen. Die Funktion gibt die auszuführende Funktion der Funktionsgruppe an. Die Funktionsgruppen sind u. a. Chassis, Bridge, Sensor, Application, Firmware, Storage und Vender specific (OEM). Der Aufbau und die Bedeutung der übertragenen *data bytes* ist für die Funktionsgruppen durch die IPMI/IPMB Spezifikation vorgegeben, bis auf die Application, Firmware und OEM Funktionsgruppen. Bei diesen Funktionsgruppen kann der Aufbau und die Bedeutung der *data bytes* implementierungsabhängig gewählt werden.

Das Bild 2.1.4.c zeigt ein Beispiel der übertragenen Bytes bei einem IPMB Request/Response.

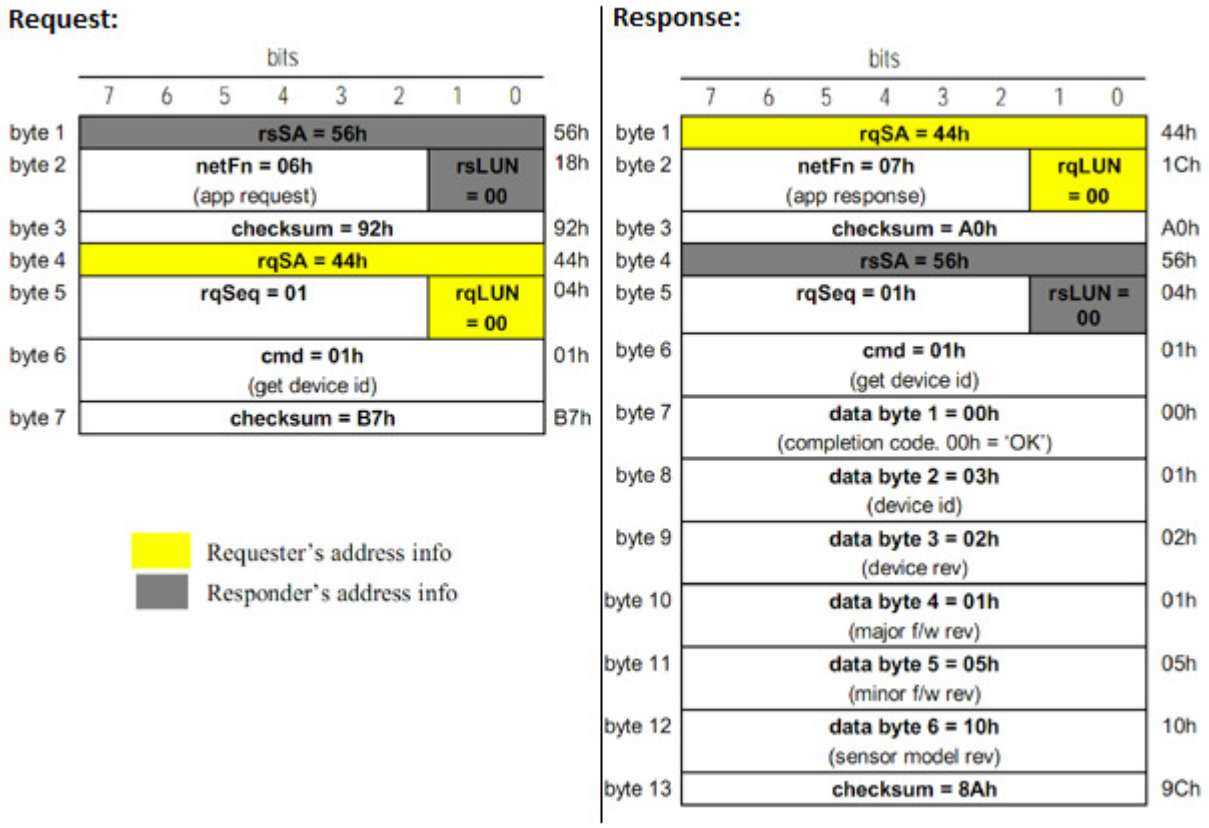

Bild 2.1.4.c: IPMB Request/Response Beispiel [ipmb]

2.2 Busanalysewerkzeuge

2.2.1 Allgemeiner Aufbau

Die allgemeinen Anforderungen an ein Busanalysewerkzeug, bestehen aus dem Erfassen und Auswerten der Kommunikation auf einem Bus. Darunter fallen Aufgaben wie Validierung und Fehleranalyse. Analysewerkzeuge werden aber auch allgemein eingesetzt, um ein Verständnis der Vorgänge einer Übertragung auf einem Bussystem zu erlangen. Bild 2.2.1.a zeigt den allgemeinen Aufbau eines Busanalysewerkzeugs.

Bild 2.2.1.a: Allgemeiner Aufbau Busanalysewerkzeug

Der Aufbau besteht aus den folgenden Bereichen:

Erfassen der Protokolldaten

- Wie im vorherigen Abschnitt 2.1.2 beschrieben, sind die Protokollebenen hierarchisch aufgebaut. Die Daten der untersteten Protokollebene beinhalten alle höheren Protokollebenen. Um alle Protokollebenen auswerten zu können, müssen die Daten der untersten Protokollebene erfasst werden.

Auswerten der Protokolldaten

- Damit die Protokollebenen ausgewertet und ausgegeben werden können, müssen die Messdaten bzw. Protokolldaten entsprechend der kodierten Protokollebenen wieder dekodiert werden.

2.3 Messdatenformate Protokolldaten

Im Folgenden werden zwei Datenformate beschrieben, samplebasierte und ereignisbasierte Formate, mit denen die Messdaten bzw. Protokolldaten gespeichert werden können.

2.3.1 Samplebasierte Protokolldaten

Bei einem samplebasierten Format wird der Spannungspegel einer Busleitung mit einer konstanten Abtastrate erfasst. Zu den Abtastzeitpunkten werden Daten erfasst egal ob sich der Spannungspegel der Busleitung ändert oder nicht. Bei diesem Format werden die Abtastwerte üblicherweise in einem Array basierten Datentyp gespeichert. Daten von AD-Wandlern (Analog-Digital Wandlern) liegen üblicherweise in einem Samplebasierten Format vor. Bild 2.3.1.a zeigt den Verlauf eines Spannungspegels bei einem Flankenwechsel von LOW nach HIGH und die entsprechenden Abtastzeitpunkte, bei dem der Spannungspegel erfasst wird.

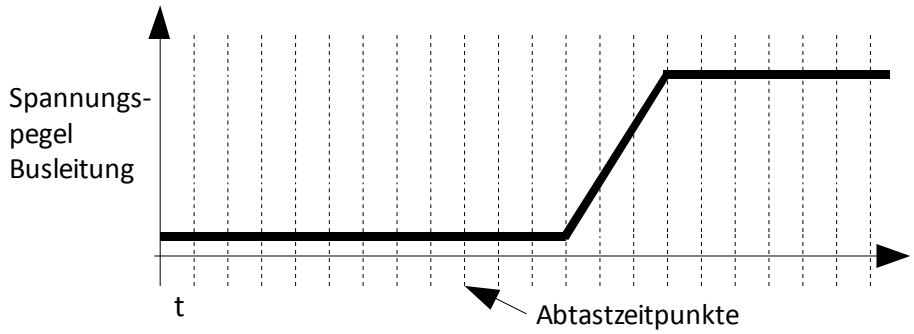

Bild 2.3.1.a: Abtastzeitpunkte eines AD-Wandlers

Die Abtastrate richtet sich nach dem zu beobachteten Signal. Eine sinnvolle Abtastrate, mit der die Signale auf einer Busleitung erfasst werden können, kann durch die Protokollspezifikation und der entsprechenden Übertragungsgeschwindigkeit vom Bussystem bzw. der Busleitung ermittelt werden. Eine sinnvolle Abtastrate für die Busanalyse ist i.d.R. um ein vielfaches höher als die minimale Abtastrate, die nötig ist, um die übertragenen Daten zu erkennen, da die tatsächlichen Signalverläufe der Busleitungen ausgewertet und wiedergegebenen werden sollen.

Um bestimmte Probleme wie Peaks zu erkennen, bei dem Signalspitzen kurzzeitig auftreten, die auch durch externe Störquellen verursacht werden können, sind u. U. sehr hohe Abtastraten unabhängig der Übertragungsgeschwindigkeit des Bussystems erforderlich. Bei zu niedriger Abtastrate, können Peaks zwischen den Abtastzeitpunkten auftreten und dadurch bei der Auswertung u.U. nicht erkannt werden.

Die Datenrate für samplebasierte Daten kann wie folgt berechnet werden:

$$Datenrate = Abtastrate \cdot Auflösung \cdot Anzahl\ Busleitungen$$

Eine Übersicht der Datenraten liefert das Diagramm 2.3.1.b

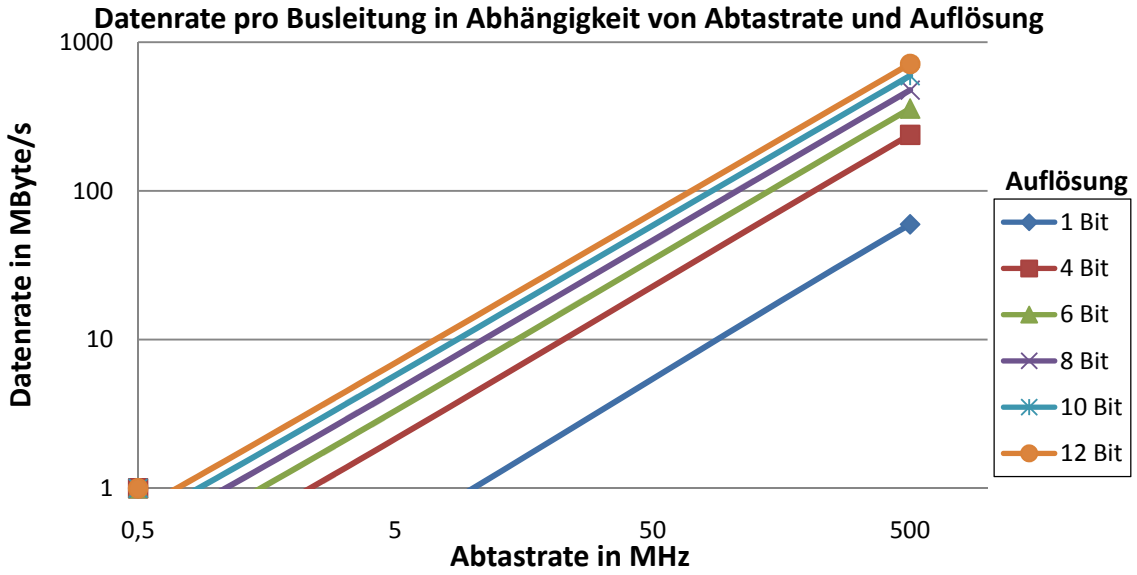

Bild 2.3.1.b: Datenrate pro Busleitung in Abhängigkeit von Abtastrate und Auflösung

Um einen mit 400 kHz betriebenen I2C Bus zu erfassen, kann z.B. eine Abtastrate von 20 MHz gewählt werden. Diese Abtastrate ermöglicht eine zeitliche Auflösung von 50 ns bzw. 50 Abtastpunkte pro Clock Periode vom I2C Bus. Als Auflösung für den Spannungspegel werden 8 Bit pro Abtastpunkt verwendet, was bei einem Spannungspegelbereich von 5 V, eine Auflösung der Spannungspegel mit ca. 0,02 V Schritten ermöglicht.

$$Datenrate\ I2C\ Bus\ Abtastrate\ 20\ Mhz/8Bit =$$

$$Abtastrate \cdot Auflösung \cdot Anzahl\ Busleitungen\ =$$

$$20\ Mhz\ \cdot 8\ Bit \cdot 2 = 38{,}15\ MByte/s$$

2.3.2 Ereignisbasierte Protokolldaten

Bei einem ereignis- oder eventbasierten Format werden nur Ereignisse festgehalten. Ein Ereignis wird durch einen Zeitpunkt und ggf. durch weitere Eigenschaften beschrieben. Welche Ereignisse festgehalten werden, wird durch Regeln bestimmt. Eine Ereignisregel ist z.B. das Überschreiten eines bestimmten Pegels auf einer Busleitung wie Bild 2.3.2.a zeigt. Die ereignisbasierten Daten werden wie die samplebasierten Daten durch Abtasten der Busleitungen gewonnen, es werden jedoch nur die entsprechenden Ereignisse gespeichert. Das hat den Vorteil, dass die Datenraten i.d.R. geringer sind als bei samplebasierten Daten. Der Nachteil ist, die Signalverläufe der physikalischen Protokollebene werden nicht bzw. nur eingeschränkt gespeichert. Es werden z.B. Ein- und Ausschwingvorgänge der Flankenwechsel auf einer Busleitung nicht gespeichert.

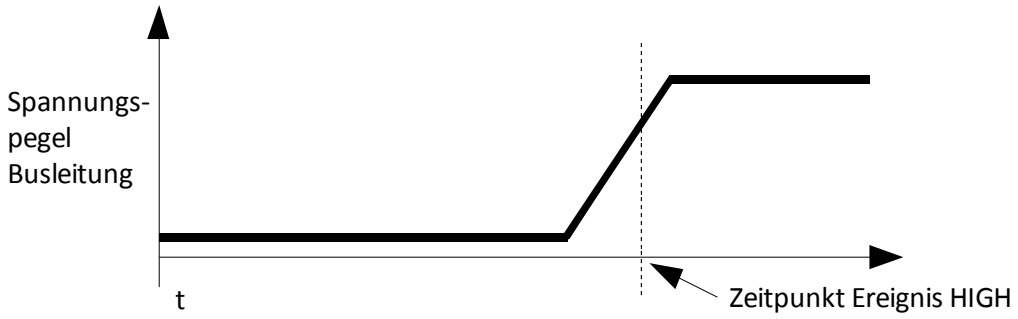

Bild 2.3.2.a: Ereignisbasierte Daten - Ereigniszeitpunkt

Die Datenrate für ereignisbasierte Formate lässt sich nicht genau festlegen, da die Datenrate u. a. von der Menge an Ereignissen abhängt bzw. von der Datenrate auf dem Bus. Es kann jedoch eine Obergrenze für den Regelbetrieb von Bussystemen mit Clockleitung wie folgt berechnet werden. Bild 2.3.2.b zeigt eine Clock Periode von einem I2C Bus als Beispiel. Bei dieser Betrachtung wird davon ausgegangen, dass auf der SDA (Data) Busleitung ein Zustandswechsel Pro Clock Periode stattfindet. Auf der SCL (Clock) Busleitung wird davon ausgegangen, dass zwei Zustandswechsel pro Clock Periode auftreten:

- Die Taktleitung SCL hat zwei Zustandswechsel pro Periode => 2 Ereignisse für die Taktleitung pro Periode
- Für jede Datenleitung SDA wird ein Zustandswechsel pro Periode angenommen => 1 Ereignis pro Datenleitung für jede Periode

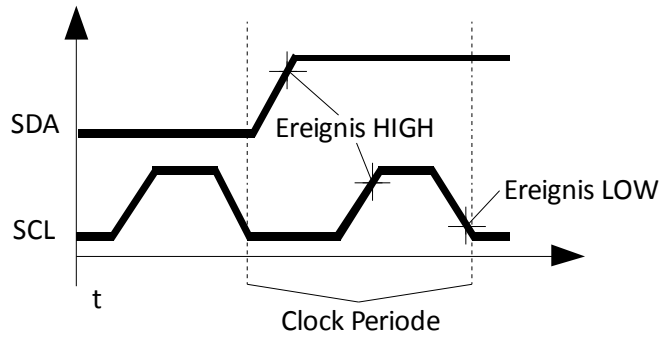

Bild 2.3.2.b: Ereignisse für eine Clock Periode auf einem I2C Bus

Bei diesem Beispiel wird angenommen, dass 72 Bit benötigt werden um ein Ereignis zu speichern. Die 72 Bit ergeben sich aus folgendem:

- 64 Bit für den Ereigniszeitpunkt
- 4 Bit für den Zustand (High, Low,…)
- 4 Bit für die Busleitung (SDA, SCL)

Für einen I2C Bus bei 400 kHz z.B. beträgt die Datenrate für ein eventbasiertes Format 10,3 MByte/s:

$$Datenrate\ I2C\ Beispiel\ =$$

$$Clock\ (Hz) \cdot\ Anzahl\ Ereignisse\ (pro\ Periode)\ \cdot\ Ereignisgrösse\ (Byte)\ =$$

$$400\ kHz\ \cdot\ 3\ Ereignisse\ \cdot\ 9\ Byte\ =\ 10{,}3\ MByte/s$$

In der Praxis wird die Datenrate i.d.R. geringer ausfallen, da ein Zustandswechsel der Datenleitung bei jeder Periode nicht zu erwarten ist, und die Clockleitung nicht durchgängig betrieben werden muss. Dadurch entstehen entsprechend weniger Ereignisse.

Ein großer Vorteil der ereignisbasierten Formate ist die Möglichkeit, mit sehr hohen Abtastraten die Schaltzeiten der Busleitungen zu erfassen, ohne die Datenrate zu beeinflussen, da sich die Anzahl der Ereignisse nicht durch die Abtastrate verändert. Beim samplebasierten Format führt die Erhöhung der Abtastrate auch zu einer Erhöhung der Datenrate. Damit z.B. die Schaltzeiten der Busleitungen mit 5 ns (200 MHz), anstatt 50 ns (20 MHz) wie beim Oben gezeigtem Beispiel erfasst werden können, bleibt die Datenrate bei den ereignisbasierten Daten gleich (10,3 MByte/s). Für die samplebasierten Daten ändert sich die Datenrate von 38,15 MByte/s auf 381,47 MByte/s.

2.4 Marktübersicht Busanalysewerkzeuge

Im Folgenden werden die Fähigkeiten der aktuell am Markt erhältlichen Busanalysewerkzeuge anhand einer Marktübersicht vorgestellt. Im Anschluss erfolgt eine Zusammenfassung und Bewertung.

Bei der Marktübersicht werden folgende Punkte berücksichtigt:

- Welche Bussysteme bzw. Protokolle unterstützt werden.
- Auf welcher Protokollebene und mit welcher Genauigkeit die Daten erfasst werden.
- Welche Protokolleben dekodiert werden und wie ein Bezug zwischen den Ebenen hergestellt wird.
- Welche Auswertungsfunktionen möglich sind.
- Wie neue Bussysteme bzw. Protokollebenen erweitert werden können.

Um die Stärken und Schwächen der Lösungen verständlicher darstellen zu können, wird folgendes vereinfachtes Beispielszenario verwendet (Bild 2.4.a):

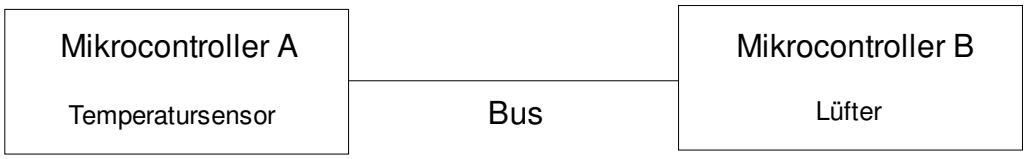

Bild 2.4.a: Beispielszenario Bussystem

Mikrocontroller A besitzt einen Temperatursensor. Beim Überscheiten einer bestimmten Temperatur sendet Mikrocontroller A eine Nachricht über das Bussystem an Mikrocontroller B, Mikrocontroller B schaltet daraufhin einen Lüfter ein. Auf dem Bussystem kommunizieren weitere Mikrocontroller mit ähnlichen Funktionen.

Es wird festgestellt, dass der Lüfter von Mikrocontroller B nicht immer anspringt, obwohl die Temperaturschwelle beim Temperatursensor überschritten wurde. Die Aufgabe eines Busanalysewerkzeugs ist es nun, die Übertragung der Steuerinformation von Mikrokontroller A an B auszugeben, damit der Fehler weiter untersucht werden kann.

Die im Folgenden vorgestellten Geräte bzw. Lösungen sollen repräsentativ für bestimmte Funktionen bzw. Geräteklassen sein, i.d.R. gibt es ähnliche Lösungen auch von anderen Herstellern.

2.4.1 Oszilloskope

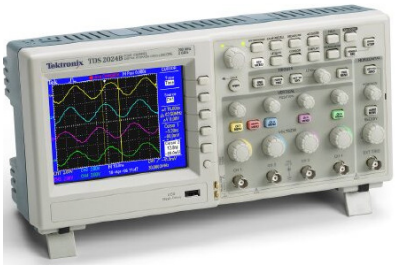

Bild 2.4.1.a: Tektronix Oszilloskop TDS1000 [tds1000]

Mit einem Oszilloskop bzw. Speicheroszilloskop (Bild 2.4.1.a) lassen sich die elektrischen Signale der Busleitungen auf der untersten, physikalischen Protokollebene grafisch auswerten. Durch die Arbeitsweise der Geräte ist die Aufzeichnungsdauer begrenzt. Speicheroszilloskope können zwar einen längeren Signalverlauf erfassen als analoge Geräte, bei aktuellen Geräten liegt die Aufzeichnungsdauer jedoch deutlich unter einer Sekunde. Die Geräte erfassen die Signale durch A/D Wandler und speichern die Daten in einen internen Speicher. Die Standardmodelle wie z.B. das Tektronix TDS1000 bieten Abtastraten von 1 GHz und können Maximal 2500 Abtastpunkte erfassen [tds1000]. Die Abtastung erfolgt mit 8 Bit, daraus ergibt sich eine Datenrate von 1 GByte/s.

Bedingt durch die begrenzte Aufzeichnungsdauer werden Trigger eingesetzt um die Messung zu starten. Ein Trigger startet die Messung, wenn die Triggerbedingung erfüllt ist. Ein Trigger bzw. eine Triggerbedingung kann u. a. eine externe Steuerleitung sein, oder das zu messende Signal selbst sein. Im letzteren Falle wird die Messung ausgelöst, wenn das Signal z.B. einen bestimmten Pegel überschreitet. Um ein Ereignis auf einem Bussystem beobachten zu können, muss ein geeigneter Trigger bestimmt werden.

Um das im Beispielszenario beschriebene Problem mit einem Oszilloskop zu untersuchen, muss eine geeignete Triggermöglichkeit bestimmt werden, die es ermöglicht, die Messung zu starten sobald Mikrokontroller A sendet. Die Information, die Mikrokontroller A sendet, ist jedoch in einer höheren Protokollschicht kodiert, die das Oszilloskop nicht dekodiert. Mit dem Oszilloskop alleine kann kein geeigneter Trigger bestimmt werden. Es muss ein externer Trigger geschaffen werden, was u. U. Zeitaufwendig ist.

Es gibt auch Oszilloskope mit spezialisierten Funktionen. Diese können bestimmte serielle Protokolle erkennen und Dekodieren. Bild 2.4.1.b zeigt ein Agilent MSO mit I2C Protocol Decode Erweiterung [agilent_mso]. Mit dieser Erweiterung kann das I2C Protokoll dekodiert werden. In Bild 2.4.1.b werden die dekodierten I2C Daten unterhalb der gelben Wellenform auf der gleichen Zeitachse dargestellt. Zusätzlich werden im unteren Bildbereich die dekodierten I2C Daten tabellarisch angezeigt. Die Triggerfunktionen werden auch entsprechend erweitert, so kann auf einer Auswahl von Ereignissen der I2C Protokollebene getriggert werden, wie z.B. auf einer bestimmten Adresse der I2C Nachricht.

Die Aufzeichnungsdauer ist jedoch immer noch problematisch. Wenn im Beispielszenario der Fehler nur alle 1000 Nachrichten auftritt, kann es trotz solcher Protokolldekodierungsfunktionen schwer sein, einen geeigneten Trigger zu bestimmen, da u. U. die genaue Fehlerursache nicht bekannt ist und damit der Zeitpunkt, an dem der Fehler auftritt, nicht bekannt ist. Zudem stehen die erweiterten Dekodierungsmöglichkeiten nur für die vom Hersteller angebotenen Protokolle zur Verfügung.

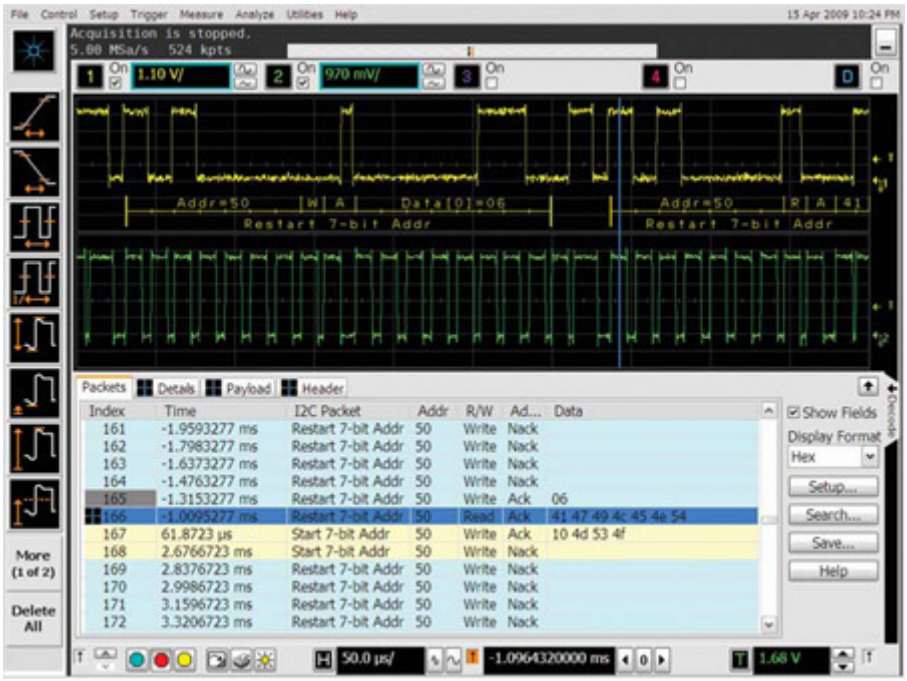

Bild 2.4.1.b Agilent MSO mit I2C Protocol Decode Erweiterung [agilent_mso]

2.4.2 Logic Analyzer

Logic Analyzer unterscheiden sich in ihrer grundlegenden Arbeitsweise nicht von Oszilloskopen. Sie arbeiten jedoch mit digitalen Eingängen, im Gegensatz zu den analogen Eingängen eines Oszilloskops. Die Grenze zwischen den Geräten ist fließend, da es auch Mixed Signal Oszilloskope mit digitalen und analogen Eingängen gibt. Die digitalen Eingänge kennen nur binäre Zustände Hi oder Low, dadurch können die Zustände der Busleitungen in binärer Form angezeigt werden wie Bild 2.4.2.a zeigt. Die beiden unteren Zeilen zeigen die binären Zustände der Busleitungen SCL und SDA von einem I2C Bus dargestellt auf der gleichen Zeitachse.

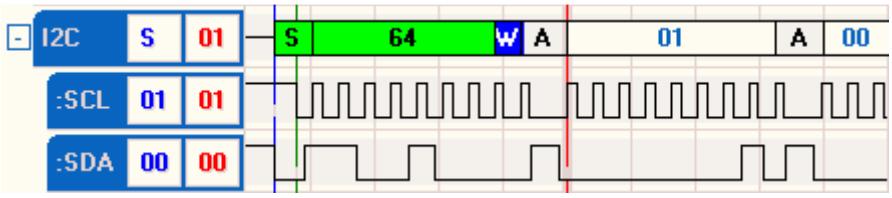

Bild 2.4.2.a DigiView DV3400 Software [digiview]

Bild 2.4.2.a zeigt den Logic Analyzer DigiView DV3400 von TechTools bzw. die DigiView Auswertungssoftware [digiview], damit können bestimmte serielle Protokolle dekodiert angezeigt werden. Die oberste Zeile im Bild zeigt die decodierte I2C Übertragung.

Das DigiView arbeitet mit internem Speicher, um die Messdaten zu speichern. Es hat, wie die Oszilloskope auch, eine begrenzte Aufzeichnungsdauer. Das DigiView verwendet aber eine Kompressionstechnik, die es ermöglicht, die Aufzeichnungsdauer zu verlängern. Die digitalen Signalzustände der Busleitungen lassen sich im Gegensatz zu den Analogen sehr effektiv komprimieren, da sie nur zwei Zustände haben und i.d.R. einen Zustand für eine gewisse Dauer annehmen. Dadurch hängt die Aufzeichnungsdauer im Wesentlichen von dem Datenaufkommen auf dem Bus ab. Das DigiView kann ohne Kompression nur 2.6 ms aufzeichnen (Abtastrate 400 MHz). Mit Kompression bei der gleichen Abtastrate, kann die Aufzeichnungsdauer bis zu 260 Tagen betragen, begrenzt jedoch durch die Busaktivität. Für einen I2C Bus gibt TechTools die Aufzeichnungskapazität mit ca. 10000 auf dem Bus übertragenen Byte an.

2.4.3 Protokoll Analyzer

Ein Protokoll Analyzer hat i.d.R. einen anderen Schwerpunkt als Oszilloskope und Logic Analyzer. Die Analysefunktionen für höhere Protokollebenen stehen im Vordergrund. Die Analyse der physikalischen Protokollebene ist nur eingeschränkt oder gar nicht möglich.

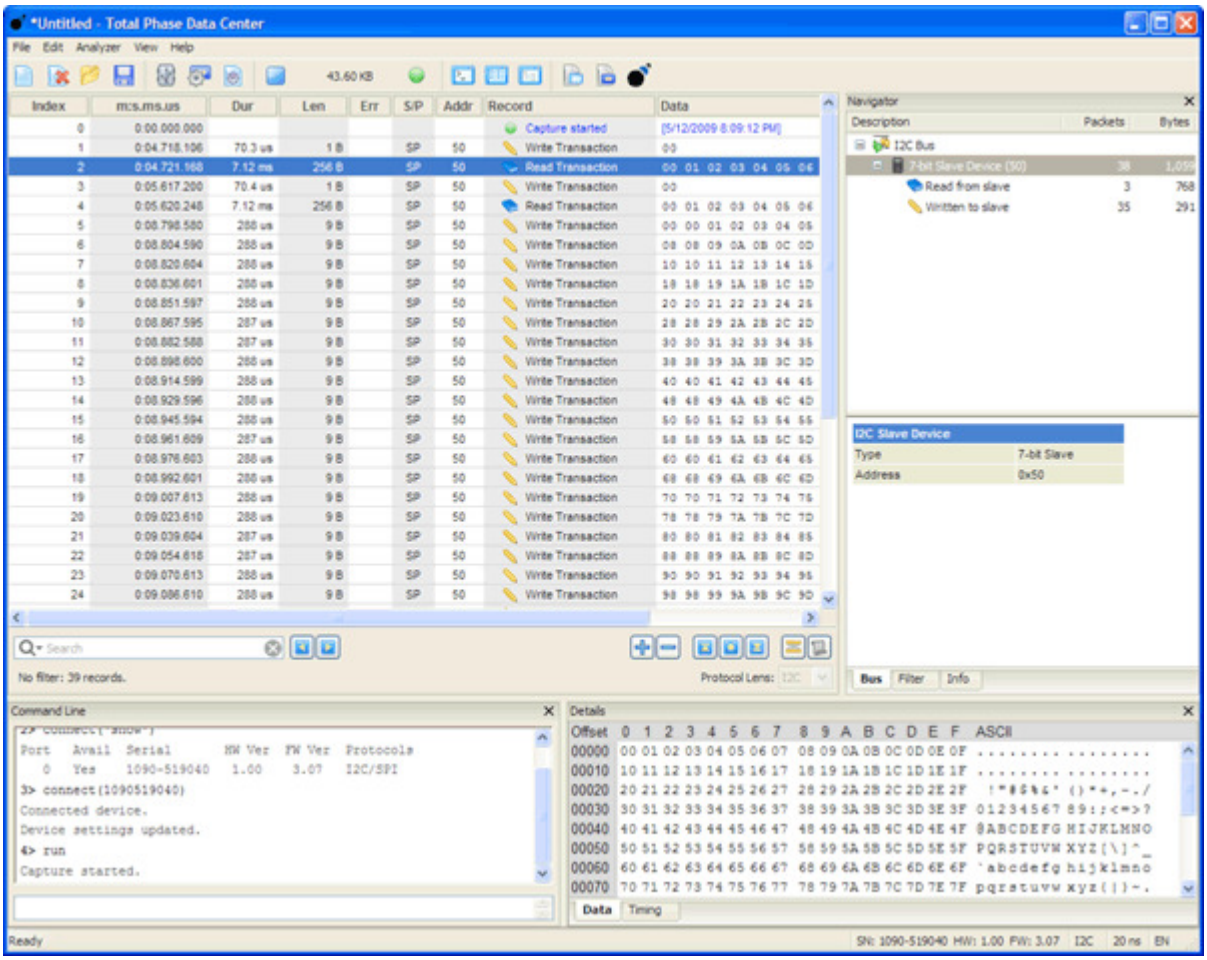

Bild 2.4.3.a: Totalphase Datacenter – I2C [totalphase]

Die Beagle Analyzer von Totalphase bestehen aus einer Hardware zur Messwerterfassung, die mit einem Host PC über USB verbunden wird. Auf dem Host PC arbeitet eine entsprechende Hostsoftware, die die Protokolldaten auswertet (Bild 2.4.3.a). Je nach Produktausführung wird ein bestimmtes Protokoll I2C, SPI oder USB unterstützt. [totalphase]

Die Lösungen bieten für das jeweilige Protokoll gute Auswertungsmöglichkeiten, da die Ausgabe entsprechend der Protokollstrukturen und Protokollereignissen formatiert ist. Die physikalische Protokollebene lässt sich jedoch nicht einsehen. Es lassen sich nur die Bitzeiten der Übertragung auf dem Bus einsehen (Auflösung Bit-Level timing liegt bei 20 ns bei der I2C Ausführung).

Ein großer Vorteil dieser Lösungen ist, dass die mögliche Aufzeichnungsdauer quasi unbegrenzt ist. Die aufgezeichneten Businformationen werden parallel zur Messung zum Hostcomputer übertragen, die Aufzeichnungsdauer ist nur durch die Speicherkapazität vom Hostcomputer begrenzt. Durch die quasi unbegrenzte Aufzeichnungsdauer, ändert sich auch der Arbeitsablauf im Vergleich zum Oszilloskop bzw. Logic Analyzer. Anstatt vor der Messung Trigger zu bestimmen, um ein bestimmtes Ereignis auf dem Bussystem zu erfassen, können hier die aufgezeichneten Daten nach der Messung durch Filter eingeschränkt werden, um bestimmte Ereignisse zu isolieren.

Diese und ähnliche Lösungen konzentrieren sich i.d.R. auf ein bestimmtes Protokoll wie hier I2C (Bild 2.4.3.a), die Nutzdaten, die durch das I2C Protokoll übertragen werden, werden nur als Bytefolge angezeigt. Das erschwert die Auswertung der übertragenen Daten, da diese i.d.R. auch eine Formatierung besitzen bzw. weitere Protokollebenen beinhalten können.

Das I2C Studio von Telos (Bild 2.3.4.b u. 2.4.3.c) [telos] arbeitet ähnlich wie die Totalphase Analyzer. Es bietet jedoch spezielle Funktionen, um bestimmte Nutzdaten der I2C Protokollebene weiter auszuwerten bzw. für die Ausgabe zu formatieren. Das I2C Studio erkennt eine Reihe von ICs, die über den I2C Bus Daten austauschen. Werden Daten gesendet oder empfangen, kann anstatt der übertragenen Bytefolge, die IC Funktion namentlich angezeigt und die übertragenen Werte mit entsprechenden Strukturen und Einheiten angezeigt werden. So kann z.B. beim Auslesen eines ICs mit Temperatursensor die Funktion „Temperatur“ und der Wert „11 °C“ angezeigt werden, anstatt der unformatierten übertragenen Bytefolge. Diese Daten müssen dann nicht mehr manuell aus den Datenblättern zugeordnet werden. Die Beschreibungen der ICs und die jeweiligen Ausgabeformate werden in XML Dateien gespeichert. Der Anwender kann diese Beschreibungen um weitere ICs ergänzen bzw. anpassen. Das Bild 2.4.3.b, zeigt die Daten einer I2C Übertragung als Bytefolge, das Bild 2.4.3.c, zeigt die gleiche Übertragung mit der Interpretationsfunktion aktiviert.

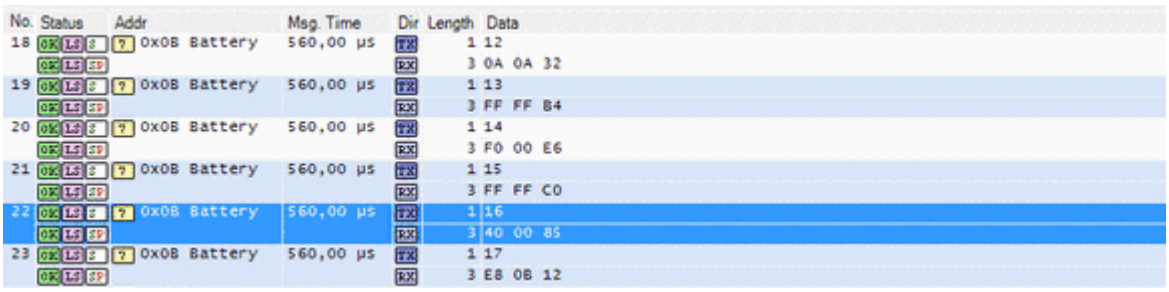

Bild 2.4.3.b: Telos I2C Studio Raw Payload View [telos]

No.	Status	Addr	Msg. Time	Dir	Value	Data	Registers
18		0x0B Battery	560,00 µs		Average Time To Empty	2570 min (0x0A0A)	0x12 (0.0..1.7)
19		0x0B Battery	560,00 µs		Average Time To Full	Not Charged min (0xFFFF)	0x13 (0.0..1.7)
20		0x0B Battery	560,00 µs		Charging Current	240 mA (0x00F0)	0x14 (0.0..1.7)
21		0x0B Battery	560,00 µs		Charging Voltage	Charger Should Be Current Source mV (0xFFFF)	0x15 (0.0..1.7)
22		0x0B Battery	560,00 µs		Over Charged Alarm	Charging Is No Longer Detected (0x00)	0x16 (1.7)
					Terminate Charge Alarm	Charging Is No Longer Detected (0x00)	0x16 (1.6)
					Over Temp Alarm	Temperature Drops Into Acceptable Range (0x00)	0x16 (1.4)
					Terminate Discharge Alarm	Discharge Is No Longer Detected (0x00)	0x16 (1.3)
					Remaining Capacity Alarm	"Remaining Capacity" > "Remaining Capacity Alarm" or "Remaining Capacity Alarm" == 0 (0x00)	0x16 (1.1)
					Remaining Time Alarm	"Average Time To Empty" > "Remaining Time Alarm" Or "Remaining Time Alarm" == 0 (0x00)	0x16 (1.0)
					Initialized	Calibration Or Configuration Information Has Been Lost And Accurracy Is Significantly Impaired (0x00)	0x16 (0.7)
					Discharging	Battery Is Discharging (0x01)	0x16 (0.6)
					Fully Charged	Battery Is No Longer Considered In A Full State (0x00)	0x16 (0.5)
					Fully Discharged	"Relative State Of Charge" > 20% (0x00)	0x16 (0.4)
					Error Codes	OK (0x00)	0x16 (0.0..0.3)
23		0x0B Battery	560,00 µs		Cycle Count	3048 (0x0BE8)	0x17 (0.0..1.7)

Bild 2.4.3.c: Telos I2C Studio Register/Value View [telos]

Die normale Aufzeichnungsfunktion kann quasi unbegrenzt Busdaten aufzeichnen, kann aber die physikalische Protokollebene nicht auswerten bzw. anzeigen. Das I2C Studio bietet mit der Analog Option eine Erweiterung an, mit der die physikalische Protokollebene, ähnlich wie mit einem Oszilloskop, als Waveform angezeigt werden kann (Bild 2.4.3.d). Die Daten werden durch einen AD Wandler in der externen Hardware aufgezeichnet und dort während der Aufzeichnung gespeichert. Nach der Messung werden die Daten an den Host übertragen. Die Aufzeichnungsdauer ist dadurch begrenzt. Die Messung wird wie bei einem Oszilloskop, durch einen Trigger gestartet. Die beiden Aufzeichnungsfunktionen können kombiniert werden. Die normale Aufzeichnungsfunktion kann einen längeren Abschnitt an Busdaten erfassen, während die Analogoption bestimmte Abschnitte, die durch Trigger bestimmt werden, erfassen kann. Beide Aufzeichnungen können dann mit der Hostsoftware mit der gleichen Zeitachse betrachtet werden.

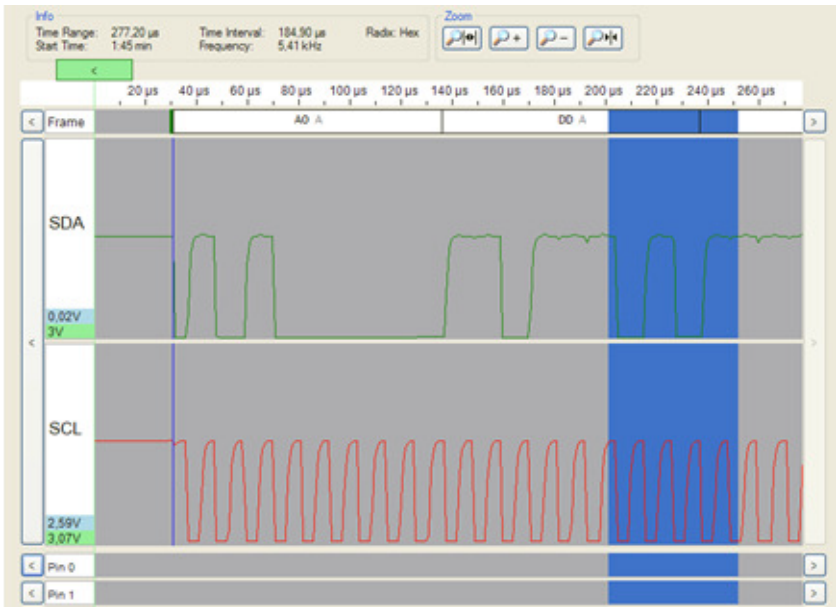

Bild 2.4.3.d: Telos Tracie XL 2.0 Analog Option [telos]

2.4.4 Zusammenfassung und Bewertung

Überblick Oszilloskope:

- Umfangreiche grafische Analysemöglichkeiten für physikalische Protokollebenen
- Nicht beschränkt auf ein bestimmtes Bussystem bzw. Protokoll
- Beschränke Aufzeichnungsdauer < 1s
- Trigger orientierte Arbeitsweise
- Einige Modelle bieten Dekodierungsmöglichkeiten für bestimmte serielle Protokolle
- Datenerfassung über AD Wandler

Überblick Logic Analyzer:

- Grafische Analyse der binären Zustände der Busleitungen
- Nicht beschränkt auf ein bestimmtes Bussystem bzw. Protokoll
- Beschränke Aufzeichnungsdauer, Verlängerung durch Kompression
- Trigger orientierte Arbeitsweise
- Einige Modelle bieten Dekodierungsmöglichkeiten für bestimmte serielle Protokolle
- Binäre Erfassung Busleitungszustände

Überblick Protokoll Analyzer:

- Fokus auf ein bestimmtes Protokoll
- Auswertung der physikalischen Ebene i.d.R. nicht möglich
- Fast unbeschränkte Aufzeichnungsdauer
- Ausführungen mit spezialisierten Funktionen für die Auswertung von Anwender-protokollebenen
- Ausführungen mit spezialisierten Erweiterungen zur Auswertung der physikalischen Protokollebene

Die gezeigten Standardlösungen können nur bestimmte Protokollebenen analysieren. Oszilloskope und Logic Analyzer spezialisieren sich auf die physikalische Protokollebene. Protokoll Analyzer spezialisieren sich auf die höheren Protokollebenen. Wie im vorherigen Abschnitt gezeigt wurde, entstehen Nachteile durch die Trennung dieser Funktionalitäten. Durch eine Kombination der Fähigkeiten können leistungsfähigere Analysewerkzeuge geschaffen werden. Die Telos Lösung z.B. erweitert die Funktionalität eines Protokoll Analyzers durch die Möglichkeit, die physikalische Protokollebene zu analysieren. Zudem können die höheren Anwenderprotokollebenen zum Teil dekodiert werden. Durch diese erweiterten Fähigkeiten, die Protokollebenen zu dekodieren und in Bezug zueinander zu Analysieren, kann der Analysevorgang deutlich beschleunigt werden. Die Telos Lösung ist jedoch nur für das I2C Protokoll ausgelegt, und die Dekodierung der Anwenderprotokolle ist auf I2C ICs spezialisiert. Es fehlen Möglichkeiten diese Konzepte und entsprechende Vorteile auch für andere Protokolle zu nutzen.

2.5 Angrenzende Themengebiete

2.5.1 Wireshark

Wireshark [wireshark], ist ein Packet Analyzer für Netzwerke. Wireshark ermöglicht es u. a., die durch ein Netzwerk übertragenen Pakete zu dekodieren und anzuzeigen [wireshark_user]. Das Bild 2.5.1.a, zeigt die Wireshark Oberfläche bei der Dekodierung von Ethernet Paketen.

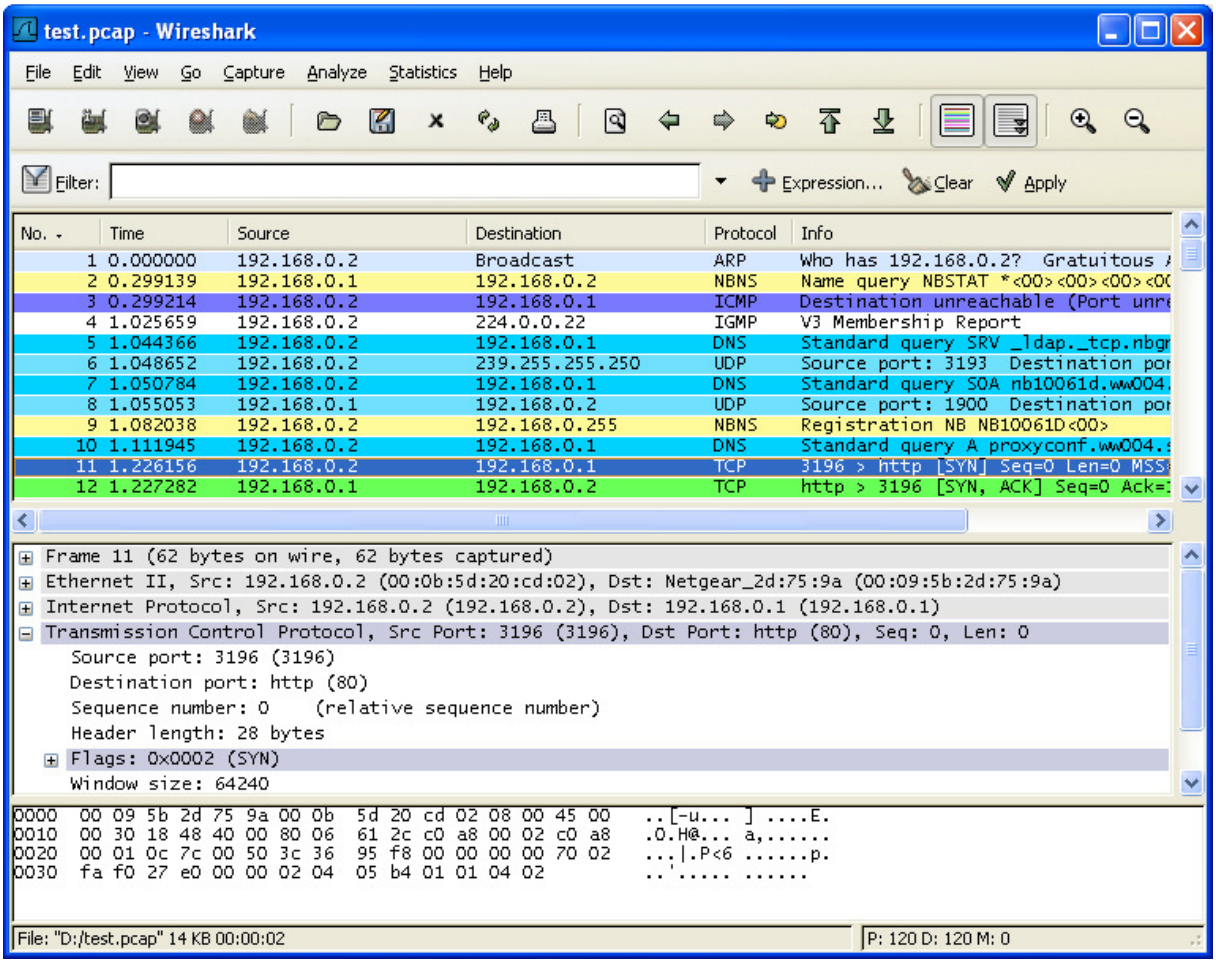

Bild 2.5.1.a: Wireshark Network Protocol Analyzer [wireshark_user]

Es lassen sich einige der Konzepte von Wireshark auch auf serielle Busprotokolle übertragen, da die Netzwerkpakete ähnlich aufgebaut sein können wie serielle Busprotokolle. Die Netzwerkprotokollebene Ethernet z.B., ist ein Bus und transportiert einen seriellen Datenstrom. (vgl. [tanenbaum_cn], S. 271f)

Es werden hier insbesondere folgende Konzepte von Wireshark untersucht:

- Die Implementierung der Dekodierungsfunktionen für Netzwerkpakete
- Die Filter und Suchfunktionen
- Die Import und Capture Funktionen für Netzwerkdaten

Wireshark kann zahlreiche Protokolle bzw. Protokollebenen dekodieren. Es können durch Code Module oder Plugins (DLL) neue Protokollebeben implementiert werden (Wireshark ist Open Source). Das Dekodieren der Protokollebenen erfolgt hierarchisch, wobei jedes Modul eine Protokollebene dekodiert. [wireshark_dev]

Die Filter von Wireshark bieten die Möglichkeit bestimmte Protokolldaten zu filtern. Die Eingabe der Filter erfolgt als String, entsprechend einer Filtersyntax. Bei der Eingabe der Filter können Tokens verwendet werden, um einen Protokolldatenbereich anzugeben. So kann z.B. mit dem Token „ip.addr“ in einem entsprechenden Filterstring „ip.addr==192.168.0.1“ ein Filter für IP Pakete mit der Adresse 192.168.0.1 erstellt werden. [wireshark_user]

Wireshark kann Netzwerkdaten aus einer zuvor erstellten Datei einlesen oder die Netzwerkdaten durch eine Capture Funktion in Echtzeit aufzeichnen. Die Capture Funktion von Wireshark, verwendet die PCAP Bibliothek [pcap], die eine Schnittstelle zur Netzwerkhardware bereitstellt.

2.5.2 Logic Pro

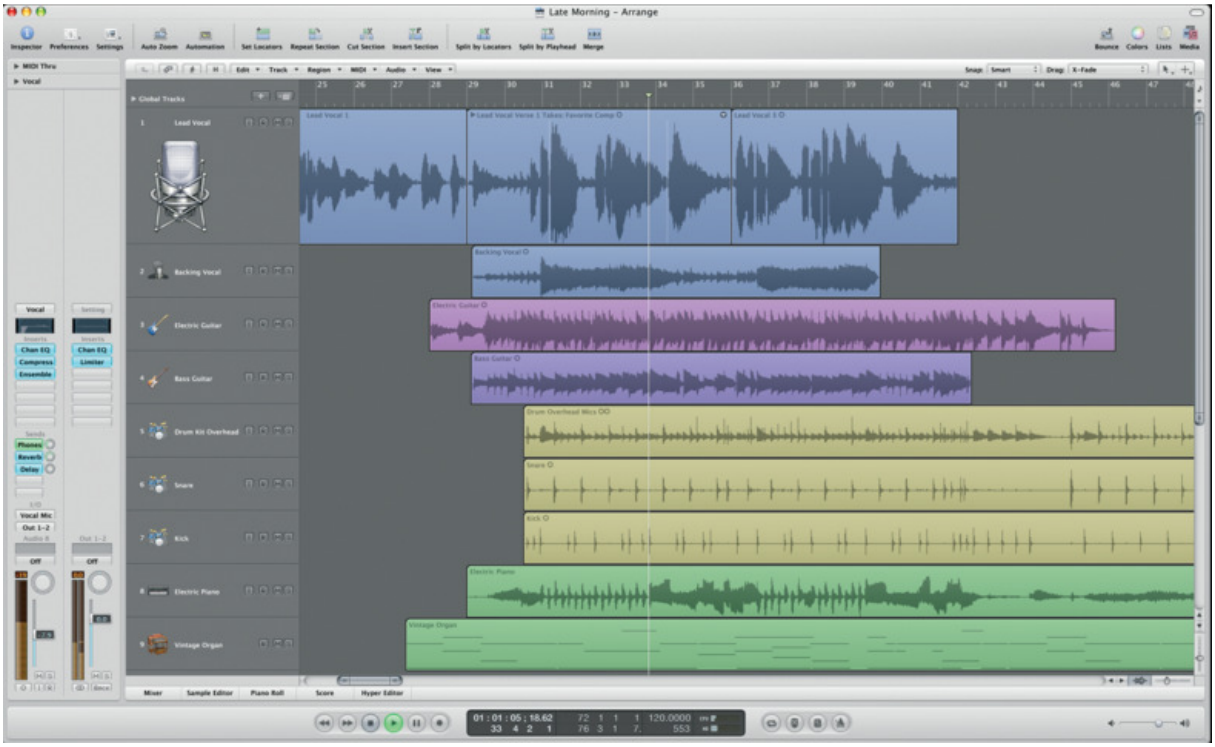

Bild2.5.2.a: Logic Pro 9 [logicpro]

Logic Pro ist eine Digital Audio Workstation und Midi Sequenzer, mit der Musik und Tonaufnahmen produziert werden (Bild 2.5.2.a) [logicpro].

Die Audio und MIDI Daten, die durch Logic Pro aufgezeichnet und angezeigt werden, ähneln den Daten, die bei der Busanalyse verarbeitet werden. Die Audiodaten werden durch A/D Wandler aufgezeichnet, ähnlich wie bei einem digitalen Speicheroszilloskop. Die MIDI Daten sind Steuerinformationen, wie z.B. wann eine Note auf einem Keyboard gespielt wurde, ähnlich einem

Ereignis auf einem Bussystem, wie z.B. ein Flankenwechsel auf einer Busleitung. Die Audio und MIDI Daten haben eine gemeinsame Zeitachse, wie auch die Daten der unterschiedlichen Protokollebenen eines Bussystems.

Die Darstellung der Audio und MIDI Daten erfolgt auf folgende weise, bzw. hat folgende Funktionen:

- Die Anordnung der Audio und MIDI Kanäle erfolgt untereinander auf einer gemeinsamen Zeitachse, wie in Bild 2.5.2.a.
- Durch horizontales Scrollen kann ein beliebiger Zeitabschnitt angezeigt werden.
- Durch Ein- und Auszoomen auf der Zeitasche kann ein großer Zeitabschnitt mit groben Details oder ein kleiner Zeitabschnitt sehr detailiert angezeigt werden.
- Durch anklicken der Objekte können weitere Details oder andere Ansichten wie z.B. Listen angezeigt werden.

Logic pro verwendet u. a. eine Treiberschnittstelle (ASIO [asio]), um externe A/D Wandler Hardware anzubinden, mit der die Audiodaten aufgezeichnet werden. Durch diese Schnittstelle können die Hersteller der Hardware unterschiedliche Konzepte unabhängig von der Hostsoftware implementieren. So können die A/D Wandler z.B. über PCI Karten, USB, Firewire oder proprietäre Lösungen angebunden werden.

3. Anforderungen

3.1 Erläuterungen zu Begriffen und Abkürzungen

Messdaten - Im Folgenden wird der Begriff Messdaten verwendet, um die Daten zu beschreiben, die durch Beobachten bzw. Messen der Datenübertragung auf einem Bussystem gewonnen werden.

3.2 Allgemeine Beschreibung

3.2.1 Produktfunktionen

- Die Messdaten bzw. Protokolldaten sollen durch eine Schnittstelle importiert werden können.
- Die Protokolldaten sollen auf unterschiedlichen Protokollebenen auf einer gemeinsamen Zeitachse grafisch ausgegeben und navigiert werden können.
- Welche Protokolldaten ausgegeben werden, soll durch Filterfunktionen eingeschränkt werden können.
- Die Protokolldaten sollen nach Datenmustern oder Ereignissen durchsucht werden können.
- Die Triggerfunktionen sollen Protokolldatenabhängiges Triggern ermöglichen.
- Die Strukturen und Daten der Protokollnutzdaten sollen bei der Ausgabe durch entsprechende Formatierungs- und Bezeichnungsmöglichkeiten kenntlich gemacht werden können.

3.2.2 Benutzermerkmale

Es wird davon ausgegangen, dass die Benutzer über eine technische Ausbildung verfügen und mit dem Aufbau und der Arbeitsweise von Bussystemen und den zu analysierenden Protokollen vertraut sind.

3.3 Spezifische Anforderungen

3.3.1 Funktionale und nichtfunktionale Anforderungen

A. Die Messdaten bzw. Protokolldaten sollen durch eine Schnittstelle Importiert werden können. (Die Messdaten werden in Abschnitt 3.3.2 ausführlicher beschrieben.)

1. Es sollen Messdaten von verschiedenen Datenquellen verarbeitet werden können. Die Datenquellen können dabei u. a. dateibasiert, oder bei Hardwareanbindung, API-basiert sein. Die Datenquellen sind zum Entwurfszeitpunkt noch nicht bekannt.
2. Die Datenquellen liefern Daten in sample- oder ereignisbasierter Form. Samplebasierte Daten sollen mit einer Auflösung von 1-12 Bit und einer Abtastrate von bis zu 500 MHz importiert und in gleicher Qualität von der Anwendung weiterverarbeitet werden können.
3. Die Ereigniszeitpunkte von ereignisbasierten Daten sollen mit einer Mindestgenauigkeit von ± 1 ns durch die Anwendung weiterverarbeitet werden können.
4. Die Messdaten können einen Zeitraum von bis zu 5 Tagen widerspiegeln.
5. Die Anwendung soll API basierte Datenquellen steuern können.

B. Die Protokolldaten sollen auf unterschiedlichen Protokollebenen auf einer gemeinsamen Zeitachse grafisch ausgegeben und navigiert werden können.

1. Die Darstellung der Ereignisse soll linear zum angezeigten Zeitabschnitt erfolgen.
2. Es sollen Protokollstrukturen und Protokolldaten validiert werden können.
3. Fehlerhafte Protokolldaten sollen ausgewiesen werden können.
4. Das Layout soll sich zur Laufzeit an den Darstellungsbereich anpassen können.

C. Welche Protokolldaten ausgegeben werden, soll durch Filterfunktionen eingeschränkt werden können.

1. Die Filterfunktionen sollen die Protokolldaten mit Vergleichsoperationen filtern können.
2. Die Filterkonfigurationen sollen im Dateiformat abgespeichert und geladen werden können.

D. Die Protokolldaten sollen nach Datenmustern oder Ereignissen durchsucht werden können.

1. Die Protokolldaten sollen mit Vergleichsoperationen durchsucht werden können.
2. Die Suchkonfigurationen sollen im Dateiformat abgespeichert und geladen werden können.

E. Die Triggerfunktionen sollen Protokolldatenabhängiges Triggern ermöglichen.

1. Die Triggerbedingungen sollen durch den Einsatz von Vergleichsoperationen im Zusammenhang mit den Protokolldaten beschrieben werden können.
2. Die Triggerkonfigurationen sollen im Dateiformat abgespeichert und geladen werden können.

3. Die Triggerfunktionen sollen genutzt werden können, um externe Geräte anzusteuern.

F. Die Strukturen und Daten der Protokollnutzdaten sollen bei der Ausgabe durch entsprechende Formatierungs- und Bezeichnungsmöglichkeiten kenntlich gemacht werden können.

1. Die Angaben über die Protokollnutzdaten sollen durch den Anwender leicht angepasst und verwaltet werden können.

Zusätzliche Nichtfunktionale Anforderungen:

- Die Implementierung neuer Protokolle bzw. Protokollebenen soll möglichst einfach und übersichtlich erfolgen.
- Die Anwendung soll performant arbeiten. Das Design soll möglichst schnelle Reaktionszeiten der Ausgabe-, Filter- und Suchfunktionen ermöglichen.

3.3.2 Eingangsdaten

Die Anwendung soll es ermöglichen bei der Messdatenerfassung mit vorhandener oder speziell entwickelter Hardware zusammenzuarbeiten. Dazu sollen samplebasierte oder ereignisbasierte Messdatenformate verarbeitet werden können (Abschnitt 2.3.). So können die Messdatenformate in Abhängigkeit der Fähigkeiten von der Datenerfassungshardware gewählt werden. So kann z.B. eine über USB extern angebundene Hardware nur ereignisbasierte Daten mit verhältnismäßig niedriger Datenrate übergeben oder eine interne PCIe Steckkarte samplebasierte Daten mit einer entsprechend höheren Datenrate übergeben. Die möglichen Kombinationen pro Busleitung, werden in Bild 3.3.2.a gezeigt. Eingangstyp A besteht aus ereignisbasierten Daten für den kompletten Messvorgang. Eingangstyp B besteht aus samplebasierten Daten für den kompletten Messvorgang. Eingangstype C besteht aus ereignisbasierten Daten für den kompletten Messvorgang, ergänzt um beliebig viele Abschnitte von samplebasierten Daten. Der Eingangstype C wird z.B. eingesetzt, wenn die samplebasierten Daten abschnittsweise durch Trigger aufgezeichnet werden. Dadurch sollen Sampledatenabschnitte mit hohen Abtastraten und entsprechend hohen Datenraten verarbeitet werden können, die aufgrund der hohen Datenraten nicht für den kompletten Messvorgang vorliegen.

Die samplebasierte Daten sollen mit einer Auflösung von 1-12 Bit und einer Abtastrate von bis zu 500 MHz importiert und in gleicher Qualität von der Anwendung weiterverarbeitet werden können.

- Eine Auflösung der Spannungspegel der Busleitungen von 1-12 Bit wird vorgegeben, da 1 Bit ausreicht, um die booleschen Zustände der Busleitungen zu erfassen und 12 Bit eine hohe Genauigkeit bei der grafischen Ausgabe durch die 4096 Spannungspegelschritte ermöglicht.
- Eine Abtastrate von bis zu 500 MHz wird vorgegeben, da Sampledaten mit hoher Genauigkeit nötig sein können um bestimmte Probleme wie z.B. Spannungspeaks zu erkennen (Abschnitt 2.3.2).

Die Ereigniszeitpunkte von ereignisbasierten Daten sollen mit einer Mindestgenauigkeit von ± 1 ns durch die Anwendung weiterverarbeitet werden können.

- Wie bei den samplebasierten Daten soll eine hohe Genauigkeit es ermöglichen Vorgänge der Busleitungen wie Spannungspeaks zu erfassen.

Die Messdaten können einen Zeitraum von bis zu 5 Tagen widerspiegeln.

- Bei einem Bussystem wie IPMB, das u. a. Steuerungs- und Regelfunktionen übernimmt, ist es u. U. nötig das Verhalten vom Bussystem und der Teilnehmer über einen längeren Zeitraum zusammenhängend zu erfassen und nachvollziehen zu können.

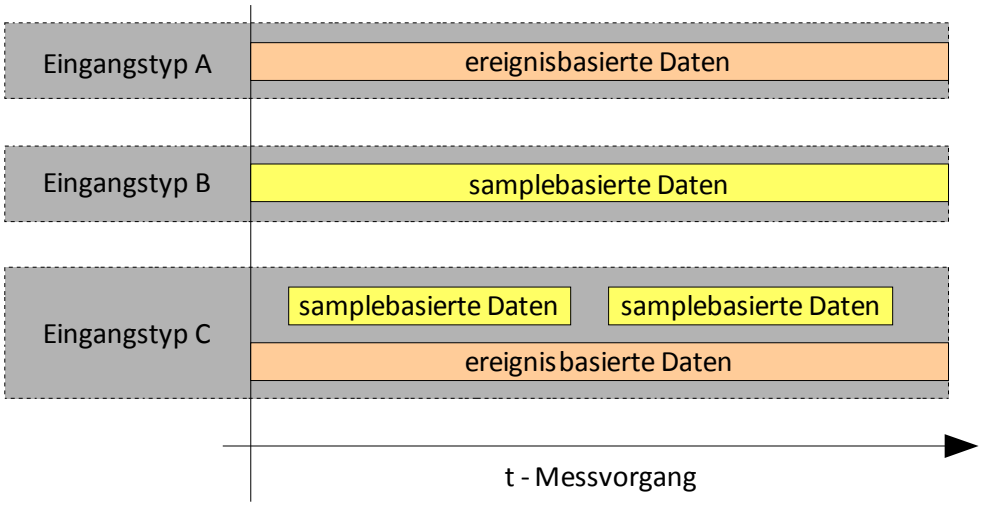

Bild 3.3.2.a: Eingangstypen Messdaten

4. Konzept und Realisierung

Das Kapitel Konzept und Realisierung ist folgendermaßen aufgebaut:

- Konzeptübersicht (Abschnitt 4.1)
- Konzept und Implementierung (Abschnitt 4.2-4.7)
- Kontrolle der Anforderungserfüllung (Abschnitt 4.8)
- Test der Implementierung (Abschnitt Teil 4.9)

4.1 Konzeptübersicht

Das Konzept wird in Module aufgeteilt. Die Aufteilung richtet sich nach den Funktionsbereichen. Das Bild 4.1.a zeigt die einzelnen Module die im Folgenden beschrieben werden.

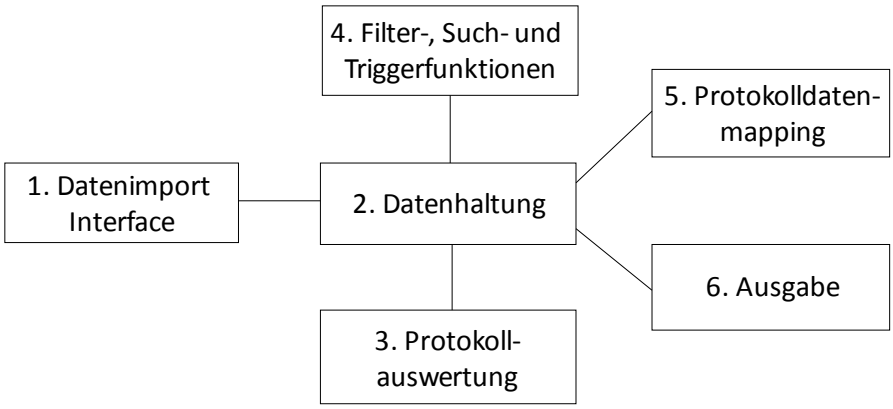

Bild 4.1.a: Anwendungsmodule Konzeptübersicht

1. **Datenimport Interface -** Das Datenimport Interface importiert die zu analysierenden Protokolldaten. Das Datenimport Interface wird in Abschnitt 4.2 beschrieben.

2. **Datenhaltung -** Das Datenmodell verwaltet die Protokolldaten und ermöglicht es anderen Anwendungsteilen die Protokolldaten zu verarbeiten. Das Datenmodell wird in Abschnitt 4.3 beschrieben.

3. **Protokollauswertung -** Bei der Protokollauswertung wird u. a. das Protokoll und die entsprechenden Protokollebenen der Eingangsdaten erkannt und dekodiert. Die Protokollauswertung wird in Abschnitt 4.4 beschrieben.

4. **Filter-, Such- und Triggerfunktionen -** Die Filterfunktionen ermöglichen das Filtern der Protokolldaten nach bestimmten Datenmustern oder Protokollereignissen. Die Filter-, Such- und Triggerfunktionen werden in Abschnitt 4.5 beschrieben.

5. **Protokolldatenmapping -** Das Mapping der Protokollnutzdaten dient dazu, die höchsten Protokollebenen, die Ebenen die projektbezogen festgelegt werden, mit Format- und Strukturinformation ausgeben zu können. Das Protokolldatenmapping wird in Abschnitt 4.6 beschrieben.

6. **Ausgabe der Protokolldaten -** Die grafische Ausgabe der Protokolldaten ermöglicht u. a. die Ausgabe der Protokollebenen auf einer Zeitachse. Die grafische Ausgabe wird in Abschnitt 4.7 beschrieben.

4.2 Datenimport Interface für externe Messdaten

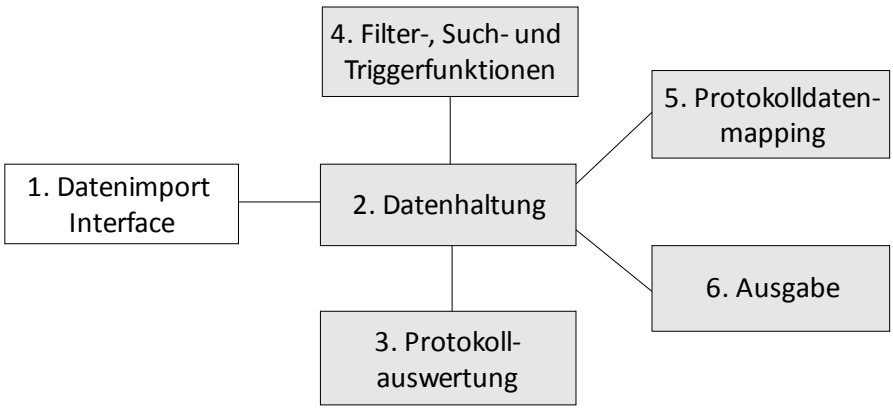

Bild 4.2.a: Einordnung Datenimport Interface

In diesem Abschnitt wird das Interface beschrieben, das für den Import von externen Messdaten und der damit verbundenen Steuerung zuständig ist.

Folgende Punkte werden besprochen:

- Übersicht Anforderungen
- Gerätetreiberarchitektur
- Konvertierung der Eingangsdaten durch Gerätetreiber
- Messdatenschnittstelle zwischen Gerätetreiber und Anwendung
- Datenkompression für samplebasierte Daten
- Steuerungsprotokoll zwischen Anwendung und Gerätetreiber
- Triggerfunktionen für externe Hardware

4.2.1 Übersicht Anforderungen

Aus den Anforderungen aus Teil 3, ergeben sich eine Reihe von allgemeinen Kriterien für das Datenimport Interface.

- Die Messdaten bzw. Protokolldaten sollen durch eine Schnittstelle Importiert werden können.
- Es sollen Messdaten von verschiedenen Datenquellen verarbeitet werden können. Die Datenquellen können dabei u. a. Dateibasiert oder bei Hardwareanbindung, API-basiert sein. Die Datenquellen sind zum Entwurfszeitpunkt noch nicht bekannt.
- Die Datenquellen liefern Daten in sample- oder ereignisbasierter Form.
- Die Anwendung soll API basierte Datenquellen steuern können.
- Die Triggerfunktionen sollen genutzt werden können um externe Geräte anzusteuern.

4.2.2 Gerätetreiberarchitektur

Um den Einsatz verschiedener Datenquellen zu ermöglichen, werden diese mit einem Gerätetreiber an die Anwendung angebunden. Ähnlich wie bei einem Betriebssystemtreiber, werden die gerätespezifischen Messdaten für die Übergabe an die anwendungsinterne Messdatenschnittstelle aufbereitet. Es wird ein spezieller Gerätetreiber für jede Datenquelle erstellt wie Bild 4.2.2.a zeigt.

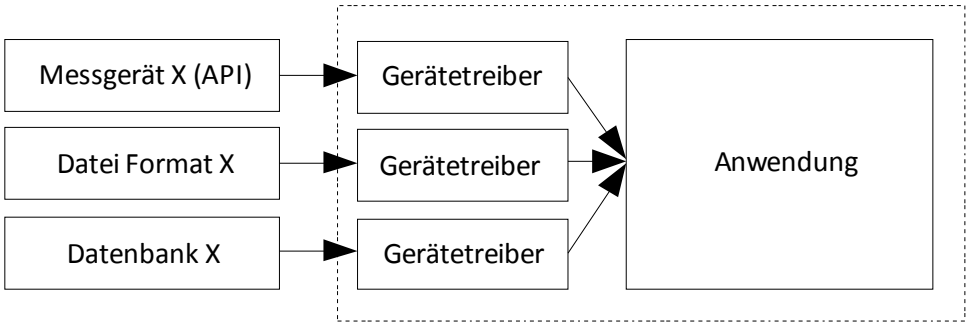

Bild 4.2.2.a: Gerätetreiber für Messdatenquellen

Die Vorteile der Gerätetreiberarchitektur

- Erleichtert die Anbindung von Datenquellen durch vorgegebene Treiberarchitektur
- Ermöglicht normierte Datenschnittstelle zwischen Gerätetreiber und Anwendung
- Ermöglicht es, aktive Datenquellen, wie z.B. API basierte Messsysteme und Datenbanken oder passive Datenquellen wie z.B. Dateien mit Messdaten zu unterstützen.
- Ermöglicht das quellenspezifische Konvertieren und Aufbereiten der Messdaten.
- Ermöglicht es, aktive Datenquellen zu steuern.

4.2.3 Konvertierung der Eingangsdaten durch Gerätetreiber

In Teil 4.3.2 wird festgelegt, dass die Anwendung nur ereignisbasierte Daten verwendet für die weitere Auswertung, wie u. a. Protokolldekodierung und Filterfunktionen. Die samplebasierten Daten werden nur zur grafischen Ausgabe der physikalischen Protokollebene verwendet und nicht weiter ausgewertet. Der Gerätetreiber muss daher in bestimmten Fällen die samplebasierten Daten in ereignisbasierte Daten konvertieren, wenn nur samplebasierte Daten vorliegen. In Teil 3.2.2 der Anforderungen werden die möglichen Eingangstypen der Messdaten festgelegt. Der Gerätetreiber konvertiert diese Eingangstypen in die entsprechenden Übergabetypen wie in Bild 4.2.3.a. Bild 4.2.3.b zeigt die Formate der Eingangstypen und die entsprechenden Formate der Übergabetypen. Beim Eingangstyp B muss der Gerätetreiber die samplebasierten Daten in ereignisbasierte Daten konvertieren, da bei diesem Eingangstyp nur samplebasierte Daten vorhanden sind, die Anwendung aber ereignisbasierte Daten für die weitere Verarbeitung benötigt.

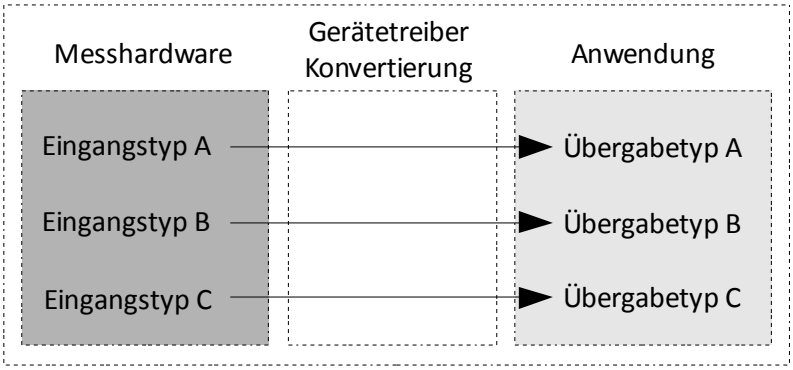

Bild 4.2.3.a: Eingangstypen und Übergabetypen der Messdaten pro Busleitung

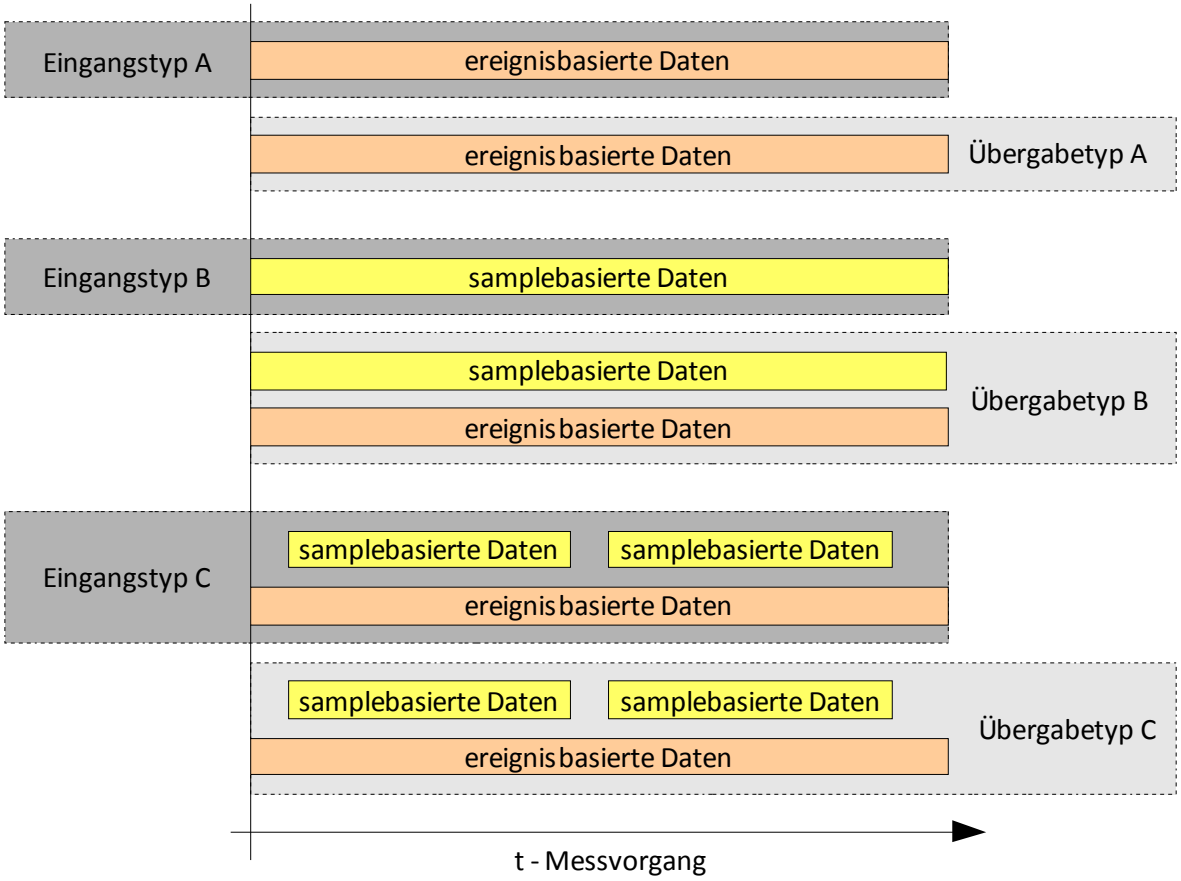

Bild 4.2.3.b: Konvertierung Messdaten Eingangstypen in Übergabetypen pro Busleitung

4.2.4 Messdatenschnittstelle zwischen Gerätetreiber und Anwendung

Die Messdaten werden der Anwendung durch den Gerätetreiber übergeben. Die ereignisbasierten und samplebasierten Daten werden unabhängig voneinander durch Schnittstellen übergeben.

Das Bild 4.2.4.a zeigt die Schnittstelle für die ereignisbasierten Daten. Ein Ereignis besteht aus einem *Timestamp*, einem *Channel* und einem *Type:*

- *Timestamp* gibt den Zeitpunkt an, an dem das Ereignis aufgetreten ist, relativ zum Startzeitpunkt der Messung. In Teil 4.3.6 wird beschrieben, wie ein Zeitpunkt durch den Datentyp Timestamp ausgedrückt wird.
- *Channel* gibt die Busleitung an, auf dem das Ereignis aufgetreten ist.
- *Type* gibt den Ereignistyp wieder. Der Ereignistyp beschreibt den neuen Zustand der Busleitung, hier HIGH, LOW, oder Unknown.

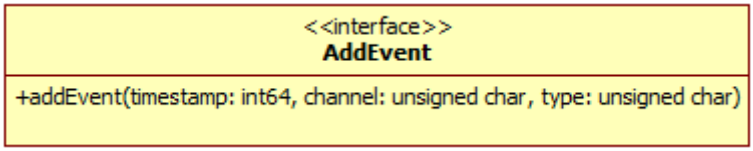

Bild 4.2.4.a: Schnittstelle für eventbasierte Daten

Bild 4.2.4.b, zeigt die Schnittstelle für die samplebasierten Daten. Die Sampledaten werden abschnittsweise, arraybasiert übergeben. Der Anfangs- und Endzeitpunkt der übergebenen Samples wird mit übergeben. Ein Abschnitt von Sampledaten wird mit den folgenden Parametern übergeben:

- *Starttime* gibt den Zeitpunkt vom ersten übergebenen Sample im übergebenen Samplearray an, relativ zum Startzeitpunkt der Messung.
- *Stoptime* gibt den Zeitpunkt vom letzten übergebenen Sample im übergebenen Samplearray an, relativ zum Startzeitpunkt der Messung.
- *Channel* gibt die Busleitung der übergebenen Sampledaten an.
- *Data* ist ein Array der zu übergebenden Sampledaten.
- *Datalength* ist die Anzahl der übergebenen Samples im übergeben Array.

Bild 4.2.4.b: Schnittstelle für samplebasierte Daten

4.2.5 Datenkompression für samplebasierte Daten

Die samplebasierten Daten werden durch das konstante Abtasten der Busleitungen gewonnen (Abschnitt 2.3.1). Es werden Daten aufgezeichnet unabhängig von der Busaktivität. Das Bussystem ist jedoch i.d.R. nicht vollständig ausgelastet. Bei einem IPMB Bus z.B. ist es nicht festgelegt, wann die Teilnehmer senden. Jeder Teilnehmer versucht nach Bedarf das Senden einzuleiten. Ist der Bus belegt, kann der Sendevorgang später erneut versucht werden. Diese Arbeitsweise setzt voraus, dass der Bus über genug Leerlauf verfügt, um Kollisionen beim Sendeversuch zu minimieren. Bild 4.2.5.a, zeigt den Leerlauf zwischen den Übertragungen auf einem Bus.

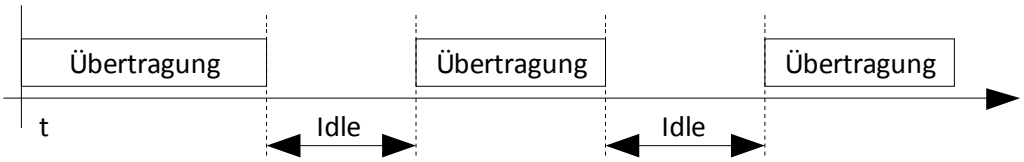

Bild 4.2.5.a: Idle Zeitabschnitte Bussystem

Dieses Verhalten kann dazu genutzt werden, die von der Anwendung zu speichernden, samplebasierten Daten zu reduzieren. Dazu werden die samplebasierten Daten für die Zeitabschnitte bei denen der Bus Idle ist, nicht gespeichert. Wie in Teil 4.2.3 beschrieben, liegen für die samplebasierten Daten parallel eventbasierte Daten vor. Die samplebasierten Daten werden nur zur Anzeige der physikalischen Protokollebene verwendet und für die weitere Auswertung nicht benötigt, daher können die Ausschnitte, bei denen der Bus Idle ist, weggelassen werden. Der Gerätetreiber kann die Idle Abschnitte mit einem entsprechenden Algorithmus erkennen. Die Übergabeschnittstelle für samplebasierte Daten zwischen Gerätetreiber und Anwendung (Teil 4.2.4) ermöglicht es dem Gerätetreiber, beliebige Ausschnitte an samplebasierten Daten zu übergeben bzw. wegzulassen.

4.2.6 Steuerungsprotokoll zwischen Anwendung und Gerätetreiber

Die aktiven Datenquellen, wie z.B. eine PCIe-Karten basierte Messhardware, die durch eine API gesteuert werden kann, sollen durch die Anwendung über den Gerätetreiber gesteuert werden können. Die Messsysteme bieten unterschiedliche Steuerungs- und Einstellungsmöglichkeiten. Um ein einfaches Steuerungsprotokoll zwischen Gerätetreiber und Anwendung zu ermöglichen, werden nur Grundfunktionen vorgegeben. Die gerätespezifische Steuerungs- und Konfigurationsfunktionen, werden durch den Gerätetreiber implementiert. Der Gerätetreiber kann einen geräteabhängigen Konfigurationsdialog anzeigen, damit auf die geräteabhängigen Steuerungs- und Konfigurationsfunktionen zugegriffen werden kann. Diese Arbeitsweise wird auch bei der ASIO Schnittstelle für Soundkarten verwendet, die im Analyseteil untersucht wurde (Abschnitt 2.5.2). Das Sequenzdiagramm in Bild 4.2.6.a, zeigt das Steuerungsprotokoll zwischen Gerätetreiber und Anwendung.

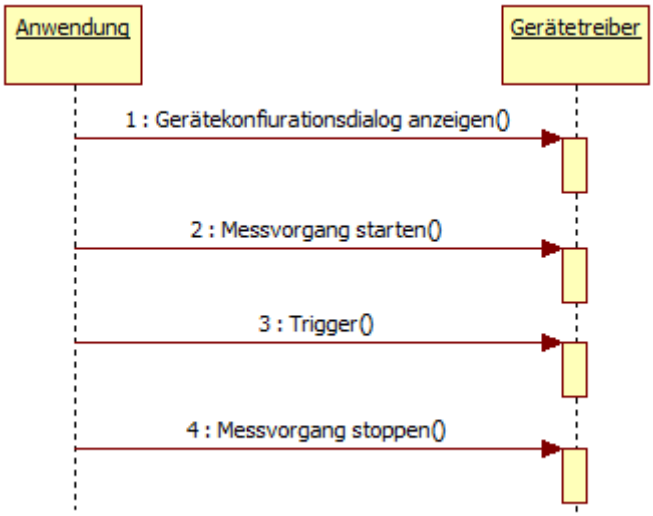

Bild 4.2.6.a: Steuerungsprotokoll zwischen Gerätetreiber und Anwendung

Nachrichten zwischen Anwendung und Gerätetreiber:

- Messvorgang starten
  - Den Messvorgang starten, die Übertragung der Messdaten starten
- Messvorgang stoppen
  - Den Messvorgang stoppen, die Übertragung der Messdaten stoppen
- Trigger (Channel 1-8)
  - Ein zuvor festgelegtes Triggerereignis ist eingetreten. Die Trigger können auf 8 unterschiedlichen Kanälen ausgelöst werden. (Die Implementierung der Trigger wird in Abschnitt 4.2.7 und 4.5.6 beschrieben)
- Gerätekonfigurationsdialog anzeigen
  - Der Gerätetreiber zeigt einen gerätespezifischen Konfigurationsdialog an

4.2.7 Triggerfunktionen für externe Hardware

Die Triggerfunktionen werden eingesetzt, um Vorgänge zu starten, sobald ein bestimmtes Protokollereignis eintritt. Die Messhardware kann z.B. durch ein Triggerereignis veranlasst werden, einen Abschnitt an samplebasierten Messdaten aufzuzeichnen. Die Triggerfunktionen können auch genutzt werden, um externe Messgeräte in den Messvorgang einzubinden. Die externen Messgeräte starten den Messvorgang sobald das Triggerereignis ausgelöst wird. Messgeräte wie Oszilloskope haben i.d.R. Triggereingänge die über I/Os angesteuert werden können.

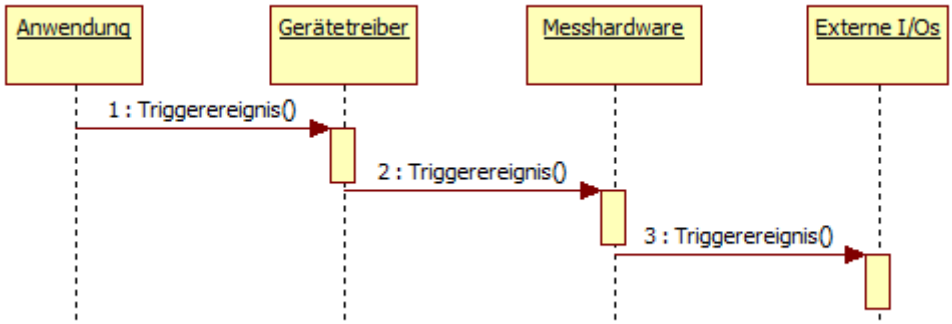

Bild 4.2.7.a: Triggerereignisse zwischen Anwendung und Gerätetreiber

Das in Abschnitt 4.2.6 beschriebene Steuerungsprotokoll zwischen Anwendung und Gerätetreiber bietet die Möglichkeit Triggerereignisse auf acht Verschiedenen Kanälen auszulösen. Wie die Triggerereignisse durch die Anwendung konfiguriert werden, wird in Teil 4.5.6 beschrieben.

Wird ein Trigger ausgelöst, wird eine Nachricht von der Anwendung an den Gerätetreiber gesendet, der Gerätetreiber kann diese Nachricht an die Messhardware weiterleiten, die diese intern verwendet oder auf I/Os legen kann. Durch die I/Os können weitere externe Geräte gesteuert werden. Das Bild 4.2.7.a zeigt ein Triggerereignis, das von der Anwendung an den Gerätetreiber gesendet wird, vom Gerätetreiber zur Messhardware und von der Messhardware zu den externen I/Os weitergeleitet wird.

4.3 Datenmodell

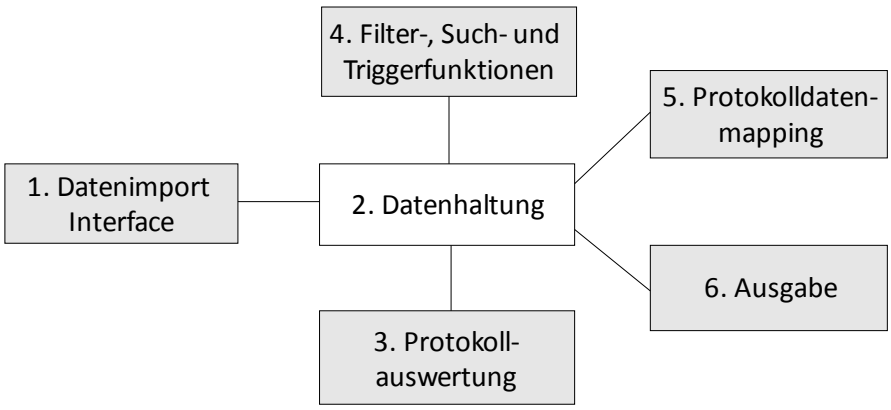

Bild 4.3.a: Einordnung Datenhaltung

In diesem Kapitel wird das verwendete Datenmodell vorgestellt. Die Hauptaufgabe des Datenmodells besteht darin, die externen Messdaten und deren Auswertung zu verwalten, damit diese Daten von den Anwendungsteilen weiterverarbeitet werden können.

Folgende Punkte werden besprochen:

- Übersicht Anforderungen
- Ereignisbasierte Verarbeitung von Protokolldaten
- Getrennte Module für Protokollebenen
- Datenstrukturen für eventbasierte Protokolldaten
- Bufferkonzept für große Datenmengen
- Datentypen für Ereigniszeitpunkte

4.3.1 Übersicht Anforderungen

Aus den Anforderungen an die Anwendung (Abschnitt 3) ergeben sich eine Reihe von allgemeinen Kriterien die beim Entwurf vom Datenmodell berücksichtigt werden müssen:

- Die Anwendungsteile müssen mit dem Datenmodell die geforderte Funktionalität umsetzen können, u. a. interne Protokollauswertung, Filterfunktionen und Ausgabefunktionen.
- Die externen Messdaten müssen mit geforderter Genauigkeit verwaltet und weiterverarbeitet werden können.
- Die Anwendung soll für zukünftige Protokolle entsprechend angepasst werden können. Das Datenmodell muss diese Anpassungen ermöglichen und begünstigen.
- Die Protokollauswertungs- und Filterfunktionen müssen von den Ausgabefunktionen unabhängig funktionieren.
- Das Datenmodell muss performantes Arbeiten ermöglichen, der Entwurf muss die entsprechende Funktionsweise der Anwendungsteile begünstigen.

4.3.2 Ereignisbasierte Verarbeitung von Protokolldaten

Beim Entwurf des Datenmodells muss u. a. festgelegt werden wie die Protokolldaten repräsentiert werden können. Dabei werden das Format der Eingangsdaten und die Arbeitsweise der Verfahren betrachtet, die von den Anwendungsteilen zur Umsetzung der geforderten Funktionalität eingesetzt werden. Die Verfahren können bestimmte Datenformate voraussetzen. Bei dieser Betrachtung wird zwischen den samplebasierten und ereignisbasierte Formaten unterschieden.

Die ereignisbasierten und samplebasierten Daten können auf verschiedene Arten durch die Messhardware kombiniert und an die Anwendung übergeben werden. Die samplebasierten Daten haben deutlich höhere Datenraten und benötigen deutlich mehr Speicherplatz als eventbasierte Daten, deshalb werden die ereignisbasierten Daten als Kompromiss eingesetzt.

Die ereignisbasierten und samplebasierten Daten können wie in Bild 4.3.2.a von der Messhardware kombiniert werden (Anforderungen 3.2.2):

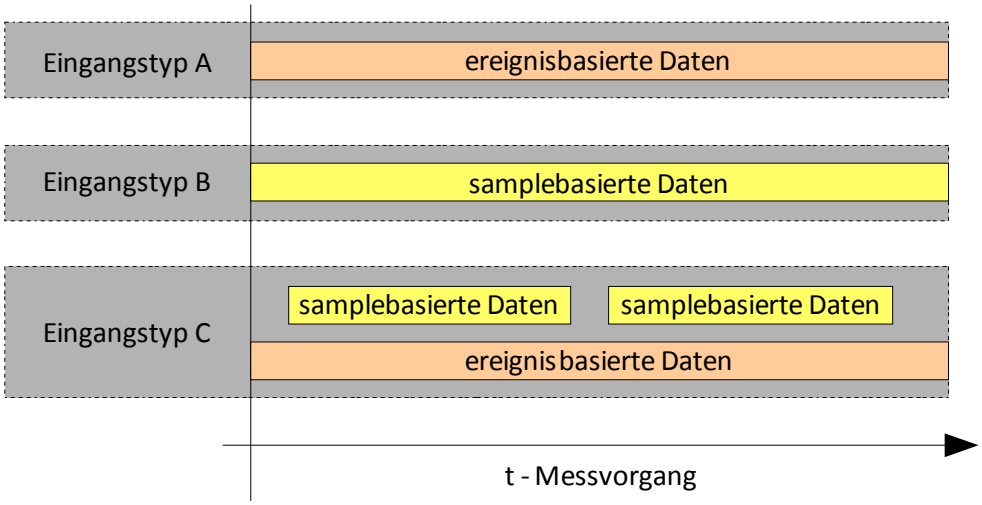

Bild 4.3.2.a: Eingangstypen Messdaten

Im Folgenden wird festgestellt welche Anwendungsteile mit samplebasierten und welche mit ereignisbasierten Daten arbeiten:

- Die Automaten für die Protokollerkennung (Abschnitt 4.4) arbeiten mit ereignisbasierten Ein- und Ausgabedaten.
- Die Protokollfilter-, Such- und Triggerfunktionen (Abschnitt 4.5) arbeiten mit den Ausgabedaten der Protokollerkennungsautomaten, daher arbeiten sie auch mit ereignisbasierten Daten.
- Die Ausgabefunktionen (Abschnitt 4.7) verwenden die ereignisbasierten Daten der Protokollautomaten, sie benötigen aber zusätzlich auch die samplebasierten Eingangsdaten für die Ausgabe der physikalischen Protokollebene (Anforderung Abschnitt 3).

Die samplebasierten Daten werden nur durch die Ausgabefunktionen bei der Ausgabe der physikalischen Protokollebene benötigt. Die restlichen Anwendungsteile arbeiten mit ereignisbasierten Daten. Bei dem Eingangstyp B, bei dem nur samplebasierte Daten vorliegen,

müssen diese Daten daher in ereignisbasierte Daten konvertiert werden, damit die weite Auswertung durch die entsprechenden Anwendungsteile erfolgen kann.

Im Folgenden werden zwei Lösungen betrachtet, wie die samplebasierten und eventbasierten Daten durch die Anwendung verwaltet werden können.

Lösung Nr. 1: Samplebasierte und ereignisbasierte Daten

Die samplebasierten und ereignisbasierten Daten werden parallel durch die Anwendung verwaltet. Beide Datentypen werden für die weitere Auswertung verwendet wie u. a. Protokollauswertung und Filterfunktionen. Beim Eingangstype B, bei dem nur samplebasierte Daten vorliegen, werden die samplebasierten Daten nach Bedarf durch den Anwendungskern in ereignisbasierte Daten konvertiert. (Bild 4.3.2.b)

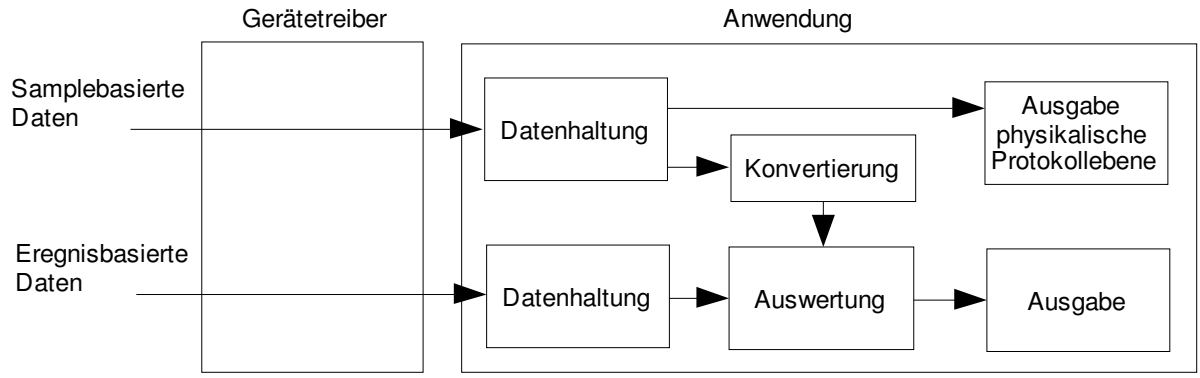

Bild 4.3.2.b: Lösung Nr.1 Samplebasierte und ereignisbasierte Daten

Lösung Nr. 2: Ereignisbasierte Daten, samplebasierte Daten nur eingeschränkt

Der Anwendungskern arbeitet nur mit ereignisbasierten Daten. Die samplebasierten Daten werden nur zur Ausgabe der physikalischen Protokollebene verwendet. Beim Eingangstype B, bei dem nur samplebasierte Daten vorliegen, werden die samplebasierten Daten durch den Gerätetreiber in eventbasierte Daten konvertiert. Der Gerätetreiber übergibt dann die ereignisbasierten und samplebasierten Daten. (Bild 4.3.2.c)

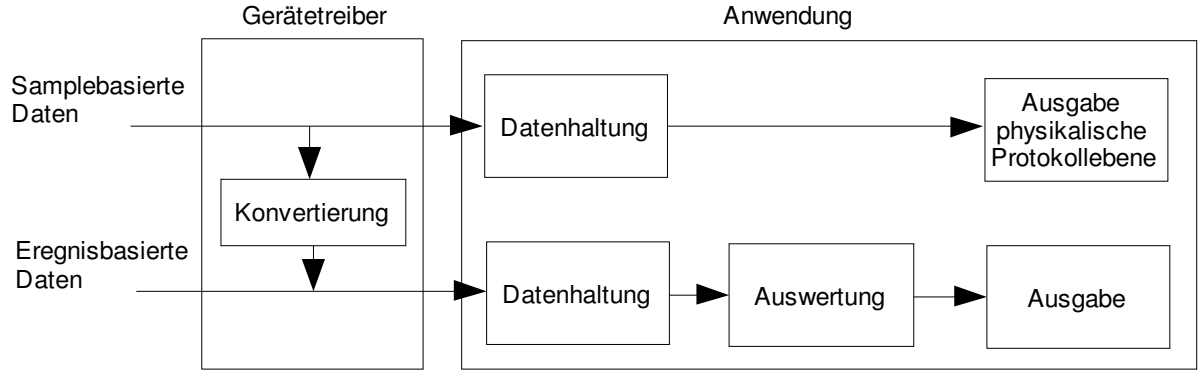

Bild 4.3.2.c: Lösung Nr.2 Ereignisbasierte Daten, samplebasierte Daten nur eingeschränkt

Ein Vorteil der zweiten Lösung ist die schnellere Verarbeitung der ereignisbasierten Daten im Vergleich zu den samplebasierten Daten, da sie bereits im geeigneten Format zur Weiterverarbeitung vorliegen. Bei der ersten Lösung müssen die samplebasierten Daten jedesmal vor der Weiterverarbeitung in ereignisbasierte Daten konvertiert werden. Werden z.B. mehrere Suchfunkionen hintereinander ausgeführt, müssen die Daten für jede suchanfrage u.U. erneut konvertiert werden.

Ein weiterer Vorteil der zweiten Lösung ist die vereinfachte Datenhaltung vom Anwendungskern. Die Konvertierungsfunktionen, die die samplebasierten Daten in ereignisbasierte Daten konvertieren, benötigen geräteabhängige Parameter. Die Komplexität vom Anwendungskern wird verringert, wenn die Konvertierungsfunktionen durch den Gerätetreiber implementiert werden. Der Anwendungskern kann die ereignisbasierten Daten im Gegensatz zu den samplebasierten Daten auf einheitliche Weise verarbeiten.

Ein Nachteil der zweiten Lösung ist die doppelte Datenhaltung, wenn, wie bei Gerätetype A, die Sampledaten vollständig gespeichert werden. In diesem Falle werden auch die ereignisbasierten Daten parallel gespeichert und der Speicherplatzbedarf ist größer als bei der ersten Lösung, die in diesem Falle nur die samplebasierten Daten speichert.

Insgesamt überwiegen die Vorteile der zweiten Lösung, daher wird diese hier eingesetzt.

4.3.3 Getrennte Module für Protokollebenen

Das Einbinden und Anpassen weiterer Busprotokolle ist eine wesentliche Anforderung. Die Protokolle sind zum Entwurfszeitpunkt noch nicht bekannt. Es muss daher eine generelle Implementierungsweise gefunden werden, die das Arbeiten mit dem allgemeinen Aufbau eines Busprotokolls ermöglicht, um die größtmögliche Flexibilität zu erhalten. In der Regel, bestehen die Protokolle aus Protokollebenen die aufeinander aufbauen. Dabei verwenden die höheren Protokollebenen unterschiedlicher Protokolle oft die gleichen unteren Protokollschichten. Es gibt z.B. eine Reihe von Protokollen die auf dem I2C Bus aufsetzen.

Um eine geeignete Implementierung zu bestimmen, werden zwei mögliche Lösungen untersucht. Bei der ersten Lösung (Bild 4.3.3.a), werden Schnittstellen zwischen den Anwendungsteilen vorgegeben. Die Implementierungen der Anwendungsteile können protokollbezogen frei gewählt werden. Eine Trennung nach Protokollebenen ist nicht vorgeschrieben. Bei der zweiten Lösung (Bild 4.3.3.b) wird eine zusätzliche Trennung nach Protokollebenen vorgenommen.

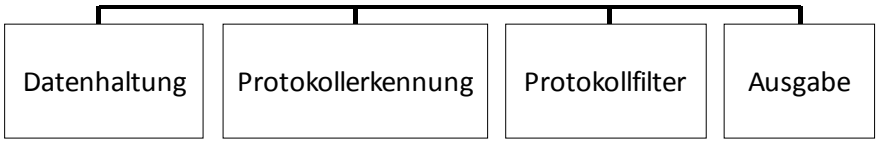

Bild 4.3.3.a: Schnittstellen zwischen Anwendungsteilbereichen, keine einheitliche Trennung der Protokollebenen

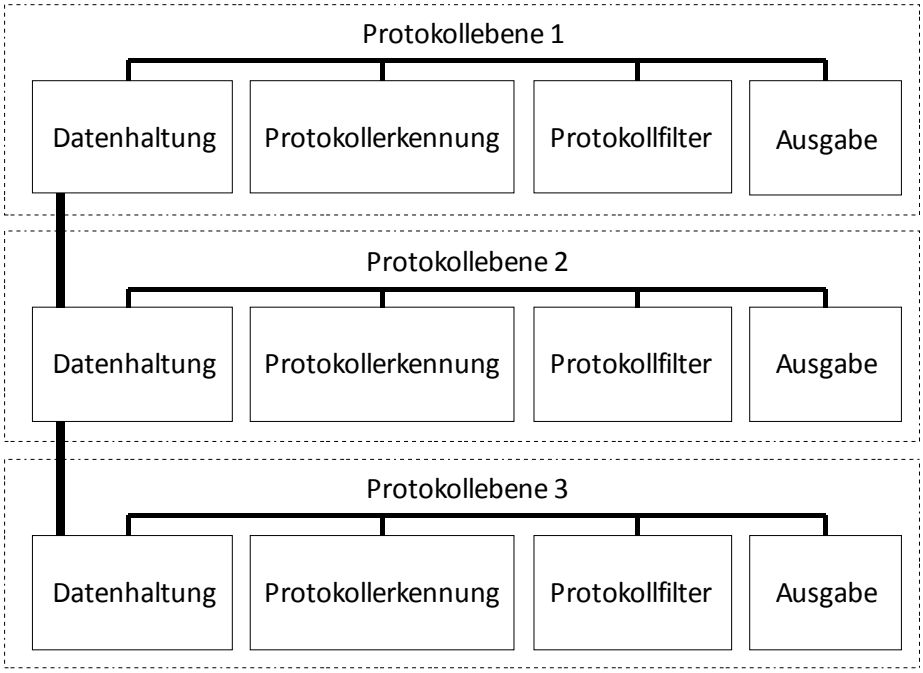

Bild 4.3.3.b: Schnittstellen zwischen Anwendungsteilbereichen und Protokollebenen

Die Vor- und Nachteile der beiden Lösungswege:

1. Aufteilung nach Anwendungsbereichen, Bild 4.3.3.a
   - Vorteile:
     - Die Aufteilung in Protokollschichten ist nicht vorgeschrieben, dies gewährt der Implementierung größere Freiheiten und kann protokollabhängig erfolgen. Die Protokollerkennung kann z.B. für alle Protokollschichten zusammenhängend Implementiert werden.
     - Die Einteilung der Datenstrukturen kann Protokollabhängig gewählt werden.
   - Nachteile:
     - Die Wiederverwendbarkeit vorhandener Protokollschichten wird erschwert, da nicht nach Protokollschichten getrennt wird.

2. Die Einteilung der Anwendungsbereiche nach Funktion und Protokollebenen, Bild 4.3.3.b
   - Vorteile:
     - Das Wiederverwenden der Protokollebenen wird durch die vorgegebene Aufteilung nach Protokollebenen erleichtert.
     - Durch die Aufteilung in Protokollebenen und die damit verbundenen Schnittstellen, steigt die Übersicht bei der Anpassung und hinzufügen neuer Protokollebenen
   - Nachteile:
     - Durch die zusätzlichen Schnittstellen, die durch die Aufteilung in Protokollebenen entstehen, steigt der Implementierungsaufwand.
     - Die Trennung der Protokollschichten kann u. U. die Kompatibilität zu derzeit unbekannten Protokollen einschränken.

Bei der Implementierung wird die Trennung nach Funktion und Protokollebenen eingesetzt (Bild 4.3.3.b). Diese Implementierungsweise entspricht der Anforderung das Einbinden und Erweitern von neuen Protokollen bzw. Protokollschichten zu begünstigen.

4.3.4 Datenstrukturen für eventbasierte Protokolldaten

Die Datenstrukturen für Protokolldaten müssen die Eingangsdaten (Abschnitt 4.2.4) und die Eventdaten aus der Protokollauswertung (Abschnitt 4.4) erfassen.

Die Eventdaten werden durch Objekte modelliert. Um die Struktur der Objekte zu bestimmen, werden die Eigenschaften der Events und die Anforderungen an die Eventobjekte untersucht.

Eigenschaften der Events:

- Die einfachste Form, mit der ein Event beschrieben werden kann, ist durch einen Zeitpunkt und einen Typ (Abschnitt 2.3.2).
- Komplexe Events können aus mehreren Zeitpunkten und Typen bzw. Eigenschaften bestehen.
- Die Events, die durch das Auswerten der Eingangsevents, durch die Protokollerkennungsautomaten entstehen (Abschnitt 4.4), stellen eine hierarchische Gruppierung der ursprünglichen Eingangsevents dar. Die Reihenfolge in der die Events von den Protokollerkennungsautomaten ausgegeben werden ist anders als die Reihenfolge der Events, sortiert nach ihrem Startzeitpunkt. Das Bild 4.3.4.a zeigt die Erkennungs- und Ereignisreihenfolge der Events. Die Erkennungsreihenfolge ist die Reihenfolge, in der die Events durch die Protokollerkennungsautomaten ausgegeben werden. Die Ereignisreihenfolge ist die Reihenfolge der Events sortiert nach ihrem Ereigniszeitpunkt. In diesem Beispiel gibt der Erkennungsautomat das Byte erst aus, nachdem alle acht Bits erkannt werden. Dieses Verhalten gilt auch für höhere Protokollebenen, so dass z.B. eine IPMB Nachricht erst ausgegeben wird, nachdem alle der Nachricht zugehörigen Bits vorhanden sind.

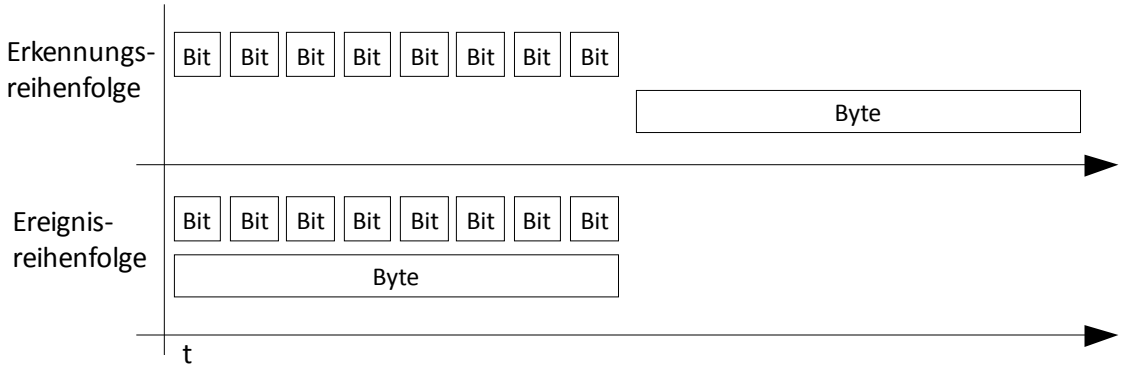

Bild 4.3.4.a: Erkennungs- und Ereignisreihenfolge der Protokollevents

Anforderungen an die Eventobjekte und Eventdatenstrukturen:

- Welche Daten und Eigenschaften von den Eventobjekten erfasst werden müssen, richtet sich nach der Ausgabe der Protokollautomaten (Abschnitt 4.4.3).
    - Ein Event, das z.B. für ein Übertragens Byte steht, kann einen Start- und Endzeitpunkt besitzen und als Eigenschaft das übertragene Byte.
- Die Anforderungen und Arbeitsweise der Anwendungsteile, die mit den Eventobjekten arbeiten, müssen berücksichtigt bzw. sollen begünstigt werden.
    - Die Ausgabefunktionen (Abschnitt 4.7) benötigen die Events nach Eventzeitpunkt und Protokollschicht sortiert.

  - Die Filter und Suchfunktionen (Abschnitt 4.5) benötigen die Events nach Eventzeitpunkt und Protokollschicht sortiert.
  - Die Anwendung arbeitet Queuebasiert nach Eventzeitpunkt sortiert, um große Datenmengen abschnittsweise bearbeiten zu können (Abschnitt 4.3.5).
  - Die Ausgabefunktionen benötigen die hierarchische Ordnung der Eventobjekte, z.B. besteht ein Byteevent aus dem entsprechenden Bitevents (Bild 4.3.4.a).
- Implementierungsrichtlinien müssen berücksichtigt werden.
  - Im vorherigen Abschnitt 4.3.3. wurde eine Trennung nach Protokollschicht festgelegt.

Die einzelnen Eventobjekte müssen durch eine geeignete Datenstruktur verwaltet werden können. Es gibt eine Vielzahl an möglichen Datenstrukturen. Im Folgenden werden drei anhand der Anforderungen untersucht. Bei allen drei Möglichkeiten werden die Eventobjekte von einem Basisobjekt mit einem Eventzeitpunkt abgeleitet.

1. Eventqueue sortiert nach Eventzeitpunkt, Bild 4.3.4.b
- Die Eventobjekte werden sortiert nach Eventzeitpunkt in einem Eventqueue angeordnet.

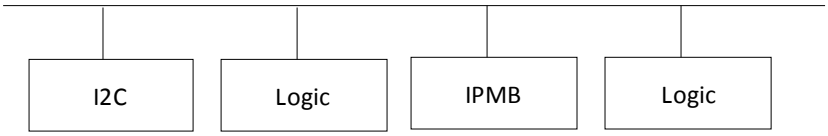

Bild 4.3.4.b Eventdatenstruktur 1

- Vorteile
  - Operationen können durch die flache Hierarchie einheitlich ausgeführt werden.
- Nachteile
  - Eine Sortierung beim Einfügen ist nötig, da die Protokollerkennung die Events in der Reihenfolge der Protokollschichten liefert, z.B. erst die Bitevents, dann das entsprechende Byteevent. Das Byteevent muss aber nach Startzeit vor den Bitevents einsortiert werden (Bild 4.4.3.a).
  - Durch die Speicherung der Events als Basisdatentypen in einer flachen Hierarchie, müssen typspezifische Operationen, wie Such- und Filterfunktionen, alle Events durchlaufen.
  - Die hierarchische Ordnung der Events wird nicht festgehalten.

2. Eventqueue sortiert nach Eventzeitpunkt und Eventhierarchie, Bild 4.3.4.c
- Die Eventobjekte werden mit der höchsten Protokollschicht zuerst hierarchisch untergeordnet, nach Eventzeitpunkt sortiert, angeordnet.

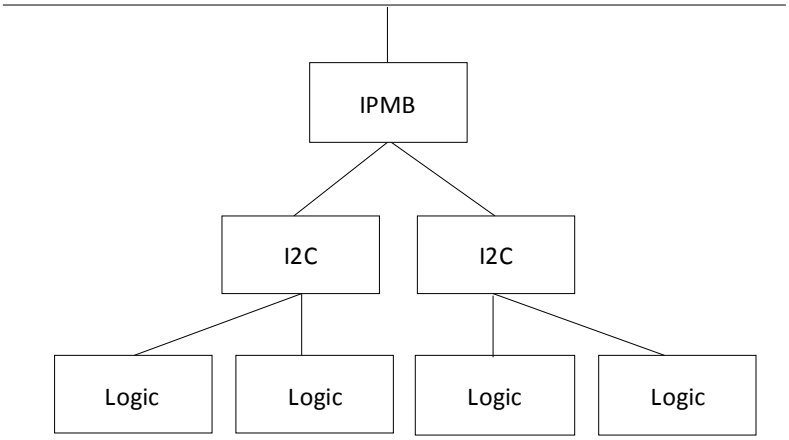

Bild 4.3.4.c Eventdatenstruktur 2

- Vorteile
  - Die Hierarchie der Events wird durch die Anordnung festgehalten.
- Nachteile
  - Wie bei der ersten Lösung ist eine Sortierung beim Einfügen notwendig. Für Protokollschicht bezogene Operationen müssen die Objekte übergeordneter Protokollschichten mit durchlaufen werden.

3. Getrennte Eventqueues nach Protokollebene, jeweils sortiert nach Eventzeitpunkt, Bild 4.3.4.d

- Die Eventobjekte werden in Queues gespeichert, getrennt nach Protokollebene und sortiert nach Eventzeitpunkt. Zusätzlich wird die Eventhierarchie durch Referenzen der Eventobjekte untereinander festgehalten, hier als gestrichelte Linie dargestellt.

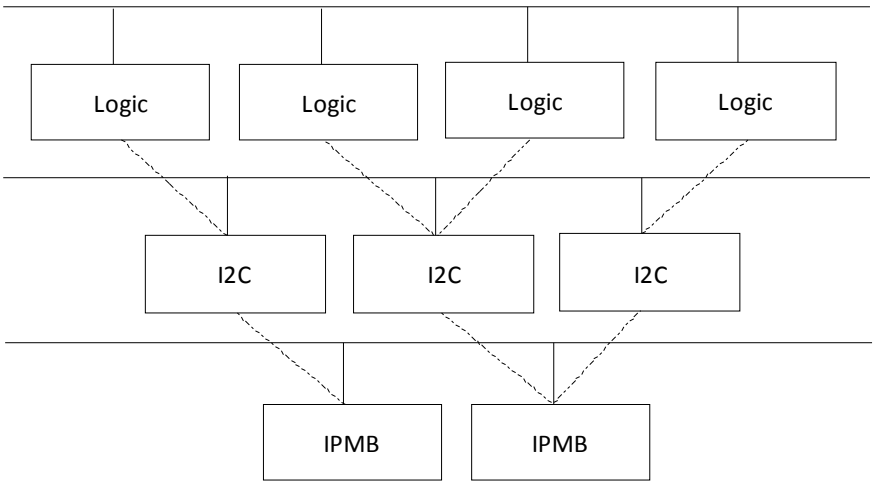

Bild 4.3.4.d Eventdatenstruktur 3

- Vorteile
  - Die Hierarchie der Events wird durch die Anordnung festgehalten.
  - Die durch die Protokollerkennung vorhandene Sortierung nach Protokollschicht wird erhalten.
- Nachteile

- Es wird zusätzlicher Speicherplatz für Referenzen zu untergeordneten Eventobjekten benötigt.

Die benötigte Funktionalität kann mit allen drei Möglichkeiten zwar umgesetzt werden, die ersten beiden Lösungen erfordern jedoch mehr Sortierungsaufwand zur Laufzeit, um die wesentlichen Anwendungsfunktionen umzusetzen. Daher wird die dritte Lösung hier Implementiert.

4.3.5 Bufferkonzept für große Datenmengen

Die Dauer einer zusammenhängenden Messung ist mit bis zu fünf Tagen durch die Anforderungen vorgegeben. Aufgrund der in Abschnitt 2.3 ermittelten Datenraten wird davon ausgegangen, dass die kompletten Daten einer Messung nicht gleichzeitig im Arbeitsspeicher abgelegt werden können. Um unabhängig vom Arbeitsspeicher beliebig große Datensätze bearbeiten zu können, wird eine Bufferstruktur eingesetzt, die in drei Ebenen eingeteilt ist. Das Bild 4.3.5.a, zeigt die drei Bufferebenen und die Protokolldaten die pro Ebene verwaltet werden.

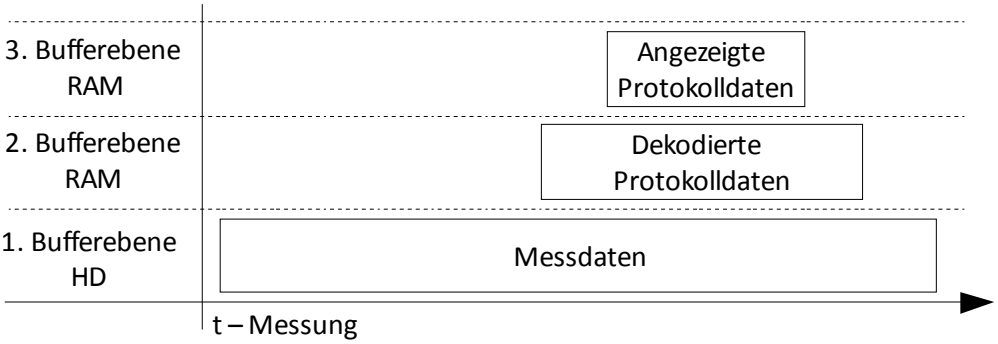

Bild 4.3.5.a: Bufferkonzept für Protokolldaten

Die erste Bufferebene ist Dateibasiert und speichert die eingehenden Messdaten. Diese Ebene kann die durch das Hostsystem zur Verfügung gestellten Laufwerke nutzen, um die Messdaten zu speichern. Die Größe der Messdatendatei ist nur limitiert durch die Fähigkeiten vom eingesetzten Dateisystem bzw. durch den freien Speicherplatz des Hostlaufwerks.

Die zweite Bufferebene speichert die dekodierten Protokolldaten der Protokollauswertung (Abschnitt 4.4). Die dekodierten Protokolldaten werden in einem FIFO Buffer (First In - First Out Buffer)im Arbeitsspeicher gehalten. Die dekodierten Protokolldaten werden nur für ein bestimmtes Zeitfenster vorgehalten, wie Bild 4.3.5.a zeigt. Die Buffergröße ist so gewählt, dass der Buffer in den Arbeitsspeicher passt. Dadurch kann eine beliebig große Messdatendatei bei der Verarbeitung durchlaufen werden. Diese Bufferebene arbeitet unabhängig der dritten Bufferebene für die Ausgabe der Protokolldaten, damit Operationen wie u. a. Filter- und Suchfunktionen (Abschnitt 4.5) mit möglichst geringem Aufwand zur Laufzeit ausgeführt werden können.

Die dritte Bufferebene verwaltet die Protokolldaten die aktuell angezeigt werden. Die Protokolldaten werden durch spezielle Objekte verwaltet, die in Abschnitt 4.7 beschrieben werden. Der Buffer ist wie die zweite Bufferebene als FIFO ausgelegt und wird im Arbeitsspeicher gehalten.

4.3.6 Datentypen für Ereigniszeitpunkte

Damit Protokolldaten in einem eventbasierten Format gespeichert werden können, müssen u. a. die Zeitpunkte der Ereignisse gespeichert werden. Um diese Zeitpunkte zu speichern, müssen geeignete Datentypen bestimmt werden. Die Zeitpunkte werden dabei grundsätzlich relativ zu einem bekannten Zeitpunkt ausgedrückt.

Der verwendete Datentyp kann aus mehreren primitiven Typen zusammengesetzt werden, wobei z.B. Stunden, Minuten, Sekunden, etc. getrennt gespeichert werden, oder es kann ein einzelner primitiver Datentyp verwendet werden. Voraussetzung ist, dass die Zeitpunkte mit der geforderten Genauigkeit abgebildet werden können. Die Verwendung eines einzelnen primitiven Datentyps ist die bevorzugte Lösung, Berechnungen und Vergleiche können dadurch mit weniger Aufwand zur Laufzeit durchgeführt werden, die Aufteilung in ein getrenntes Format wie z.B. „HH:MM:SS“ wird nur nach Bedarf durchgeführt.

Um einen Datentyp zu bestimmen, werden die Eigenschaften der darzustellenden Zeitpunkte aus den Anforderungen ermittelt. Dabei werden die geforderte Genauigkeit und der größte mögliche Unterschied zwischen zwei Zeitpunkten bestimmt.

Bei der Verwendung einer einzelnen Variablen, entspricht jeder Variablenwert einem festen Zeitabschnitt. Der Wertebereich der Variable ist linear auf den darzustellenden Zeitraum abgebildet. Die kleinste darstellbare Zeiteinheit entspricht somit dem Unterschied zwischen zwei Variablenwerten. Der maximale darstellbare Zeitpunkt, wird von der kleinsten Zeiteinheit und der Anzahl an Variablenwerten bestimmt.

$$max.Zeitpunkt = kleinste\ Zeiteinheit \cdot Anzahl\ Variablenwerte$$

Um den Anforderungen zu entsprechen, müssen Zeitpunkte mindestens mit der gleichen Genauigkeit der Eingangsdaten gespeichert werden können. Die maximale Abtastrate der samplebasierten Eingangsdaten beträgt 2 ns. Die geforderte maximale Messdauer beträgt 5 Tage. Mit der obengenannten Formel kann die Dimensionierung der Variablengröße vorgenommen werden. Um der Genauigkeit der Abtastrate zu entsprechen, entspricht jeder Variablenwert der Abtastrate bzw. Abtastdauer von 2 ns.

Tabelle 4.3.6.a: Maximale Messdauer abhängig von Abtastrate und Variablengröße

Abtastrate (Zeit pro Variablenwert)	**Variablengröße**	**max. relativer Zeitpunkt (mögliche Messdauer)**
500 MHz (2 ns)	32 Bit Integer	8.58 s
500 MHz (2 ns)	64 Bit Integer	427007 Tage

Aus der Tabelle 4.3.6.a geht hervor, dass ein 64 Bit Integer die Anforderung an Messdauer und Genauigkeit erfüllt.

4.4 Protokollauswertung

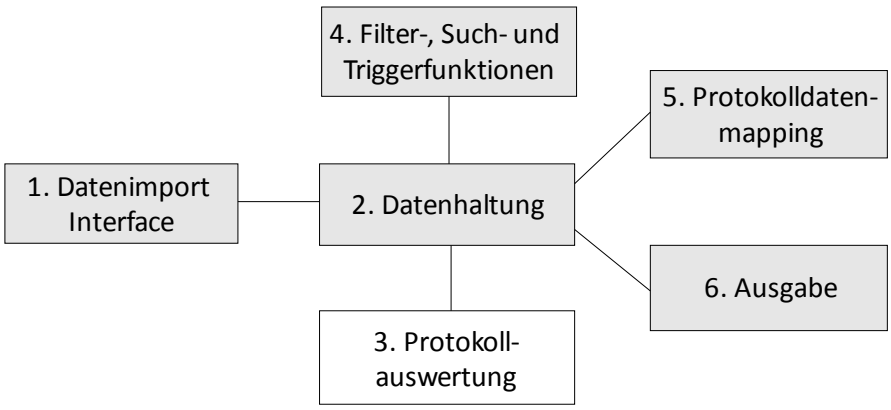

Bild 4.4.a: Einordnung Protokollauswertung

In diesem Kapitel wird die Arbeitsweise der Protokollauswertung vorgestellt. Die Hauptaufgabe der Protokollauswertung besteht darin, die Eingangsmessdaten auszuwerten bzw. die entsprechenden Protokolle zu finden und die resultierenden Daten für die Weiterverarbeitung bereitzustellen. Dabei müssen die Daten der unterschiedlichen Protokollebenen verifiziert und Fehler erkannt werden können.

Folgende Punkte werden besprochen:

- Übersicht Anforderungen
- Protokollauswertung durch Automaten
- Eingabe und Ausgabeformate der Automaten
- Verifizierung und Fehlererkennung
- Beispiel IPMB Protokollschicht

4.4.1 Übersicht Anforderungen

Aus den Anforderungen in Abschnitt 3 ergeben sich die folgenden Kriterien, die bei der Protokollauswertung berücksichtigt werden müssen:

- Bei der Protokollauswertung müssen die Eingangsdaten ausgewertet werden. Dabei müssen die Protokollstrukturen und die übertragenen Daten erkannt werden.
- Es müssen Fehler erkannt und ausgewiesen werden können.
- Es müssen Protokollstrukturen und Protokolldaten verifiziert werden können.
- Die resultierenden Daten müssen in geeigneter Form vorliegen, damit Filter und Ausgabefunktionen ermöglicht werden.
- Die Protokollauswertung soll unabhängig von der grafischen Ausgabe funktionieren.

4.4.2 Protokollauswertung durch Automaten

Für die Protokollauswertung werden endliche Automaten eingesetzt. Diese Automaten werden häufig eingesetzt, um Busprotokolle zu erkennen und lassen sich gut implementieren. (vgl. [hopcraft], S. 45ff])

4.4.3 Eingabe und Ausgabeformate der Protokollautomaten

Die Protokollautomaten arbeiten mit ereignisbasierten Daten (Abschnitt 2.3.2).

Durch die Trennung nach Protokollebenen, wie in Teil 4.3.3 beschrieben, werden getrennte Automaten für jede Protokollebene eingesetzt.

Jeder Automat besitzt eine Eingabe und Ausgabe, die durch die entsprechende Protokollebene vorgegeben ist. Die Automaten bilden eine Verarbeitungskette, wobei die Eingabe einer Schicht mit der Ausgabe der vorherigen korrespondiert. Bild 4.4.3.a zeigt ein Beispiel einer Verarbeitungskette für die Protokollschichten Logic, I2C und IPMB.

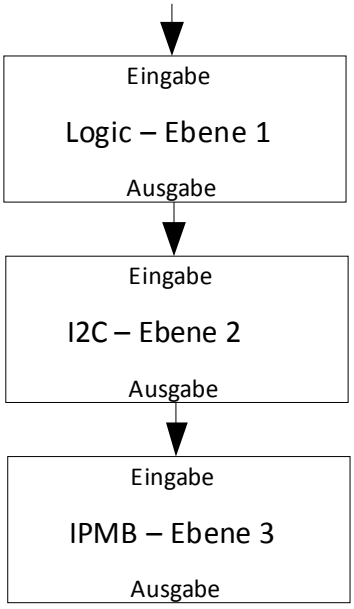

Bild 4.4.3.a Verarbeitungskette der Protokollautomaten

4.4.4 Verifizierung und Fehlererkennung

Um eine Strategie für die Verifizierung und Fehlererkennung festzulegen, werden zunächst die zu verifizierenden Bereiche, die Eingangsdaten und die Qualität der Verifizierung klassifiziert. Es wird auf den allgemeinen Ablauf der Strategiefindung eingegangen und ein konkretes Beispiel für die IPMB Protokollschicht vorgestellt.

Bei der Übertragung durch ein Bussystem gibt es eine Reihe von Fehlerquellen bzw. Bereiche, die Verifiziert werden können. Diese Bereiche werden hier wie folgt klassifiziert.

Verifizierungsklassen

- Verifizierung der Sender
  - Korrektes Senden u. Empfangen auf der physikalischen Busebene
  - Einhaltung von Protokollkonventionen wie z.B. Headerformat, CRC Prüfsummen Adressierung und Zeitlimits
  - Verifizierung der gesendeten Nutzinformation
- Verifizierung der Empfänger
  - Korrektes Senden u. Empfangen auf der physikalischen Busebene
  - Einhaltung von Protokollkonventionen, wie z.B. ACK Bestätigung
  - Verifizierung der empfangenen Nutzinformation
- Verifizierung der Übertragung auf dem Bus
  - Korrekte Übertragung auf der physikalischen Busebene

Welche Fehler automatisiert erkannt bzw. Verifizierungen automatisiert durchgeführt werden können, hängt von den Informationen ab, die bei der Verarbeitung durch die Protokollautomaten zur Verfügung stehen. Diese Informationen werden hier wie folgt klassifiziert.

Informationsklassen

- Messdaten – Die Daten, die durch das Beobachten der Busleitungen gesammelt werden.
- Protokollspezifikation – Die Beschreibung von Protokollstrukturen und Konventionen wie z.B. Header und Prüfsummen.
- Busteilnehmerinformation – Die Konstellation und Adressierung der Busteilnehmer.
- Nutzdatenformatierung – Informationen über die Nutzdaten wie Format und Kennzeichnung und ggf. Prüfsummen.
- Konkrete Nutzdaten – Vorgegebene Nutzdaten und Kommunikationsmuster in Form von Testfällen.

Die erkannten Fehler bzw. mögliche Verifikation und deren Genauigkeit werden hier wie folgt klassifiziert.

Fehlerqualitätsklassen

- Fehler nicht erkennbar, bzw. nicht verifizierbar
- Fehler erkennbar, Zuordnung zu einer konkreten *Verifizierungsklasse* nicht möglich
- Fehler erkennbar, Zuordnung zu einer konkreten *Verifizierungsklasse* möglich

Welche *Informationsklassen* bei der Protokollerkennung verwendet werden, hängt von den Anforderungen ab und beeinflusst die resultierenden *Verifizierungs- und Fehlerqualitätsklassen*. Die Architektur erlaubt es, für unterschiedliche Protokolle bzw. Anforderungen unterschiedliche Konfigurationen an *Informationsklassen* zu verwenden.

Im Folgeneden werden allgemeine Eigenschaften zu den *Informationsklassen* aufgeführt, die beim Entwurf berücksichtigt werden.

- Die *Messdaten* werden aus der Sicht eines Busteilnehmers gewonnen. Durch die Busarchitektur ist es nicht möglich, festzustellen, welcher Teilnehmer die Daten tatsächlich sendet. Rückschlüsse auf den Sender können nur durch die Auswertung der Übertragung erfolgen, formal betrachtet, können nur Wahrscheinlichkeiten festgelegt werden.
- Die *Protokollspezifikationen* können unvollständig sein. Es kann zu nicht eindeutigen Interpretationen kommen, besonders bei einer strikteren Verifizierung.
- Die *Busteilnehmerinformation* und die *Nutzdatenformatierung* können durch die Automaten verwendet werden, um z.B. Prüfsummen innerhalb der Nutzdaten zu überprüfen. Sie können aber auch nur bei der späteren Ausgabe als Information für den Anwender verwendet werden.
- *Konkrete Nutzdaten* für einen Test vorzugeben, hat den Vorteil, auch komplexe Kommunikationsabläufe automatisiert verifizieren zu können. Es bedarf dafür aber u.U. einen hohen Aufwand um einen solchen Test vorzubereiten.

Für allgemeine Protokollimplementierungen werden die *Messdaten* und die *Protokollspezifikationen* verwendet. Zusätzlich können die *Busteilnehmerinformation* und die *Nutzdatenformatierung* verwendet werden. *Konkrete Nutzdaten* werden nur in speziellen Fällen zum Einsatz kommen, da die Vorgabe der Nutzdaten den allgemeinen Einsatz einschränkt.

Den Ursprung bzw. die *Verifizierungsklasse* eines Fehlers, kann i.d.R. nicht automatisiert bestimmt werden. Da die *Messdaten*, wie oben beschrieben aus Sicht eines Busteilnehmers vorliegen, ist i.d.R. nicht automatisiert ermittelbar, ob der Fehler bei der Senderlogik, Übertragung vom Sender zum Bus oder durch die Übertragung auf dem Bus aufgetreten ist. Daher beschränken sich die *Fehlerqualitätsklassen* i.d.R. auf erkennbare und nicht erkennbare Fehler. Die weitere Bestimmung der Fehlerursache kann durch den Anwender erfolgen, z.B. durch ein Ausschlussverfahren.

Für Protokolle mit vorgegebenen Dialogstrukturen zwischen Busteilnehmern kann die Verifizierung der Dialogabläufe durch die Protokollerkennungsautomaten erfolgen. Es muss im Einzelfall entschieden werden, da sich der Implementierungsaufwand entsprechend erhöht.

Wird ein Fehler durch einen Protokollerkennungsautomaten festgestellt, ist es von Vorteil, wenn der Fehler entsprechend erfasst wird, die Erkennung vom restlichen Nachrichtenabschnitt jedoch weiterläuft. Bei normalen Anwendungen, die nicht an einer Auswertung von fehlerhaften Übertragungen interessiert sind, wird häufig eine komplette Nachricht ignoriert, wenn z.B. ein ACK nicht gesendet wird, oder eine Prüfsumme falsch ist. Für die Auswertung durch den Anwender ist es übersichtlicher, wenn die fehlerhafte Nachricht trotzdem auf der höchstmöglichen Protokollebene erkannt wird, und der erkannte Fehler entsprechend ausgewiesen wird.

Durch diese Vorgehensweise, die Auswertung eines Events nicht beim ersten Fehler abzubrechen, können auch mehrere Fehler pro Event erkannt werden. Die Speicherung der Fehler in den Ausgabeevents erfolgt daher in Listenform.

Im folgenden Abschnitt wird an einem Automaten für die IPMB Protokollschicht vorgestellt, welche *Informationsklassen* zum Einsatz kommen und welche Möglichkeiten zur Fehlererkennung sich daraus ergeben.

4.4.5 Beispiel IPMB Protokollschicht

Die Eingabeevents für den Protokollerkennungsautomaten der IPMB Protokollschicht Bild 4.4.5.a, entsprechen den Ausgabeevents der darunterliegenden I2C Protokollschicht.

Eingabeevents IPMB Protokollschicht:

- Start
- Stop
- Data (Address)
- Data ACK (Address ACK)

Ausgabeevents IPMB Protokollschicht:

- IPMB Event

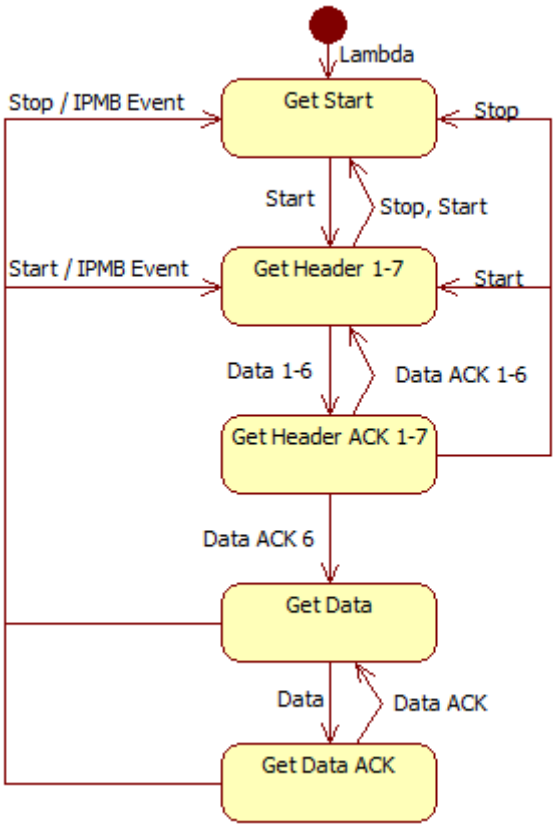

Bild 4.4.5.a: Protokollerkennungsautomat für die IPMB Protokollschicht

Das IPMB Protokoll wird in Teil 3 vorgestellt. Vereinfacht dargestellt, besteht eine IPMB Nachricht aus einem Header von 7 Bytes und anschließenden X Bytes an Nutzdaten. Jedes Byte wird gefolgt von einem ACK Bit.

In den Automatenzuständen werden Prüfungen durchgeführt, so ist z.B. das dritte Byte im Header eine Prüfsumme aus den ersten zwei Bytes, diese wird überprüft und ggf. ein Fehler vermerkt. Da

jeder Eingabeevent einen Timestamp besitzt, kann auch das Einhalten von Zeitlimits entsprechend überprüft werden.

Es gibt bei diesem Automaten nur eine Eingangsfolge, um eine IPMB Nachricht fehlerlos zu übertragen. Dazu muss im Zustand „Get Data“ ein Stop übertragen werden, dann erfolgt die Ausgabe von einem IPMB Event. Trotzdem werden auch durch andere Eingangsfolgen IPMB Events ausgegeben. Diese IPMB Events werden jedoch durch Fehler gekennzeichnet, da z.B. ein ACK oder ein Stop fehlt. Eine andere Möglichkeit wäre, diese fehlerhaften IPMB Nachrichten nicht auf der IPMB Ebene auszugeben, sondern nur auf der darunterliegenden I2C Ebene und nicht als IPMB Nachrichten gekennzeichnet. Das würde jedoch die Zuordnung als Fehlerhafte IPMB Übertragung durch den Anwender erschweren, deshalb werden diese Daten mit der entsprechenden Fehlerkennzeichnung auf der IPMB Ebene ausgegeben.

Der hier implementierte IPMB Automat könnte auch bei fehlerhaften Headerdaten ein IPMB Event mit Fehlerkennzeichnung ausgeben, hier wird aber ein vollständiger Header verlangt. Diese Entscheidung wurde getroffen, da dieser Automat allgemein für das IPMB Protokoll einsetzbar sein soll. Es kann auf dem gleichen Datenbus auch andere I2C Nachrichten geben, diese sollen nicht als fehlerhafte IPMB Nachrichten ausgegeben werden.

Wie die zwei vorherigen Beispiele zeigen, können unterschiedliche Konfigurationen der Automaten für eine Protokollschicht sinnvoll sein. Die Ein- und Ausgabeevents der Automaten dienen als Schnittstelle. Es können Automaten mit unterschiedlichem Verhalten bereitgestellt werden. Der Anwender kann so das gewünschte Verhalten bestimmen.

Für die hier vorgestellte IPMB Protokollschicht, kommen folgende *Informationsklassen (Klassifizierung unter Punkt 4.4.4)* zum Einsatz:

- Messdaten
- Protokollspezifikation (IPMB v1.0)
- Busteilnehmerinformation (via XML Datei)
- Nutzdatenformatierung (via XML Datei)

Der Protokollautomat verwendet die Messdaten und die Protokollspezifikation. Zusätzlich wird die Nutzdatenformatierung verwendet, um Prüfsummen und Länge der Nutzdaten automatisiert zu überprüfen. Die Nutzdatenformatierung und die Busteilnehmerinformation sind so gespeichert, dass sie durch den Anwender leicht angepasst werden können. Das Verfahren wird in Teil 4.6 genauer beschrieben.

Für die hier vorgestellte Implementierung der IPMB Protokollschicht ergeben sich folgende automatisierte Fehlererkennungsmöglichkeiten gemessen an den Verifizierungsklassen aus Abschnitt 4.4.4.

Automatisierte Fehlererkennung:

- Korrektes Senden u. Empfangen auf der physikalischen Busebene

  - Nicht vollständig. Fehlerhafte Prüfsummen in Header und Nutzdaten können jedoch Hinweise auf Fehler der Busebene liefern. Nicht erkannt werden können z.B. Nachrichten, die durch einen Busfehler nicht weitergeleitet werden, da hier keine Referenzinformation vorliegt. Der Empfang eines Busteilnehmers kann nur indirekt über den Zustand der ACK Bits überprüft werden.
- Einhaltung von Protokollkonventionen
  - Konventionen für einzelne Nachrichten wie u. a. Headerstruktur, Nachrichtenlänge und Prüfsummen werden überprüft. Die Abfolge von Nachrichtendialogen, wie der korrekte Antworttyp auf einen bestimmten Anfragetypen, wird nicht überprüft.
- Verifizierung der gesendeten Nutzinformation
  - Die Prüfsummen und Nachrichtenlängen der Nutzdaten werden überprüft. Eine Auswertung der Nutzdaten erfolgt nicht.

4.5 Filter-, Such- und Triggerfunktionen

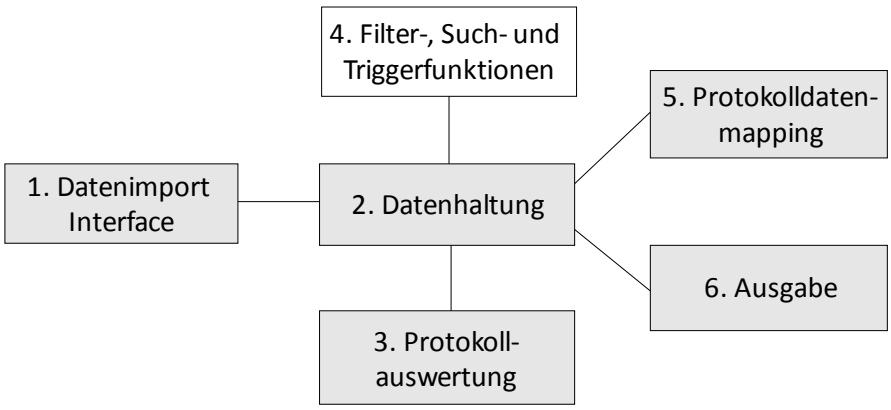

Bild 4.5.a: Einordnung Filter-, Such- und Triggerfunktionen

In diesem Kapitel werden die Protokolldaten bezogenen Filter-, Such- und Triggerfunktionen beschrieben.

Folgende Punkte werden besprochen:

- Übersicht Anforderungen
- Konzept
- Filtersyntax für einfache Filtertypen
- Komplexe Filterkonfigurationen durch Filterblöcke
- Suchfunktionen
- Triggerfunktionen

4.5.1 Übersicht Anforderungen

Aus den Anforderungen in Abschnitt 3 ergeben sich die folgenden Kriterien für die Filter-, Such-, und Triggerfunktionen:

- Die verarbeiteten bzw. angezeigten Protokolldaten sollen durch Filterfunktionen eingeschränkt werden können. Die Filterfunktionen sollen dazu die Protokolldaten mit Vergleichsoperatoren filtern können.
- Die Filterparameter sollen durch den Anwender konfigurierbar sein und Filterkonfigurationen sollen abgespeichert und geladen werden können.
- Eine Suchfunktion soll es dem Anwender ermöglichen, nach Protokolldaten und Protokollereignissen zu suchen.
- Die Triggerfunktionen sollen es ermöglichen, auf Protokolldaten und Protokollereignisse zu triggern.

4.5.2 Filterkonzept

Um ein geeignetes Konzept für die Arbeitsweise der Filter zu bestimmen, wurde die Arbeitsweise der Filter- und Suchfunktionen von Wireshark (Abschnitt 2.5.1) untersucht. Wireshark ist für Netzwerkdaten ausgelegt. Die benötigte Funktionalität ist jedoch sehr ähnlich. Bei der Untersuchung wurden folgende Konzepte festgestellt, die in ähnlicher Weise hier verwendet werden:

- Ein Filter kann von einem Filterstring bzw. einer Filtersyntax in Form einer Textzeile beschrieben werden.
  - Die Filter können schnell und einfach durch den Anwender konfiguriert werden.
  - Die Filtereinstellungen lassen sich leicht verwalten.
- Es können Protokoll abhängige Tokens definiert werden. Die Tokens stehen für einen Protokolldatenbereich oder Protokollereignis. Diese Tokens können im Filterstring verwendet werden.
  - Die Eingabe wird für den Anwender einfacher und ist weniger fehleranfällig.
  - Die Lesbarkeit der Filtereinstellungen wird erhöht.
- Die Implementierung der Suchfunktion basiert auf der Filterfunktionalität.
  - Die einheitliche Arbeitsweise der Filter- und Suchfunktion ist für den Anwender übersichtlicher.
  - Der Implementierungsaufwand der Suchfunktionalität ist erheblich geringer.

Entsprechend den Anforderungen sollen die Protokolldaten durch Vergleichsoperatoren gefiltert werden können. Um eine entsprechende Filterfunktion zu implementieren, muss festgelegt werden, welche Protokolldaten als Argumente für die Vergleichsoperationen verwendet werden können. Dazu wird folgende Einteilung vorgenommen:

- Die übertragenen Nutzdaten als Bytefolge
- Protokollereignisse wie z.B. I2C Start, Stop oder ACK/NACK
- Fehlerereignisse wie z.B. Prüfsummenfehler

Welche Daten bzw. Ereignisse für Filter verwendet werden können, ist abhängig von der Protokollschicht. Die Filter arbeiten mit den Daten der Eventstrukturen (Abschnitt 4.3), die durch die Protokollauswertung (Abschnitt 4.4) bestimmt werden. Die Funktionalität der Filter wird in einer Basisklasse implementiert und kann für jede Protokollschicht geerbt und angepasst werden.

Die Filter werden als endliche Automaten Implementiert (vgl. [hopcraft], S. 45ff]), als Eingabe dient eine Eventstruktur wie z.B. eine IPMB Nachricht. Ausgegeben werden nur die binären Zustände *Akzeptiert* und *Nicht-Akzeptiert.* Durch diesen Aufbau können die Filter eingesetzt werden, um u. a. Such- und Triggerfunktionen umzusetzen.

Der Anwender konfiguriert die Filter durch eine Filtersyntax, die im folgenden Teil beschrieben wird.

4.5.3 Filtersyntax für einfache Filtertypen

Um die Filtersyntax möglichst einfach für den Anwender zu gestallten, wird zwischen einer vereinfachten und einer erweiterten Syntax unterschieden. Mit der vereinfachten Syntax kann ein Filter mit drei Parametern bestimmt werden.

Die vereinfachte Syntax für einen Filterstring ist folgendermaßen aufgebaut:

- `Token = Protokolldatenbereich oder Protokollereignis`
- `Operation = gleich, ungleich, grösser, kleiner`
- `Parm = Byte, Zahl`

Filterstring = <Token> [<Operation>] [<Parm>]

Beispiele für einfache Filterstrings:

- Das Headerfeld CMD der IPMB Nachricht muss einen Wert von 10 besitzen
  - ipmb.cmd = 10
- Das fünfte Byte der IPMB Nachricht muss einen anderen Wert als 0xF7 besitzen
  - byte5 != 0xF7

Ein Token steht für ein Protokolldatenbereich oder Protokollereignis. Für jede Protokollschicht werden die Tokens definiert, so gibt es z.B. für IPMB das Token ‚ipmb.cmd' das für das 6. Byte im Header steht.

Mit den speziellen Tokens „byte<Zahl>", wie z.B. „byte1" oder „byte5", können die Protokolldaten Byteweise angesprochen werden.

Mit dem erweiterten Eingabeformat können auch einzelne Bits angesprochen werden. Bei den „byteX" Tokens können zwei zusätzliche Parameter, RShift und Mask, angegeben werden. Der Parameter Mask gibt eine Bytemaske für das Protokolldatenbyte an. RShift gibt an, um wie viele Bits das Protokolldatenbyte mit einem Logical Shift Right verschoben wird. Die Operationsreihenfolge als C-Code ausgedrückt:

((ByteX & *Mask*) >> *RShift)*

Das erweiterte Eingabeformat für einen Filterstring ist wie folgt:

- `Token = Protokolldatenbereich oder Protokollereignis`
- `Operation = gleich, ungleich, grösser, kleiner`
- `Parm = Byte, Zahl`
- `RShift = Zahl`
- `Mask = Byte`

Filterstring = <Token> [<Operation>] [<Parm>][<RShift>][<Mask>]

Beispiele für erweiterte Filterstrings:

- Die Bits 3:2 des ersten Bytes der IPMB Nachricht müssen dem Wert 0x01 entsprechen.
  - byte1 = 0x01 2 0x0C

4.5.4 Komplexe Filterkonfigurationen durch Filterblöcke

Um auch komplexere Filterfunktionen zu ermöglichen, können eine beliebige Anzahl von den im vorherigen Abschnitt beschriebenen Filtern zu einer Filtereinheit zusammengefasst werden. Die Filter der Filtereinheiten werden logisch UND verknüpft. Die Filtereinheiten können wiederum zu einem Filterblock zusammengefasst werden. Dabei werden die Filtereinheiten logisch ODER-verknüpft. Es können beliebig viele Filtereinheiten zu einem Filterblock zusammengefasst werden. Bild 4.5.4.a zeigt die logische Verknüpfung innerhalb eines Filterblocks.

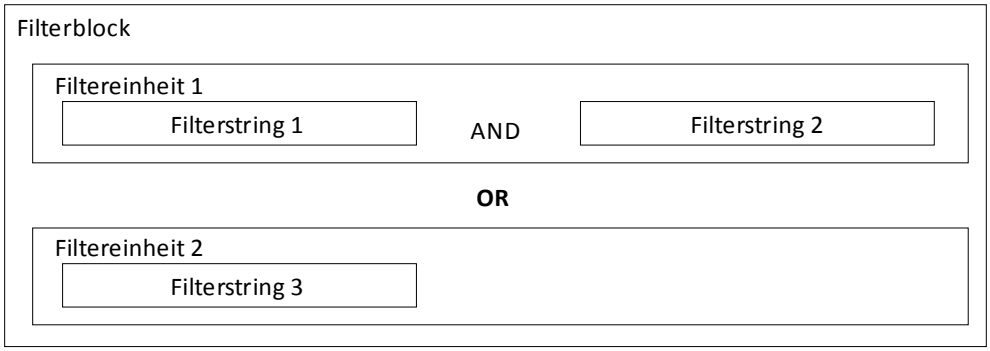

Bild 4.5.4.a: Die logische Verknüpfung innerhalb eines Filterblocks

Die Filtereinheiten und Filterblöcke werden durch eine GUI-Maske erstellt und verwaltet. Die Filterblöcke können im XML Format abspeichert und geladen werden.

4.5.5 Suchfunktionen

Die Suchfunktionen verwendet die Filterimplementierung (Abschnitt 4.5.2), um die Suchfunktionalität umzusetzen. Dadurch arbeiten die Suchfunktionen mit der gleichen Syntax wie die Filterfunktionen.

4.5.6 Triggerfunktionen

Die Triggerfunktionen arbeiten genau wie die Suchfunktionen (Abschnitt 4.5.5) mit dem Unterschied, dass ein Trigger ausgelöst wird, sobald ein Event den Trigger- bzw. Suchkriterien entspricht.

4.6 Mapping von Protokollnutzdaten

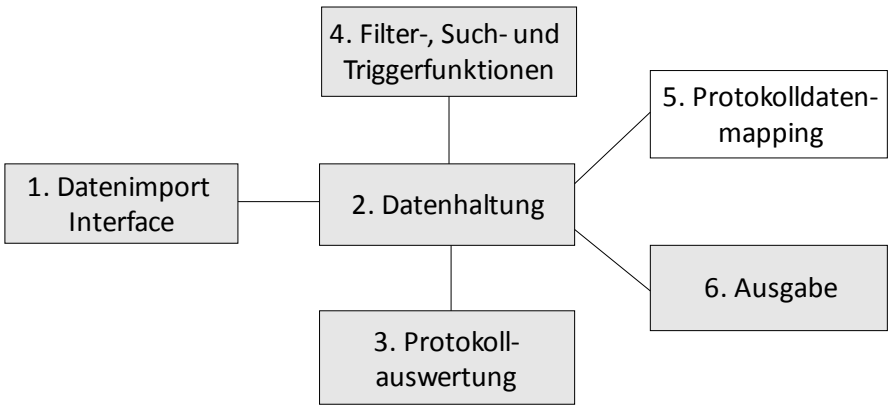

Bild 4.6.a: Einordnung Protokolldatenmapping

In diesem Kapitel wird das Verfahren vorgestellt, mit dem die Protokollnutzdaten mit konfigurierbaren Struktur- und Formatierungsinformation zur Laufzeit verknüpft werden. Mit diesen Informationen können die Protokollnutzdaten bei der Ausgabe formatiert und bei der Protokollauswertung automatisiert analysiert werden. Als Protokollnutzdaten werden hier die Protokolldaten bzw. ist die Protokollschicht bezeichnet, für die es keine reguläre Anwendungsintegration gibt.

Folgende Punkte werden besprochen:

- Übersicht Anforderungen
- Konzept
- IPMB Implementierung

4.6.1 Übersicht Anforderungen

Aus den Anforderungen in Abschnitt 3, ergeben sich eine Reihe von allgemeinen Kriterien für das Mapping der Protokollnutzdaten:

- Die in den Protokollnutzdaten enthaltenen Strukturen und Daten sollen bei der Ausgabe durch entsprechende Formatierungs- und Bezeichnungsmöglichkeiten kenntlich gemacht werden können.
- Für die automatisierte Protokollauswertung werden Informationen über den Aufbau der Protokollnutzdaten benötigt.
- Die Angaben über die Protokollnutzdaten sollen durch den Anwender leicht angepasst und verwaltet werden können.

4.6.2 Konzept

Die übertragenen Nutzdaten besitzen ein eigenes Format und können als weitere Protokollschicht betrachtet werden. Um die Nutzdaten auszuwerten, könnten diese als reguläre Protokollschicht implementiert werden. Der benötigte Aufwand eine Protokollschicht zu implementieren, ist in vielen Fällen jedoch zu groß, da z.B. die obersten Protokollschichten oft nur für ein bestimmtes Projekt eingesetzt werden. Ohne Formatierungsinformationen können diese Protokollschichten nur als Datenstrom klassifiziert und entsprechend ausgegeben werden. Es wird daher eine Lösung benötigt, die es dem Anwender erlaubt, mit möglichst wenig Aufwand bestimmte Formatinformationen für höhere Protokollschichten anzugeben.

Um eine Lösung zu finden, wird zunächst allgemein festgestellt, welche Anwendungsteile die Nutzdatenformatierung benötigen und welche Formatinformationen dafür sinnvoll sind.

Die Informationen können durch die Ausgabe (Abschnitt 4.7), die Protokollauswertung (Abschnitt 4.4) und die Filter und Suchfunktionen (Abschnitt 4.5) verwendet werden.

Für die Formatierung durch die Ausgabe:

- Strukturinformationen
  - z.B. Felder, Feldlängen
- Benennung
  - z.B. Namen bzw. Beschriftungen und Beschreibung der Felder und Daten
- Ausgabeformatierungen
  - z.B. Farbe, Schrift, Schriftgröße

Für die automatisierte Analyse durch die Protokollautomaten:

- Strukturinformationen
  - Felder, Feldlängen, Prüfsummen

Für die Filter und Suchfunktionen

- Strukturinformationen
  - z.B. Felder, Feldlängen
- Benennung
  - z.B. Namen bzw. Beschriftungen und Beschreibung der Felder und Daten

Welche Informationen benötigt und wie diese zusammenhängen, ist protokollabhängig. Das IPMB Protokoll z.B. integriert bereits grundlegende Formatierungsinformationen für die Nutzdaten. Es werden dort bereits u. a. Nachrichtenklassen vorgegeben. Bei dem I2C Protokoll gibt es für die Nutzdaten keine von vorneherein festgelegten Formatierungen. Aus diesem Grund ist eine Anpassung für jedes fest implementierte Protokoll sinnvoll, da sonst besondere Protokolleigenschaften u. U. nicht berücksichtigt werden können.

Die Nutzdateninformationen werden in einem XML Dokument gespeichert. Das XML Format bietet eine standardisierte Möglichkeit, die entsprechenden Zusammenhänge auszudrücken und die Informationen zu verwalten.

4.6.3 IPMB Implementierung

Das IPMB Protokoll (Abschnitt 2.1.4) definiert bereits eine Nachrichtenhierarchie, dabei wird nach Funktionsgruppe (NETFN) und nach Funktion (Command) gruppiert. Ein bestimmter Nachrichtentyp kann mit einem Schlüssel aus NETFN und Command bestimmt werden.

Den Funktionsgruppen (NETFN) und Funktionen (Command) können Farben für die Ausgabe zugewiesen werden. Zu jedem Nachrichtentyp können einzelne Bytes benannt und Prüfsummenpositionen angegeben werden. Die Nutzdateninformationen werden wie folgt durch eine XML Datei beschrieben.

Der Aufbau der XML-Datei für einen IPMB Nachrichtentypen:

- IPMB
  - NETFN (Id, Name, Farbe)
    - Command (Id, Name, Farbe)
      - Dataname (Index, Name)
      - CRC (Index, Startindex, Stopindex, CRCTyp)

Mit *Dataname* wird das Byte an Position *Index* mit *Name* beschriftet.

Mit *CRC* wird angegeben, dass das Byte an Position *Inde,* eine Prüfsumme aus den Bytes von *Startindex* bis *Stopindex* ist. *CRCType* gibt den CRC-Algorithmustyp an. Die CRC-Algorithmen sind im Code hinterlegt und sind dem *CRCType* zugeordnet.

Die Implementierung kann nach dem gleichen Muster erweitert werden, so können z.B. auch einzelne Bits oder Bitbereiche benannt werden.

Beispiel XML-Datei für IPMB:

```
<IPMB>
  <netFN id = "2" name = "NetFN Group1" color = “0x00FF00”>
    <command id = "5" name = "testname cmd5" color = “0xFF0000”>
      <dataname index = "1">cpufan</dataname>
      <dataname index = "2">casefan</dataname>
      <dataname index = "3">hdd1</dataname>
      <dataname index = "4">hdd2</dataname>
      <crc index = "5", startindex = "1", stopindex = "4" algo="crc8">
    </command>
</IPMB>
```

4.7 Grafische Ausgabe der Protokolldaten

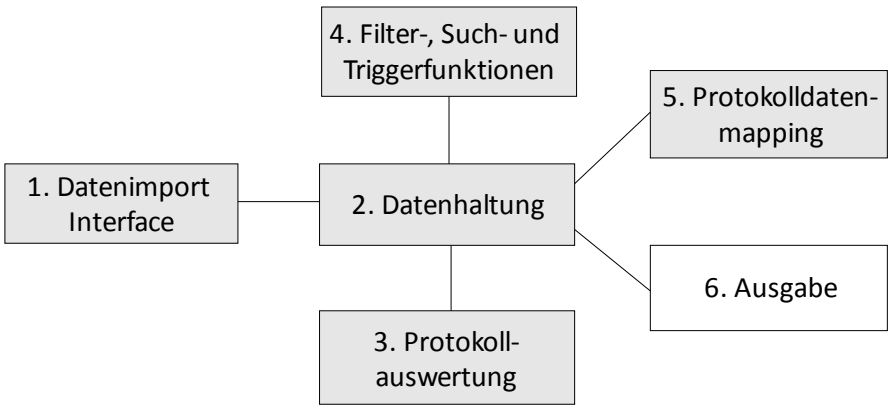

Bild 4.7.a: Einordnung Ausgabe

In diesem Abschnitt werden die Konzepte erläutert, die zur grafischen Ausgabe der Protokolldaten eingesetzt werden.

Folgende Punkte werden besprochen:

- Übersicht Anforderungen
- Ausgabefunktionalität als Bibliothek
- Vektorbasierte Ausgabeobjekte
- Ausgabepositionierung durch lokale Koordinaten
- Automatisierte Layouts für Ausgabeobjekte
- Ausgabeobjekte durch Wrapperklassen

4.7.1 Übersicht Anforderungen

Aus den Anforderungen in Abschnitt 3 ergeben sich die folgenden Kriterien, die bei der grafischen Ausgabe der Protokolldaten berücksichtigt werden:

- Die Protokolldaten sollen auf unterschiedlichen Protokollebenen auf einer gemeinsamen Zeitachse grafisch ausgegeben und navigiert werden können.
- Die Darstellung der Ereignisse soll linear zum angezeigten Zeitabschnitt erfolgen.
- Das Layout soll sich zur Laufzeit an den Darstellungsbereich anpassen können.
- Die Implementierung neuer Protokollschichten soll möglichst einfach und übersichtlich erfolgen.

4.7.2 Ausgabefunktionalität als Bibliothek

Ein wesentliches Merkmal der Anwendung ist die Erweiterbarkeit um neue Protokolle bzw. Protokollschichten. Die Implementierung neuer Protokolle soll möglichst flexibel erfolgen und möglichst einfach sein. Um die Implementierung der Ausgabe für neue Protokollschichten einfach zu gestallten, kann die Ausgabe für alle Protokollschichten gleich erflogen, wie z.B. eine Gruppierung der Protokolldaten immer nach Bytes. Eine solche einheitliche Ausgabe ist hier jedoch nicht ausreichend, da u. a. Protokoll spezifische Zusammenhänge der Protokolldaten verdeutlicht werden sollen. Um eine Implementierung zu ermöglichen, die flexibel und einfach ist, werden spezielle Funktionen für die Protokolldatenausgabe bereitgestellt. Diese Funktionen werden zu einer Bibliothek zusammengefasst. Die Konzepte dieser Bibliothek werden im Folgenden erläutert.

Ausgabeobjekte - Die Ausgabe der Protokolldaten erfolgt durch Ausgabeobjekte, diese werden durch die Bibliothek als Basisklassen definiert. Für jeden Protokolleventtyp (Abschnitt 4.3.4) wird ein Ausgabeobjekt implementiert.

4.7.3 Vektorbasierte Ausgabeobjekte

Grafische Ausgaben lassen sich im Allgemeinen in zwei Gruppen unterteilen, Rastergrafiken und Vektorgrafiken. Rastergrafiken bestehen aus einem Array von Pixeln. Ein Pixel beschreibt dabei einen Bildpunkt. Es wird keine zusätzliche Information über die Zusammensetzung der abgebildeten Objekte festgehalten. Vektorgrafiken arbeiten Objektbasiert. Diese Objekte werden durch ihre Eigenschaften beschrieben. Eine Linie z.B. wird u. a. durch Startpunkt und Endpunkt beschrieben. (vgl. [greenberg] S. 115f)

Rasterbasierte Ausgabe

- Vorteile:
  - Pixelgenaue Positionierung bei pixelbasierten Medien
  - Einheitliches Format durch Pixelarray
- Nachteile:
  - Eingeschränkte Skalierbarkeit
  - Evtl. hohe Speicheranforderung durch Pixelarray
  - Pixelbasiert

Vektorbasierte Ausgabe

- Vorteile:
  - Skalierbarkeit
  - Objektbasiert, i.d.R. geringere Speicheranforderung als Rasterformate
- Nachteile:
  - Keine genaue Pixelpositionierung/Aliasing bei Rasterung
  - Höhere Komplexität durch Objektzusammensetzung

Die Ausgabeobjekte verwenden einen vektorbasierten Ansatz. Dadurch kann die Skalierung auf die spätere Ausgabegröße durch die Bibliothek erfolgen. Die Ausgabeobjekte benötigen keine Information über die Ausgabegröße. Der Implementierungsteil für die Skalierung der Ausgabe muss nicht mehr für jedes Ausgabeobjekt geschrieben werden, wie es bei einem rasterbasierten Ansatz nötig wäre. Bild 4.7.3.a u. 4.7.3.b zeigen die Abhängigkeiten von der Ausgabegröße bei rasterbasierter- bzw. vektorbasierter Ausgabe.

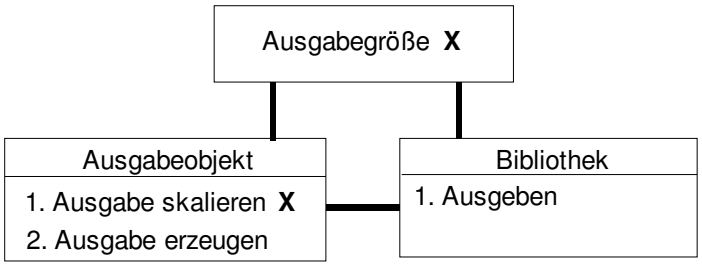

Bild 4.7.3.a: Abhängigkeiten bei rasterbasierter Ausgabe

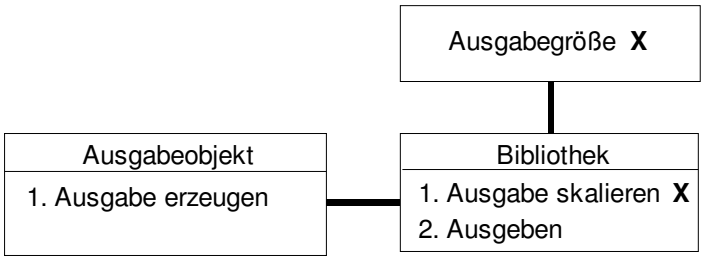

Bild 4.7.3.b: Abhängigkeiten bei vektorbasierter Ausgabe

4.7.4 Ausgabepositionierung durch lokale Koordinaten

Die Ausgabeobjekte bestehen aus primitiven Vektorobjekten wie u. a. Linien, Quadrate und Text, wie Bild 4.7.4.a und 4.7.4.b zeigen.

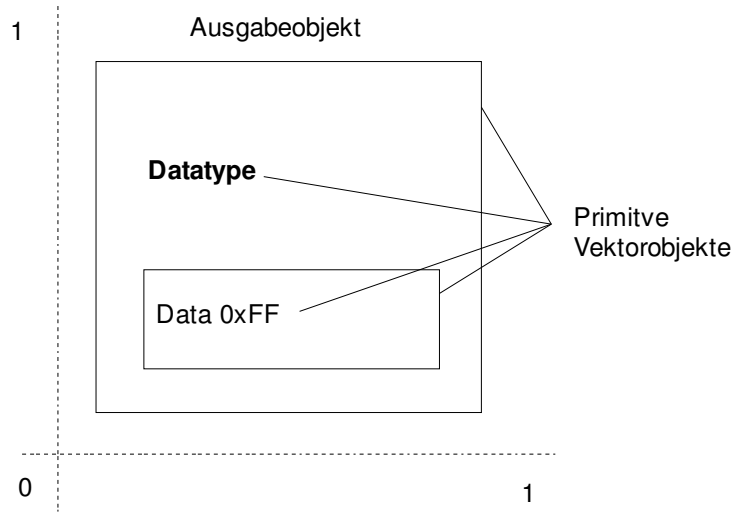

Bild 4.7.4.a: Ein Ausgabeobjekt besteht aus primitiven Vektorobjekten

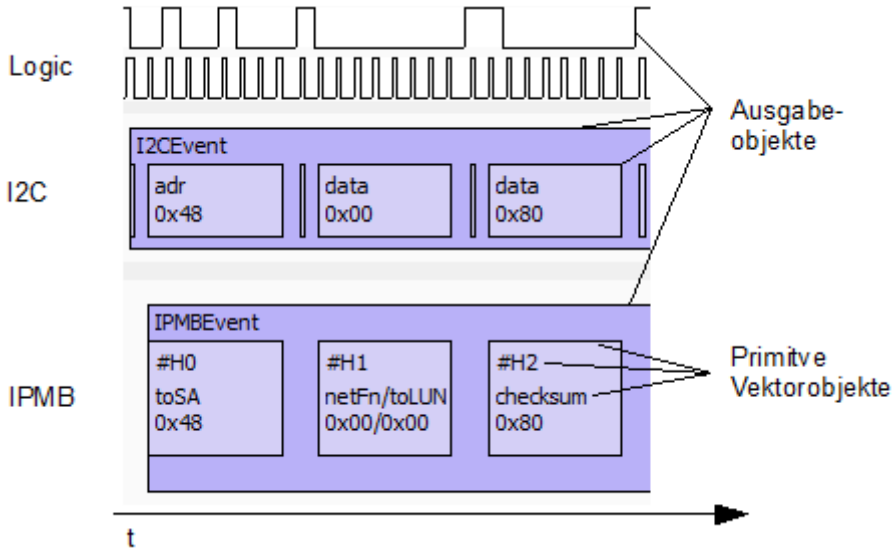

Bild 4.7.4.b: Beispielausgabe der Protokollschichten Logic, I2C und IPMB

Die Positionierung und Größe der Ausgabeobjekte auf der X- und Y-Achse wird durch die Bibliothek bestimmt. Dadurch können die Ausgabeobjekte ohne Wissen über Größe und Position implementiert werden. Dazu werden die primitiven Vektorobjekte mit statischen lokalen Koordinaten bei der Implementierung der Ausgabeobjekte positioniert, Bild 4.7.4.a. Die Positionen der primitiven Vektorobjekte werden dabei nur durch ein Verhältnis zum Ausgabeobjekt angegeben, in Form von

2D Koordinaten, z.B. ist *0;0* oben links, 0,5;0,5 mittig und *1;1* unten rechts. Durch diese Arbeitsweise bleibt die Ausgabe ausreichend flexibel für neue Protollebenen, reduziert aber den Implementierungsaufwand, so dass neue Protokollebenen i.d.R. durch wenige Codezeilen implementiert werden können.

4.7.5 Automatisierte Layouts für Ausgabeobjekte

Damit ein Konzept für das Layout bestimmt werden kann, müssen folgende Anforderungen berücksichtigt werden:

- Die Darstellung der Ereignisse soll linear zum angezeigten Zeitabschnitt erfolgen.
- Das Layout soll sich zur Laufzeit an den Darstellungsbereich anpassen können.
- Die Implementierung neuer Protokollschichten soll möglichst einfach und übersichtlich erfolgen.

Damit die im vorherigen Abschnitt beschriebenen Ausgabeobjekte bei der Ausgabe entsprechend angeordnet werden können, wie z.B. in Bild 4.7.4.b, muss bestimmt werden, wie eine solche Anordnung bei der Implementierung der Ausgabeobjekte ausgedrückt werden kann.

Entlang der X-Achse sollen die Ausgaben linear zum angezeigten Zeitabschnitt der Protokolldaten erfolgen. Die Events besitzen einen Start- und Endzeitpunkt, dadurch ist die Positionierung bzw. Länge auf der X-Achse bereits vorgegeben.

Auf der Y-Achse muss die Positionierung und Höhe der Ausgabeobjekte vorgegeben werden können. Dazu werden die Ausgaben der Ausgabeobjekte verschachtelt wie in Bild 4.7.5.a. Jede Ausgabeschicht bestimmt wie viel Platz der Untergeordneten verbleibt. Um diese Verschachtelungen bei der Implementierung auszudrücken, werden die Ausgabeobjekte zu einer Baumstruktur angeordnet wie in Bild 4.7.5.b. Ein Ausgabeobjekt bestimmt den verbleibenden Platz für die untergeordneten Ausgabeobjekte.

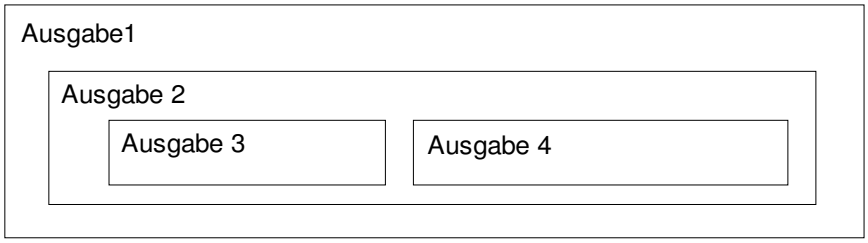

Bild 4.7.5.a: Layout der Ausgabeobjekte durch Verschachtelung

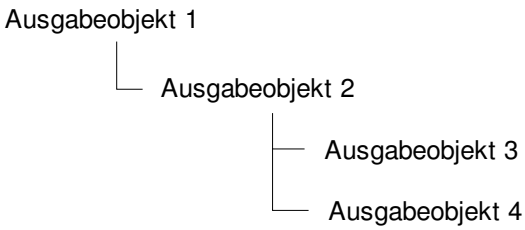

Bild 4.7.5.b: Anordnung der Ausgabeobjekte zur einer Baumstruktur

Durch diese Vorgehensweise können die Layouts sehr einfach bei der Implementierung der Ausgabeobjekte bestimmt werden, zudem lassen sich die Layouts zur Laufzeit skalieren.

4.7.6 Ausgabeobjekte durch Wrapperklassen

Die Eventobjekte aus Teil 4.3 stellen die Information bereit, die durch die Ausgabeobjekte ausgegeben werden soll. Die Eventobjekte sollen unabhängig von der grafischen Ausgabe arbeiten. Die Ausgabe soll Performant erfolgen, unabhängig der Menge an Eingangsevents. Die Funktionalität der Ausgabeklassen kann mittels Vererbung, eigenständiger Klassen oder Wrapperklassen implementiert werden:

- Bei der Vererbung wird eine neue Klasse von der bestehenden Eventklasse abgeleitet.
- Bei einer eigenständigen Klasse werden die Informationen der Eventklasse bei der Instanziierung übergeben.
- Eine Wrapperklasse kann die Funktionalität der Datenklassen dynamisch zur Laufzeit erweitern. Das Decorator Entwurfsmuster beschreibt solche Wrapperklassen. (vgl. [gamma], S. 175ff)

Vererbung

- Vorteile:
  - Ein direkter Zugriff auf die Basisklasse ist möglich.
- Nachteile:
  - Abgeleitete Objekte müssen statt Basisobjekte instanziiert werden. So muss z.B. das Grafikobjekt, das vom Eventobjekt abgeleitet ist, auch bereits durch die Datenschicht instanziiert werden.
  - Erhöhter Speicherbedarf auch wenn Funktionalität nicht benötigt wird.
  - Erhöhter Initialisierungsaufwand auch wenn Funktionalität nicht benötigt wird.

Eigenständige Klasse

- Vorteile:
  - Ein dynamisches Erweitern der Funktionalität zur Laufzeit wird ermöglicht.
  - Die Objekte stellen Interface zu Datenklassen, Ausgabeobjekte dadurch unabhängiger von den Datenklassen.
- Nachteile:

- Es entsteht Aufwand zur Laufzeit, da die Daten der Eventobjekte den Ausgabeobjekten übergeben werden müssen.

Wrapperklasse

- Vorteile:
  - Ein dynamisches Erweitern der Funktionalität zur Laufzeit wird ermöglicht.
  - Die Datenklassen bleiben eigenständig.
  - Es entsteht kein genereller Overhead für die Datenklassen.
  - Der Zugriff auf die Datenklasse erfolgt über Referenzen.
- Nachteile:
  - Die Wrapperklassen bzw. Ausgabeklassen sind Abhängig von den Eventklassen.

Für die Ausgabeobjekte wurde eine Wrapper basierte Implementierung gewählt. Die Eventklassen bleiben durch die Implementierung der Ausgabeobjektklassen unberührt und können weiterhin eigenständig eingesetzt werden. Die Ausgabeimplementierung kann ausgetauscht oder geändert werden, ohne die Eventklassen zu berühren.

Die Ausgabeobjekte, die Wrapperklassen der Eventklassen sind, werden zur Laufzeit nur für die Eventobjekte instanziiert, die auch ausgegeben werden sollen. Die Ausgabeobjekte greifen über Referenzen auf die Eventobjekte zu. Dadurch kann die Ausgabe performant erfolgen, unabhängig der Menge an Protokolldaten, da nur ein Teilbereich der Protokolldaten zur gleichen Zeit ausgegeben wird.

4.8 Kontrolle der Anforderungserfüllung

Im Folgenden wird überprüft, ob das Konzept die Anforderung aus Abschnitt 3.3.1 erfüllt. Dazu werden die Punkte der Anforderungen aufgeführt, gefolgt von den Erläuterungen in kursiver Schrift.

A. Die Messdaten bzw. Protokolldaten sollen durch eine Schnittstelle importiert werden können.

1. Es sollen Messdaten von verschiedenen Datenquellen verarbeitet werden können. Die Datenquellen können dabei u. a. dateibasiert, oder bei Hardwareanbindung, API-basiert sein. Die Datenquellen sind zum Entwurfszeitpunkt noch nicht bekannt.
    - *Die Gerätetreiberarchitektur (Abschnitt 4.2.2) ermöglicht das Anbinden von unterschiedlichen Datenquellen.*
2. Die Datenquellen liefern Daten in sample- oder ereignisbasierter Form. Die möglichen Kombinationen der Eingangsdaten werden im Abschnitt *3.3.2 Eingangsdaten* näher beschrieben.
    - *Die Gerätetreiberarchitektur (Abschnitt 4.2.2) und die entsprechenden Schnittstellen (Abschnitt 4.2.3) ermöglichen den Einsatz von samplebasierten und ereignisbasierten Daten.*
3. Samplebasierte Daten sollen mit einer Auflösung von 1-12 Bit und einer Abtastrate von bis zu 500 MHz importiert und in gleicher Qualität von der Anwendung weiterverarbeitet werden können.
    - *Die Gerätetreiberarchitektur (Abschnitt 4.2.4) ermöglicht den Import der Daten, durch die Datenhaltung (Abschnitt 4.3.2) werden die samplebasierten Daten verwaltet, durch die Timestamps (Abschnitt 4.3.6) werden die Sampledatenabschnitte mit der erforderlichen Genauigkeit verwaltet.*
4. Die Ereigniszeitpunkte von ereignisbasierten Daten sollen mit einer Mindestgenauigkeit von ± 1 ns durch die Anwendung weiterverarbeitet werden können.
    - *Durch die Datenhaltung (Abschnitt 4.3) werden die ereignisbasierten Daten verwaltet, durch die Timestamps (Abschnitt 4.3.6) werden die Ereigniszeitpunkte mit der erforderlichen Genauigkeit verwaltet.*
5. Die Messdaten können einen Zeitraum von bis zu 5 Tagen widerspiegeln.
    - *Die Timestamps der Datenhaltung (Abschnitt 4.3.6) sind so ausgelegt, dass sie einen Zeitraum von mindestens 5 Tagen zulassen. Durch das Bufferkonzept (Abschnitt 4.3.5) können beliebig große Messdatendatein verarbeitet werden.*
6. Die Anwendung soll API basierte Datenquellen steuern können.
    - *Das Steuerungsprotokoll zwischen Anwendung und Gerätetreiber (Abschnitt 4.2.6), ermöglicht das Steuern von API basierten Datenquellen.*

B. Die Protokolldaten sollen auf unterschiedlichen Protokollebenen auf einer gemeinsamen Zeitachse grafisch ausgegeben und navigiert werden können.

1. Die Darstellung der Ereignisse soll linear zum angezeigten Zeitabschnitt erfolgen.
    - *Die Ausgabefunktionen (Abschnitt 4.7.4+5), geben die Ausgabeobjekte (Abschnitt 4.7.3) linear zum angezeigten Zeitabschnitt aus.*
2. Es sollen Protokollstrukturen und Protokolldaten validiert werden können.
    - *Die Protokollauswertung (Abschnitt 4.4) ermöglicht es Protokollstrukturen und Protokolldaten der Protokollebenen zu dekodieren und zu validieren.*

3. Fehlerhafte Protokolldaten sollen ausgewiesen werden können.
    - *Bei der Protokollauswertung (Abschnitt 4.4.4) können Fehler erkannt werden, die durch die Ausgabeobjekte (Abschnitt 4.7.3) ausgegeben werden können.*
4. Das Layout soll sich zur Laufzeit an den Darstellungsbereich anpassen können.
    - *Durch die automatisierten Layouts (Abschnitt 4.7.5) wird das Layout zur Laufzeit an den Darstellungsbereich angepasst.*

C. Welche Protokolldaten ausgegeben werden, soll durch Filterfunktionen eingeschränkt werden können.

1. Die Filterfunktionen sollen die Protokolldaten mit Vergleichsoperationen filtern können.
    - *Das Filterkonzept (Abschnitt 4.5.2) ermöglicht das Filtern der Protokolldaten mit Vergleichsoperatoren.*
2. Die Filterkonfigurationen sollen im Dateiformat abgespeichert und geladen werden können.
    - *Die Filterblöcke (Abschnitt 4.5.4) ermöglichen es, Filterkonfigurationen im XML Dateiformat zu verwalten.*

D. Die Protokolldaten sollen nach Datenmustern oder Ereignissen durchsucht werden können.

- Die Protokolldaten sollen mit Vergleichsoperationen durchsucht werden können.
    - *Die Suchfunktionen (Abschnitt 4.5.5) ermöglichen das Durchsuchen der Protokolldaten mit Vergleichsoperatoren.*
- Die Suchkonfigurationen sollen im Dateiformat abgespeichert und geladen werden können.
    - *Die Suchkonfigurationen können wie die Filterblöcke (Abschnitt 4.5.4) im XML Dateiformat verwalten werden.*

E. Die Triggerfunktionen sollen Protokolldaten abhängiges Triggern ermöglichen.

1. Die Triggerbedingungen sollen durch den Einsatz von Vergleichsoperationen im Zusammenhang mit den Protokolldaten beschrieben werden können.
    - *Die Triggerfunktionen (Abschnitt 4.5.6) ermöglichen es, Triggerbedingungen durch Vergleichsoperationen festzulegen.*
2. Die Triggerkonfigurationen sollen im Dateiformat abgespeichert und geladen werden können.
    - *Die Triggerkonfigurationen können wie die Filterblöcke (Abschnitt 4.5.4) im XML Dateiformat verwalten werden.*
3. Die Triggerfunktionen sollen genutzt werden können um externe Geräte anzusteuern.
    - *Die Triggerfunktionen für externe Hardware (Abschnitt 4.2.7) ermöglichen es, externe Geräte über den Gerätetreiber (Abschnitt 4.2.6) zu steuern.*

F. Die Strukturen und Daten der Protokollnutzdaten sollen bei der Ausgabe durch entsprechende Formatierungs- und Bezeichnungsmöglichkeiten kenntlich gemacht werden können.

- *Das Protokolldatenmapping (Abschnitt 4.6) ermöglicht es, die Protokollnutzdaten bei der Ausgabe zu formatieren und entsprechend zu bezeichnen.*

1. Die Angaben über die Protokollnutzdaten sollen durch den Anwender leicht angepasst und verwaltet werden können.
    - *Das Protokolldatenmapping (Abschnitt 4.6.2) verwaltet die Formatierungs- und Bezeichnungsinformationen im XML Dateiformat, um eine Anpassung durch den Anwender zu ermöglichen.*

Zusätzliche Nichtfunktionale Anforderungen:

- Die Implementierung neuer Protokolle bzw. Protokollebenen soll möglichst einfach und übersichtlich erfolgen.
    - *Das Konzept begünstigt die Implementierung neuer Protokolle durch die Trennung der Anwendungsteile nach Protokollebenen (Abschnitt 4.3.3) und durch die speziellen Ausgabefunktionen (Abschnitt 4.7), die das Einbinden neuer Protokolle erleichtern.*

- Die Anwendung soll performant arbeiten. Das Design soll möglichst schnelle Reaktionszeiten der Ausgabe-, Filter- und Suchfunktionen ermöglichen.
    - *Das Konzept ist für ein performantes arbeiten ausgelegt. Der Einsatz von ereignisbasierten Daten (Abschnitt 4.3.2), der Aufbau der Datenstrukturen (Abschnitt 4.3.4), das Bufferkonzept (Abschnitt 4.3.5), die Arbeitsweise der Ausgabeobjekte (Abschnitt 4.7.6) und die Trennung der Filter-, Such- und Triggerfunktionen (Abschnitt 4.5) von der grafischen Ausgabe (Abschnitt 4.7) sollen schnelle Reaktionszeiten der Anwendungsfunktionen ermöglichen.*

4.9 Test der Implementierung

4.9.1 Testorganisation

Das Ziel bei diesem Test ist die Implementierung mit Produktionsdaten zu testen. Dazu wurde ein IPMB Bus verwendet, der Teil eines Fujitsu Blade Server Systems im Labor der HAW Hamburg ist. Dabei wurden die Daten vom IPMB Bussystem mit einem Hardwaremodul aufgezeichnet, das im Vorfeld dieser Arbeit im Rahmen eines studentischen Projekts entwickelt wurde [buchfink]. Die Hardware liefert die Daten vom Bussystem im ereignisbasierten Format (Abschnitt 2.3.2).

4.9.2 Testdurchführung

Die Daten vom IPMB Bussystem wurden aufgezeichnet und konnten anschließend erfolgreich grafisch analysiert werden. Das Bild 4.9.2.a zeigt die Ausgabe der Implementierung. Es werden drei Protokollebenen angezeigt, ganz oben die Logic Ebene mit den binären Zuständen der SDA und SCL Busleitung, darunter die I2C Ebene mit den übertragenen Bytes und in der Mitte die IPMB Ebene mit dem Mapping der Protokolldaten.

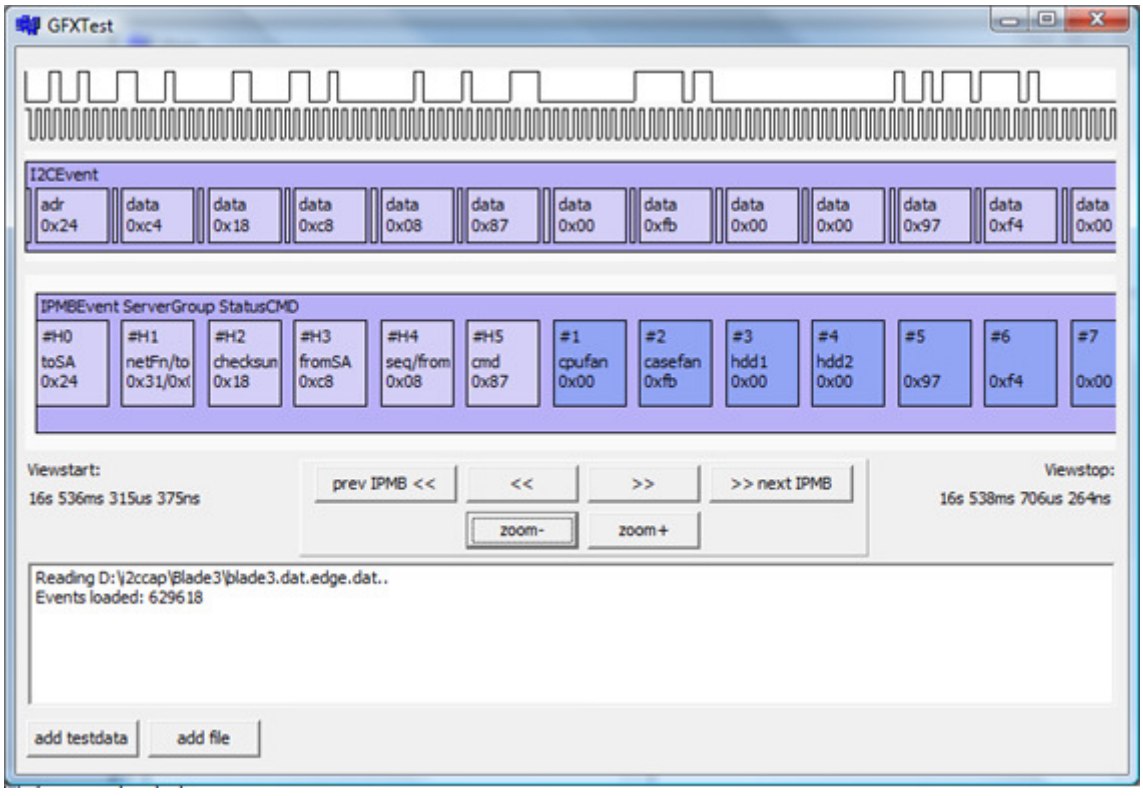

Bild 4.9.2.a: Test der Implementierung mit IPMB Produktionsdaten

5. Zusammenfassung und Ausblick

5.1 Zusammenfassung

In dieser Arbeit wird ein Konzept für ein Busanalysewerkzeug beschrieben, das es ermöglicht, die Protokollebenen eines Protokollstacks wie IPMB und I2C grafisch zu analysieren. Die Protokollebenen werden auf einer gemeinsamen Zeitachse dargestellt, um einen nahtlosen Übergang zwischen den Protokollebenen zu ermöglichen. Dadurch können die einzelnen Protokollebenen und deren Zusammenspiel untereinander sichtbar gemacht werden. Durch den Einsatz von Zoom- und Scrollfunktionen können die Protokolldaten navigiert und mit unterschiedlichen Detailstufen angezeigt werden. Das Konzept beschreibt die Funktionsbereiche wie Datenimport, Datenmodell, Protokollauswertung, Filter-, Such-, und Triggerfunktionen, Mapping der Protokollnutzdaten und die grafische Ausgabe der Protokolldaten. Das Konzept wurde für das IPMB und I2C Protokoll implementiert und erfolgreich getestet.

5.2 Ausblick

Die hier entwickelte Lösung umfasst die Basisfunktionalität eines Busanalysewerkzeugs. Diese Funktionalität kann durch weitere Arbeitspakete erweitert werden, wie u. a.:

- Ausbau der Navigations- und Präsentationsmöglichkeiten der Protokolldaten
- Erweitern der automatisierten Analysefunktionen für Protokolldaten
- Laufzeitbasiertes Konfigurationsmanagement für die Anbindung von Messdatenquellen
- Datenexportfunktionen für Protokolldaten, wie z.B. webbasierte Reports
- Einbinden von weiteren Protokollen bzw. Bussystemen

Literaturverzeichnis

[agilent_mso] Agilent MSO: http://www.home.agilent.com/agilent/product.jspx?cc=DE&lc=ger&nid=-536902766.536908416 &pageMode=OP (01.11.2009)

[asio] Steinberg ASIO SDK: http://www.steinberg.net/de/company/3rd_party_developer.html (01-11-2009)

[buchfink] Buchfink, Holger: Hardware zur Messwerterfassung für einen I2C-Bus Analyzer, Studentisches Projekt, HAW-Hamburg 2009

[digiview] DigiView DV3400: http://tech-tools.com/dv_dv3400.htm (01.11.2009)

[gamma] Gamma, Erich et al.: Design Patterns Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

[greenberg] Greenberg, Ira: Processing Creative Coding and Computational Art, Springer-Verlag New York, 2007

[hopcroft] Hopcroft, John E., Motwani, Rajeev, Ullman, Jeffrey D.: Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie, Pearson Studium, München, 2002

[i2c] NXP: UM10204 I2C-bus specification and user manual, Rev. 03, 2007

[ipmb] Intel, Hewlett-Packard, Nec, Dell: Intelligent Platform Management Bus Communications Protocol Specification v1.0, rev. 1.0, 1999

[ipmi] Intel, Hewlett-Packard, Nec, Dell: Intelligent Platform Management Interface Specification Second Generation v2.0, rev. 1.0, 2004

[logicpro] Apple Inc., Logic Pro 9: http://www.apple.com/de/logicstudio/logicpro/ (01-11-2009)

[osi] ISO International Standard 7498-1-1994(E): Information technology – Open systems Interconnection – Basic Reference Model: The Basic Model, ISO/IEC, Switzerland 1994

[pcap] pcap man page (2003): http://www.tcpdump.org/pcap3_man.html (01-11-2009)

[schnell] Schnell, Gerhard (Hrsg.), Wiedemann, Bernhard (Hrsg.): Bussysteme in der Automatisierungs- und Prozesstechnik, Friedr. Vieweg & Sohn Verlag | GWV Fachverlage GmbH, Wiesbaden, 2006

[spi] Freescale Semiconductor, Inc: M68HC11 Reference Manual, Rev. 6.1, 2007, Synchronous Serial Peripheral Interface, S. 292 ff

[tanenbaum_ds] Tanenbaum, Andrew S.: Distributed Systems - Principles and Paradigms, Prentice-Hall, Inc., New Jersey, 2002

[tanenbaum_cn] Tanenbaum, Andrew S.: Computer Networks, Pearson Education, Inc., 2003

[tds1000] Tektronix TDS1000:
http://www2.tek.com/cmswpt/psdetails.lotr?ct=PS&cs=psu&ci=13295&lc=EN (01.11.2009)

[telos] Telos I2C Studio: http://www.telos.de/i2c-studio/ (01.11.2009)

[totalphase] Totalphase Beagle I2C/SPI Protocol Analyzer:
http://www.totalphase.com/products/beagle_ism/ (01.11.2009)

[wireshark] Wireshark: http://http://www.wireshark.org/ (01.11.2009)

[wireshark_dev] Ulf Lamping , Richard Sharpe , Ed Warnicke (2004-2008). Wireshark Developer's Guide http://www.wireshark.org/docs/wsdg_html_chunked/ (01-11-2009)

[wireshark_user] Ulf Lamping (2004-2008): Wireshark User's Guide
http://www.wireshark.org/docs/wsug_html_chunked/ (01-11-2009)

Anhang

A. CD-ROM

Inhalt CD-ROM:

Arbeit im .PDF Format:

/abschlussarbeit.pdf

Quelltext der Implementierung:

/source/src

Quelltext Dokumentation:

/source/source_docs.html

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 4. Januar 2010

Ort, Datum

Unterschrift