Institut für Parallele und Verteilte Systeme
Abteilung Bildverstehen

Fakultät Informatik, Elektrotechnik und Informationstechnik

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Diplomarbeit Nr. 3247

Entwickelung eines embedded CAN-USB-Ethernet Software-Adapters für einen ARM Mikrocontroller

Mohsen Rajabi Siahboumi

Studiengang:	INFORMATIK
Prüfer:	Prof. Dr. rer. nat. habil. Paul Levi
Betreuer:	Dr. rer. nat. Hamid Reza Rajaie
begonnen am:	03.11.2011
beendet am:	03.05.2012
CR-Klassifikation:	B.4.2, C.3

Kurzfassung

Der CAN-Bus wird als Kommunikationstechnik zwischen verschiedenen Geräten verwendet. Ein Beispiel dafür ist der Bereich der sogenannten Embedded Systeme. Die Embedded Geräte sind über einen CAN-Bus miteinander verbunden und kommunizieren darüber. In manchen technischen Anwendungen ist es erstrebenswert den Datenaustausch zwischen verschiedenen CAN-Teilnehmern auf Geräte zu erweitern die in einem lokalen Netzwerk oder per über USB verbunden sind. Um diese Kommunikation zwischen den CAN-Bus Teilnehmern und Geräten über andere Medien, wie LAN oder USB, zu ermöglichen, kommt ein CAN-USB oder CAN-Ethernet Adapter in Frage. In dieser Diplomarbeit wird ein CAN-USB und ein CAN-Ethernet Softwareadapter auf einer ARM Mikrocontroller Platine entwickelt. Der Mikrocontroller wird über seinen CAN Port an den CAN-Bus angeschlossen. Mit einer zweiten Schnittstelle wird der Mikrocontroller entweder über Ethernet an ein Netzwerk oder per USB Port an einen PC angeschlossen. Das implementierte CAN-USB oder CAN-Ethernet Adapterprogramm auf dem Mikrocontroller hat die Aufgabe, die empfangenen Nachrichten in das jeweilige andere Format zu konvertieren und je nach gewünschtem Medium, CAN, Ethernet oder USB, über den entsprechenden Port weiterzuleiten.

Abstract

CAN bus is a communication technique for data exchange between different types of electronic devices, among them devices used in embedded systems. In a controller area network, the embedded devices are connected with each other via a CAN bus and communicate through it.

In a number of industrial applications, it is desirable to let the CAN nodes exchange data with other devices, supporting different types of protocols such as USB and Ethernet. CAN-USB and CAN-Ethernet adapters can be used in such applications.

In this thesis, a CAN-USB and a CAN-Ethernet software adapter for an ARM microcontroller is developed. The microcontroller is connected via its CAN port to a CAN bus and is able to communicate with other USB devices through its USB port. It is also connected to a LAN by its Ethernet interface.

The implemented CAN-USB and CAN-Ethernet adapter program on the microcontroller accepts the incoming messages, i.e., CAN, USB or Ethernet, and depending on the desired format convert them to other types and send them out through the corresponding interfaces.

Danksagung

Bei der Erstellung dieser Diplomarbeit gab es zahlreiche Hilfestellung von verschiedenen Personen.

Einige von ihnen möchte ich hier noch Namentlich erwähnen, weil sie mir besonders geholfen haben.

Prof. Paul Levi für die Ermöglichung der Diplomarbeit und die Bereitstellung eines interessanten Themas.

Dr. Hamid Reza Rajaie für die sehr gute Betreuung, die vielen hilfreichen Tipps und die anregenden Diskussionen.

Der Cilarix GmbH und den freundlichen Mitarbeitern für die Unterstützung in jeglicher Hinsicht.

Meiner Frau Yu Wang für ihre Geduld und Unterstützung, die sie mir jederzeit entgegenbrachte.

Meinen Eltern für ihre Liebe und Zustimmung in allen Lebenslagen.

Inhaltsverzeichnis

Kurzfassung ..... **i**

Danksagung ..... **ii**

Inhaltsverzeichnis ..... **iii**

Abbildungsverzeichnis ..... **vii**

Tabellenverzeichnis ..... **ix**

Abkürzungsverzeichnis ..... **x**

1 Einführung **1**

1.1 Motivation 1

1.2 Aufgabenstellung 1

1.3 Aufbau und Struktur der Arbeit 2

2 Hardware und Software Entwicklungsplattform **3**

2.1 ARM Architektur 3

2.1.1 Befehlssatz 3

2.1.2 Pipelining der ARM Architektur 4

2.2 AT91SAM9263 Mikrocontroller 4

2.3 AT91SM9263EK Evaluation Board Revision B 6

2.4 GNU-basierte Software Entwicklung auf AT91SAM 7

2.4.1 Der SAM-ICE 8

2.4.2 Der SAM-BA 8

2.4.3 Die Eclipse 9

2.4.4 Die GNU ARM-basierte-Toolchain 10

3 Die AT91SAM9263 Peripherie **11**

3.1 Das Beispiel Programm 11

3.1.1 Die Peripheriegeräte 11

3.1.2 AT91SAM9263 Evaluation Kit 12

3.2 Die Implementierung 12

3.2.1 Nutzung der Peripheriegeräte ... 12
3.2.2 Die Verwendung des Advanced-Interrupt-Controller(AIC) ... 13
3.2.3 Die Verwendung des Timer-Counter ... 14
3.2.4 Die Verwendung des Periodic-Interval-Timer ... 17
3.2.5 Die Verwendung des Parallel Input/Output Controller ... 19

4 USB ... 22
4.1 USB Grundlagen ... 22
4.1.1 Aufbau des USB-Kabels und der Signalpegel ... 23
4.1.2 USB Protokoll (Paketformat) ... 23
4.2 USB Device Port des AT91SAM9263 Board ... 25
4.2.1 Überblick ... 25
4.2.2 Block Diagramm ... 26
4.3 USB Device Zustandsdiagramm (Enumeration) ... 26
4.4 USB Framework ... 28
4.4.1 Framework Beschreibung ... 29
4.5 AT91 USB CDC Treiber Entwicklung ... 30
4.6 Implementierung der USB Test Programme auf dem PC und dem Mikrocontroller ... 31
4.6.1 Aufgabenerklärung ... 31
4.6.2 Handshake Prinzip ... 31
4.6.3 Nachrichten Paket Format ... 32
4.7 Implementierung des Test Programms auf dem Mikrocontroller ... 33
4.7.1 Software Architektur des AT91 USB Framework und des CDC Serial Driver ... 34
4.7.2 Flussdiagramm der System-Schleife des Testprogramms auf dem Mikrocontroller ... 35
4.7.3 Implementierung des Testprogramms auf dem PC ... 40

5 CAN ... 42
5.1 Grundlagen ... 42
5.1.1 Physikalische Bitübertragung ... 42

5.1.2 Übertragungsrate und Leitungslänge 43
5.1.3 Topologie 43
5.1.4 Frame Typen 44
5.1.5 Data-Frame Aufbau 44
5.2 CAN auf dem Mikrocontroller AT91SAM9263 45
5.2.1 Überblick 45
5.2.2 CAN Controller Merkmale 47
6 CAN-USB Adapter Implementierung 52
6.1 Beispielszenario 52
6.2 PCAN-USB von Peak System 53
6.3 Handshake Prinzip für das Beispielszenario 54
6.4 Nachrichten Paket Format 56
6.5 Implementierung des USB-CAN Adapters auf dem Mikrocontroller 57
6.5.1 Flussdiagramm des Main-Programms des USB-CAN Adapters 57
6.5.2 Die Evaluierung des USB-CAN Adapters 62
7 CAN-Ethernet Adapter 64
7.1 Toolchain 64
7.1.1 BitBake 64
7.1.2 OpenEmbedded 64
7.1.3 Ångström Linux für Embedded Systeme 64
7.2 Konfiguration, Erstellen und Starten des Linux Kernels 65
7.3 SocketCAN 65
7.4 Socket Programmierungsgrundlagen 66
7.4.1 Grund Funktionen der Socket Programmierung: 67
7.4.2 Ablauf der Socket Kommunikation 67
7.5 CAN-Ethernet Adapter Implementierung 68
7.6 Die Evaluierung des CAN-Ethernet Adapter Anwendungsprogramms 70
7.6.1 Die Implementierung des Testprogramms auf den PCs 70
7.6.2 Der Evaluierungsverlauf 71
8 Zusammenfassung und Ausblick 72

8.1 Zusammenfassung 72
8.2 Ausblick 72

Literaturverzeichnis 73

Abbildungsverzeichnis

Abbildung 2-1: AT91SAM9263 Block Diagramm [AT09] 5
Abbildung 2-2: AT91SM9263EK Evaluation Board Rev.B 7
Abbildung 2-3: Atmel SAM-ICE Emulator für AT91 Mikrocontrollern[GNU07] 8
Abbildung 2-4: SAM-BA Main Windows 9
Abbildung 3-1: Advanced Interrupt Controller Block Diagramm [ATM11] 13
Abbildung 3-2: Timer Counter Block Diagramm [ATM11, S.734] 15
Abbildung 3-3: Periodic Interval Timer (PIT) [ATM11, S.127] 17
Abbildung 3-4: PIO Controller Block Diagramm [ATM11] 20
Abbildung 4-1: der sternförmige Aufbau des USB mit einem Hub 22
Abbildung 4-2: PID Feld 24
Abbildung 4-3: Start-of-Frame Paket 24
Abbildung 4-4: Token Paket 24
Abbildung 4-5: Daten Paket 25
Abbildung 4-6: Handshake Paket 25
Abbildung 4-7: USB Device Diagramm [ATM11, S.860] 26
Abbildung 4-8: USB Device Zustandsdiagramm [ATM11, S. 874] 27
Abbildung 4-9: USB Framework Architektur [ATM06] 28
Abbildung 4-10: Handshake Prinzip 32
Abbildung 4-11: Nachrichten Packet Format 33
Abbildung 4-12: Architektur des Testprogramms[ATM06] 34
Abbildung 4-13: Flussdiagramm der System-Schleife des Test Programms auf dem Mikrocontroller (User Applikation) 35
Abbildung 4-14: Board Schaltplan zur Periphere Interface Device [ATM11, S. 862] 37
Abbildung 4-15: Flussdiagramm der System-Schleife des Testprogramms auf dem Host (User Applikation) 41
Abbildung 5-1: Das Prinzip eines Controller Area Network 42
Abbildung 5-2: Standard Frame 44
Abbildung 5-3: Extended-Frame 44
Abbildung 5-4: CAN Block Diagramm[ATM11, S. 650] 46
Abbildung 5-5: Flussdiagramm des Nachrichtenannahmeverfahrens [ATM11, S. 653] 47
Abbildung 5-6: Partition des CAN-Bit-Time [ATM11, S. 656] 50
Abbildung 6-1: Schema des Beispielszenarios 52
Abbildung 6-2: PCAN-USB Device [PEA11] 53
Abbildung 6-3: Hauptfenster von PCAN-View 53
Abbildung 6-4: Handshake Prinzip für das Beispielszenario 55
Abbildung 6-5: USB Data-Paket Format 56
Abbildung 6-6: Flussdiagramm des Main-Programms des USB-CAN Adapters 58

Abbildung 7-1: CAN Kommunikationsschichten für SocktCAN (links) und konventionelle Treiber (recht) [SOC] 66
Abbildung 7-2: Client/Server Beziehung der Socket API für TCP [NET] 68
Abbildung 7-3: CAN-Ethernet Adapter Anwendungsprogramm 69
Abbildung 7-4: CAN-Ethernet Adapter Testaufbau 70

Tabellenverzeichnis

Tabelle 4-1: Signalpegel [Sch06] 23
Tabelle 4-2: USB-Endpoint Beschreibung[ATM11, S. 859] 25
Tabelle 5-1: Abhängigkeit zwischen maximaler Leitungslänge und Bitrate [Mes] 43
Tabelle 5-2: Beispiel für Familien IDs 48
Tabelle 5-3: Beispiel für das Nachrichtenannahmeverfahren 48

Abkürzungsverzeichnis

AIC	Advanced Interrupt Controller
ALU	Arithmetic Logic Unit
API	Application Programming Interface (Programmierschnittstelle)
ARM	Advanced RISC Machines
CAN	Controller Area Network
CAN_MAM	CAN Message Acceptance Mask Register
CAN_MCR	CAN Control Register
CAN_MDH	CAN Data High Register
CAN_MDL	CAN Data Low Register
CAN_MFID	CAN Message Family ID Register
CAN_MID	CAN Message ID Register
CAN_MMR	CAN Message Mode Register
CAN_MSR	CAN Status Register
CCR	Channel Control Register
CDC	Communication Device Class
CLKDIS	Counter Clock Disable Command
CMR	Channel-Mode-Register
CODR	Clear Output Data Register
CPCS	RC Compare Status
CPCTRG	RC Compare Trigger Enable
CPIV	Current Periodic Interval Value
CRC	Cyclic Redundancy Check
CSMA/CR	Carrier Sense Multiple Access / Collision Resolution

CVS	Concurrent Versions System
DMA	Direct Memory Access
DPR	Dual-Port-RAM
EBI	External Bus Interface
FIFO	First In, First Out
GND	Ground
I/O	Input/Output
IDCR	Interrupt Disable Command Register
IDE	Integrated Drive Electronics
IDE	Integrated Development Environment
IECR	Interrupt Enable Command Register
IPT	Information Processing Time
ISA	Instruction Set Architecture
ISR	Interrupt Status Register
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LLCF	Low Level CAN Framework
MCK	Main-Clock-Frequency (Master Clock)
MIDE	Identifier Version
MOT	Mailbox Object Type
Msg	Message
ODR	Output Disable Register
OER	Output Enable Register
PC	Personal Computer
PCER	Peripheral Clock Enable Register

PER	PIO-Enable Register
PICNT	Periodic Interval Counter
PID	Packet ID
PIO	Parallel Input/Output
PIT	Periodic Interval Timer
PITIEN	Period Interval Timer Enabled
PIV	Periodic Interval Value
PIVR	Periodic Interval Value Register
PMC	Power Management Controller
PROP SEG	PROPagation Segment
PUDR	Pull-Up Disable-Register
PWM	Pulse Width Modulation
RC	Register C
RISC	Reduced Instruction Set Computer
SIE	Serial Interface Engine
SJW	ReSynchronization Jump Width
SODR	Set Output Data Register
SPI	Serial Peripheral Interface
SVN	Apache Subversion
SYNC	SYNChronization
SYNC SEG	SYNChronization Segment
TC	Timer Counter
TCP	Transmission Control Protocol
TQ	Time Quantum
TWI	Two Wire Interface
UDP	USB Device Port

UDP	User Datagram Protocol
USART	Universal Synchronous/Asynchronous Receiver Transmitters
USB	Universal Serial Bus
VCC	Voltage of the common collector

1 Einführung

1.1 Motivation

In unserem Leben und unserer Industrie kommen Mikrocontroller vielfach zum Einsatz. Zahlreiche Beispiele dafür sind im Haushaltsbereich (Spülmaschinen, Fernsehen, usw.) und im Industriebereich (Fahrzeugtechnologie, Automatisierungstechnik und Robotik) zu finden.

In so genannten „eingebetteten Systemen“ (Embedded Systeme) werden Mikrocontroller in einem technischen Kontext mit Hilfe verschiedener Techniken, wie zum Beispiel über einen CAN-BUS, eingebunden. Die Mikrocontroller übernehmen dabei verschiedene Aufgaben wie zum Beispiel die Steuerung, die Überwachung und die Signalverarbeitung.

Über den CAN-BUS können Informationen (Daten) zwischen verschiedenen Mikrocontrollern ausgetauscht werden. Manchmal wird bei technischen Anwendungen, eine Kommunikation über einen CAN-BUS zwischen den Mikrocontrollern und einem Computer benötigt, da ein Computer kein CAN Interface hat, um diesen Informationsaustausch zu ermöglichen. Aus diesem Grund wird die Entwicklung eines CAN-USB-Adapters oder eines CAN-Ethernet-Adapters notwendig.

1.2 Aufgabenstellung

In vielen Anwendungen benötigt man einen Datenaustausch (Kommunikation) zwischen zwei Geräten. In diesem Fall zwischen den CAN-BUS Geräten auf der einen Seite und den USB oder Ethernet Geräten auf der anderen Seite. Ein Anwendungsfall könnte z. B. das Steuerungssystem eines Fußballroboters sein. Die Systeme bestehen aus einem PC und mehreren Mikrocontrollern, die über einem CAN-BUS miteinander kommunizieren. Zwischen dem PC und dem CAN-BUS wird ein USB oder Ethernet Interface benötigt. Dafür muss ein Software-Adapter entwickelt und an eine geeignete Hardware angepasst werden.

In dieser Diplomarbeit soll so ein Software-Adapter entwickelt werden. Als Hardware dafür wird eine ARM Mikrocontroller basierte Platine verwendet. Für diese Embedded Software wird eine Bibliothek für die benötigten Sub-Module des Mikrocontrollern (z.B. CAN, USB, Ethernet, Counter-Timer, I/O etc.) entwickelt. Um diesen Adapter mit dem PC verbinden und testen zu können, wird eine Bibliothek und entsprechende Computertestprogramme entwickelt. Mit dem implementierten Mikrocontroller soll die Konvertierung von CAN Signalen zu Ethernet- oder USB Signalen realisiert werden.

1.3 Aufbau und Struktur der Arbeit

In Kapitel 2 wird zunächst der verwendete Mikrocontroller und die erforderlichen Komponenten für die Einrichtung einer Arbeitsgruppe einer GNU-basierten Entwicklungsumgebung für die Atmel AT91 Mikrocontroller-Familie vorgestellt.

In Kapitel 3 werden die verwendeten Peripheriegeräte, wie Parallel Input/Output Controller, Advanced Interrupt Controller, Timer Counter und Power Managemet Controller auf dem Mikrocontroller, erklärt. Mit Hilfe einer Beispielanwendung wird gezeigt, wie die genannten Peripheriegeräte programmiert und verwendet werden können, und welche Abhängigkeit untereinander bestehen.

In Kapitel 4 werden zunächst die USB Grundlagen und das USB Paketformat erklärt. Darauf aufbauend wird der verwendete USB Device Port auf dem Mikrocontroller mit dem angewendeten Framework und den CDC Treibern näher erläutert. Mit Hilfe eines Test Szenarios wird die genaue Funktionsweise des USB Device Port auf dem Mikrocontroller und den abhängigen Peripheriegeräten gezeigt und die Implementierung der zugehörigen Bibliotheken vorgestellt. Zur Verifizierung wird ein Testprogramm für den PC geschrieben, dieses lässt die beiden Geräte über USB kommunizieren und überwacht die korrekte Arbeitsweise bei der Übertragung.

Kapitel 5 konzentriert sich auf die CAN-Bus Grundlagen, den Aufbau der CAN Nachrichten und die Beschreibung des CAN-Controllers auf dem Mikrocontroller.

Kapitel 6 erklärt durch ein Beispielszenario die Implementierung des CAN USB Adapters. Damit der Mikrocontroller über den CAN Port mit einem anderen PC kommunizieren kann, wird die benötigte Bibliothek für den CAN-Controller auf dem Mikrocontroller und für den PC entwickelt. Mit Hilfe der entwickelten Bibliotheken für den CAN-Controller und den USB Device Port wird der CAN-USB Adapter implementiert. Schließlich wird die Evaluierung des Adapterprogramms durch das Beispiel Szenario durchgeführt.

In Kapitel 7 wird die Implementierung des Ethernet-CAN Adapters mit Hilfe der Socket Programmierung erklärt. Dazu werden die verwendete Toolchain, das installierte Embedded-Betriebssystem auf dem Mikrocontroller und das Ethernet-CAN Adapterprogramm erklärt. Analog zu Kapitel 6 wird das Adapterprogramm durch das Beispiel Szenario evaluiert.

Abgeschlossen wird diese Arbeit mit einer Zusammenfassung und einem Ausblick.

2 Hardware und Software Entwicklungsplattform

2.1 ARM Architektur

Die ARM ist ein 32 Bit Reduced-Instruction Set Computer (RISC) mit der Instruction Set Architektur (ISA), die vom britischen Unternehmen ARM Holdings entwickelt wird. Sie heißt aktuell Advanced RISC Machine während der Vorgänger Acron RISC Machine hieß.

In produzierten Stückzahlen gerechnet, ist die ARM-Architektur die am häufigsten verwendete 32 Bit Befehlssatz-Architektur [Turl02]. Das Unternehmen ARM Holdings entwickelt und verkauft die ARM-Chips mit unterschiedliche Lizenzen an Halbleiterhersteller wie Z. B. Apple, Atmel, HP und Intel.

„Ziel der Entwicklung der ARM-Chips war es, einen Core mit minimalster Verlustleistung zu entwickeln, damit das Anwendungsziel, einen optimierten Prozessor für batteriebetriebene Systeme zu entwickeln, erreicht werden konnte“ [Bei04 S. 516].

Aufgrund seiner guten Leistung und des geringen Stromverbrauchs wird der Chip unter anderem in folgenden Bereichen eingesetzt:

- Embedded Systeme
- Smartphones
- PDAs
- ISDN-Controller
- Routern

2.1.1 Befehlssatz

Die ARM-Architektur beinhaltet folgende RISC Eigenschaften:[ARM10]

- Load/Store Architektur: Befehle für Zugriff auf den Speicher
- Einheitliches 16 oder 32 mal 32-Bit-Register File
- Feste Befehlslänge von 32-Bit zur Erleichterung für Decodierung und Pipelining, auf Kosten eines verringerten Befehlssatzes.
- Befehle sind meistens in Single-Cycle-Ausführung. Das heißt, jeder Befehl wird in einem Schritt einen Vergleich ausführen und abhängig von dem Ergebnis in die eigentliche Befehlsausführung gehen.

Zusätzlich stellt die ARM-Architektur folgende Eigenschaften zur Verfügung: [ARM10]

- Kontrolle sowohl über die Arithmetisch Logische Einheit (ALU) als auch über den Schifter in den meisten datenverarbeitenden Befehlen. Dadurch kann die Verwendung der ALU und des Schifters maximiert werden.
- Auto-Inkrement und Auto-Dekrement Adressierungsmodi zu Optimierung der Programmschleifen.
- Laden und speichern multipler Befehlssätze, um den Datendurchsatz zu maximieren.
- Bedingte Ausführungen von fast allen Befehlssätzen, um den Durchsatz der Ausführung zu maximieren.

Durch diese Erweiterungen zu einer grundlegenden RISC-Architektur, können die ARM-Prozessoren eine gute Balance zwischen hoher Leistung, kompakter Code-Größe geringem Stromverbrauch und kleiner Chipfläche erreichen[ARM10].

2.1.2 Pipelining der ARM Architektur

In den ARM-Prozessoren gibt es 3- und 5-stufige Pipelines je nach Prozessortyp. Zum Beispiel hat der ARM7 Prozessor eine 3-stufige Pipeline während der ARM 9 Prozessor ein 5-stufige Pipeline besitzt.

In 3 Stufigen Pipelines wird ein Befehl in folgenden 3 Phasen ausgeführt[Fur00]:

- Fetch: Der Befehl wird in die Pipeline geladen.
- Decode: Der Befehl wird decodiert.
- Excute: Ausführung des Befehls

In ARM9 wird ein Befehl in folgenden 5 Phasen ausgeführt [ARM10]:

- Fetch: Der Befehl wird in die Pipeline geladen.
- Decode: Der Befehl wird decodiert.
- Excute: Ausführung des Befehls
- Mem: Die Operationen werden auf dem Datenspeicher ausgeführt.
- Write-Back: Das Ergebnis wird zurückgeschrieben.

2.2 AT91SAM9263 Mikrocontroller

In dieser Diplomarbeit wird als Hardware ein ARM Mikrocontroller AT91SAM9263 verwendet. Die Abbildung 2-1 zeigt das Block Diagramm des AT91SAM9263.

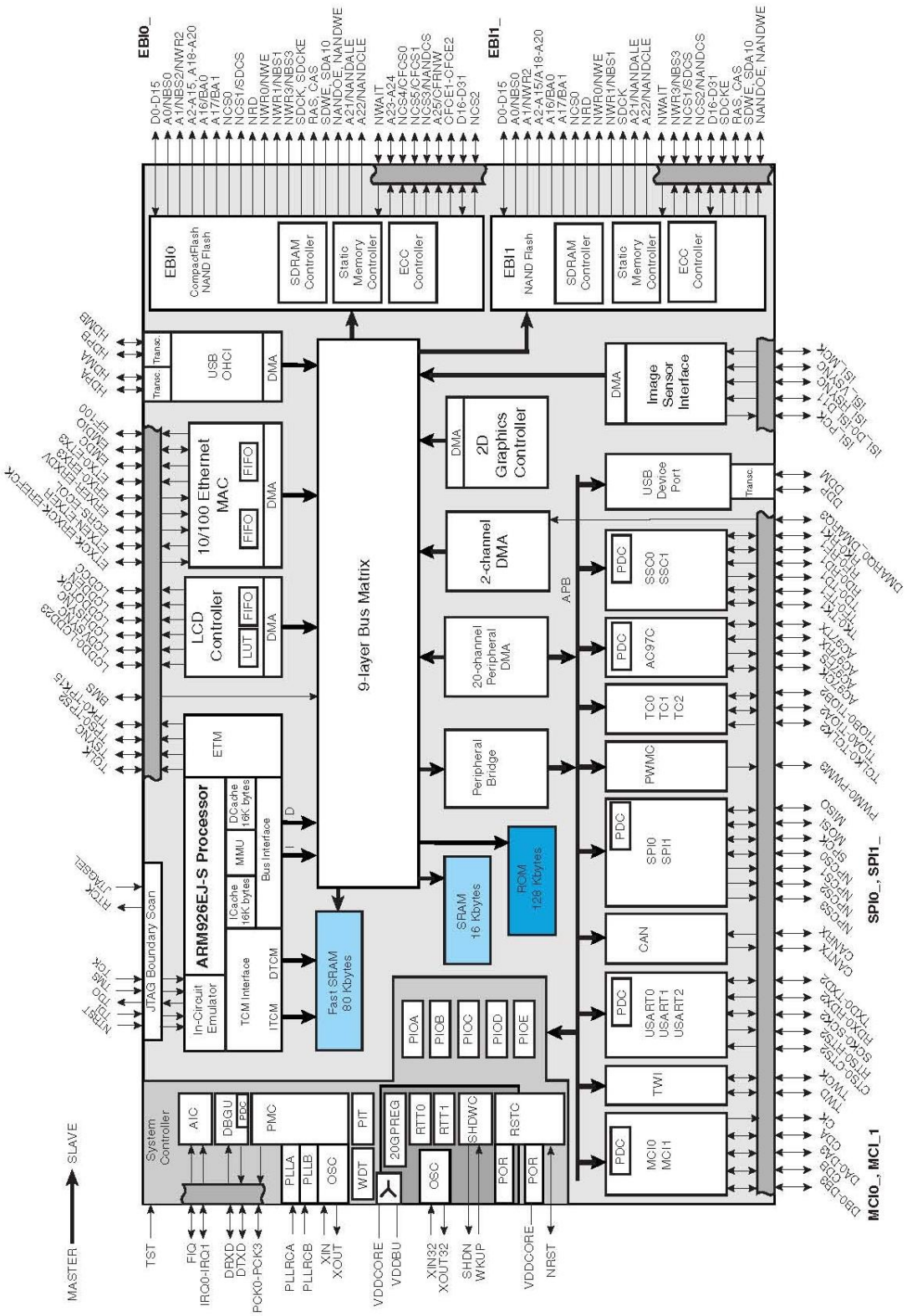

Abbildung 2-1: AT91SAM9263 Block Diagramm [AT09]

Der AT91SAM9263 ist ein 32-Bit-Mikrocontroller, der auf dem ARM926EJS-Prozessor mit einer 9-Schicht-Matrix Architektur basiert. Aus diesem Grund wird eine maximale interne Bandbreite von neun 32-Bit-Bussen erreicht. Zusätzlich verfügt er über zwei unabhängige externe Speicher Busse (EBI0 und EBI1) und einen IDE-Anschluss für Festplatten [ATM11].

Der AT91SAM9263 bettet verschiedene externe und interne Peripheriegeräte wie einen LCD-Controller mit Unterstützung eines 2D-Grafik-Controllers, ein 2-Kanal-DMA-Controller, ein Bild-Sensor-Interface, sowie ein USART, ein SPI, ein TWI, ein Timer Counter, ein PWM-Generator, eine Multimedia Card-Schnittstelle und ein CAN-Controller ein [ATM11].

2.3 AT91SM9263EK Evaluation Board Revision B

Das AT91SAM9263 ist auf einem Entwicklungsboard (AT91SM9263EK Evaluation Board Rev.B) integriert. Das Board AT91SAM9263EK ist in Abbildung 2-2 dargestellt und hat folgende Ausstattung: [ATM09]

- 64 Mbyte SDRAM Speicher
- 4 MByte PSRAM Speicher auf EBI1
- 256 Mbyte NAND-Flash-Speicher
- NOR-Flash-Speicher
- Einen 1,8 Zoll Festplatte Stecker
- TWI seriellen Speicher
- Eine USB-Gerät Port-Schnittstelle
- Zwei USB-Host-Port-Schnittstelle
- Eine RS232 serielle Schnittstelle
- Eine DBGU serielle Kommunikationsschnittstelle
- Einen serielle CAN-2.0B Kommunikation-Port
- Eine JTAG / ICE Debug-Schnittstelle
- Ethernet-100-base-TX mit drei Status LED
- AC97 Audio
- Ein 3,5 Zoll VGA-TFT-LCD-Modul mit Touchscreen
- Zwei Benutzereingabe Drucktasten
- Eine Wakeup-Taste
- Ein Reset-Taste
- Ein Data Flash /SD/SDIO/MMC-Kartenslot
- Einen Lithium Batterie-Halter

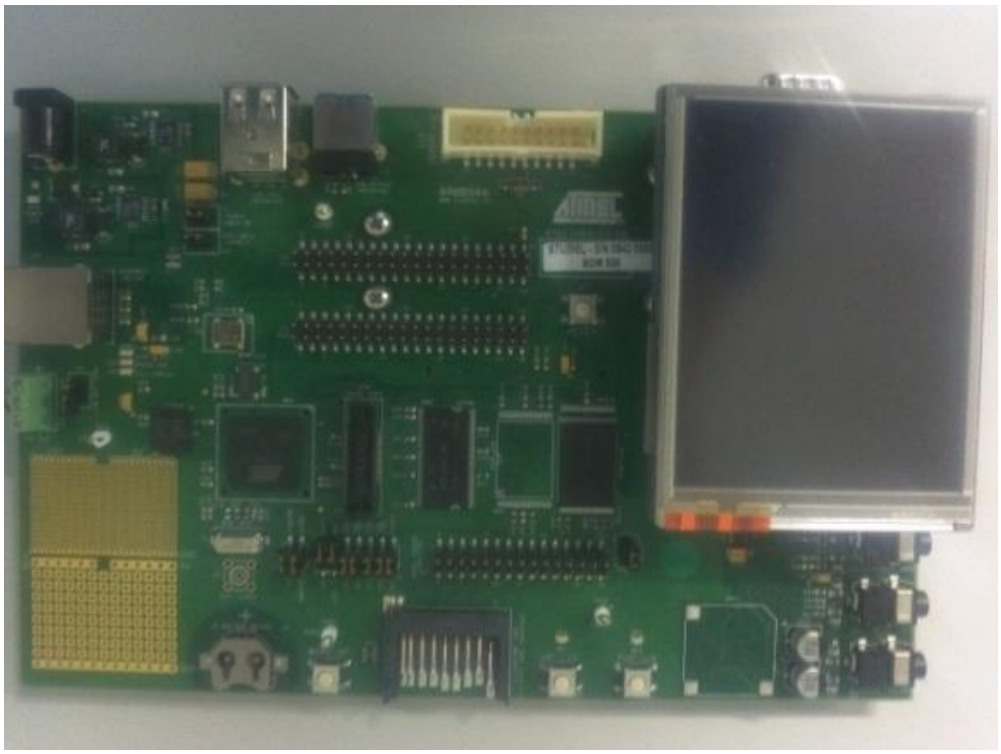

Abbildung 2-2: AT91SM9263EK Evaluation Board Rev.B

2.4 GNU-basierte Software Entwicklung auf AT91SAM

Bei Entwicklungen in der ARM®-Welt werden als häufigste Lösung kommerzielle Pakete, wie z. B. IAR®EWARM oder ARM®RealView® verwendet. Diese integrieren alle Tools die für eine Embedded-Software-Programmierung erforderlich sind und werden gut unterstützt. Jedoch haben Open-Source-Lösungen wie die GNU-Toolchain im Moment viel an Wettbewerbsfähigkeit gewonnen. Im Gegensatz zu kommerziellen Angeboten fehlt ihnen aber ein vollwertiges und einfach zu bedienendes Gesamtpaket für den Endanwender [GNU07].

In folgender Aufzählung werden die erforderlichen Komponenten für die Einrichtung einer Arbeitsgruppe mit einer GNU-basierte Umgebung auf der Atmel AT91Mikrocontroller-Familie vorgestellt. Diese bieten folgende Features:[GNU07]

- Erstellung und Bau von Projekten mit der GNU-Compiler Toolchain
- Fehlerbehandlung mit dem Atmel SAM-ICE-Emulator
- Speicher-Programmierung mit SAM-BA
- Integration dieser Aufgaben in einer Eclipse basierten Entwicklungsumgebung

Verschiedene Pakete müssen kombiniert werden um ein voll ausgestattetes System, das mit kommerziellen Produkten vergleichbar ist, zu erhalten. In diesem Abschnitt werden die einzelnen Komponenten der GNU-Umgebung vorgestellt.

2.4.1 Der SAM-ICE

Der SAM-ICE Adapter (siehe Abbildung 2-3) ermöglicht den Zugriff auf das JTAG Interface und den DBGU-Port des Atmel AT91-Mikrocontroller mittels eines USB Ports. Der SAM-ICE ermöglicht es, dass sowohl die Software bequem auf das Board aufgespielt werden kann, wie auch das die aufgespielte Software debugged werden kann.

Abbildung 2-3: Atmel SAM-ICE Emulator für AT91 Mikrocontrollern[GNU07]

Eine ausführliche Installationsanleitung für SAM-ICE Atmel finden Sie in [SAM08]. Das SAM-ICE hat folgende Features:[SAM08]

- Unterstützung aller Atmel AT91 ARM7 ™ / ARM9 ™ basierten Mikrocontroller
- Kein Netzteil erforderlich, da er über USB mit Strom versorgt wird.
- Maximale Geschwindigkeit 8 MHz JTAG
- Automatische Erkennung der Geschwindigkeit
- Unterstützung für mehrere Geräte
- Vollständige Plug und Play Unterstützung
- Standard 20-Pin-JTAG-Anschluss
- Unterstützung für anpassungsfähige Taktung

2.4.2 Der SAM-BA

Die SAM-Boot Assistent (SAM-BA) Software bietet eine einfache Möglichkeit zur Programmierung verschiedener Atmel AT91 ARM auf Thumb basierten Geräten. Die Software läuft unter Windows 2000 und Windows XP. Die Kommunikation mit dem Zielgerät kann über eine RS232-, einer USB-Verbindung oder direkt über ein SAM-ICE (JTAG link) erfolgen [SAM06]. Eine ausführliche Installationsanleitung für SAM-BA ist in [GNU07] zu finden.

Die Abbildung 2-4 zeigt das Hauptfenster des SAM-BA Assistenten.

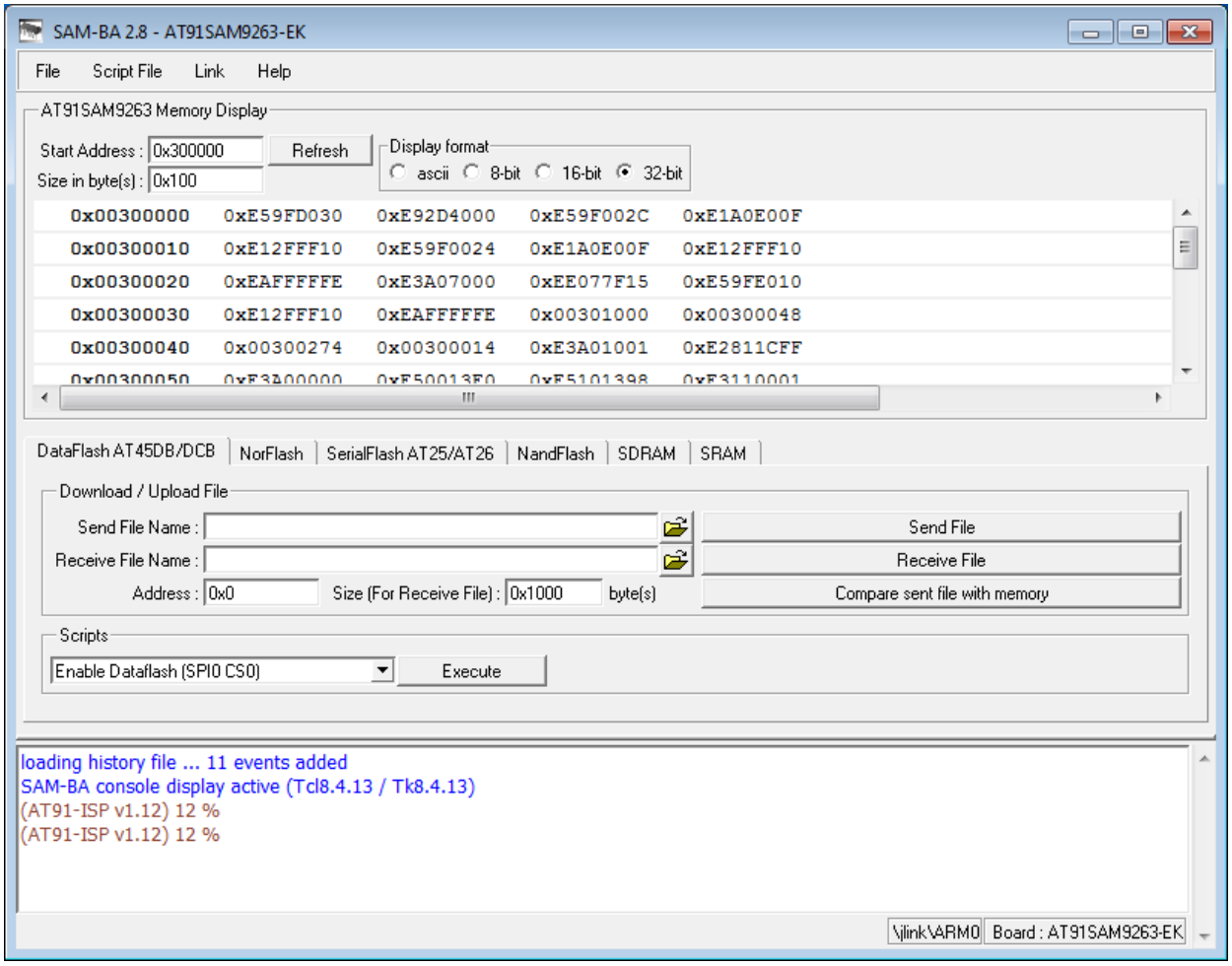

Abbildung 2-4: SAM-BA Main Windows

2.4.3 Die Eclipse

Eclipse ist ein Open-Source-Integrated Development Environment (IDE), die ursprünglich auf die Java-Programmierung ausgerichtet war. Es wurden mehrere Plug-Ins entwickelt um eine C/C++-Programmierung und das Debugging sowie die Versionsverwaltung (integrierte Versionskontrollsysteme wie SVN und CVS) zu ermöglichen. Durch die vorhandene Modularität sind die Plug-Ins einfach zu integrieren. Die Installationsanleitung für Eclipse finden Sie in [GNU07].

Das Eclipse CDT ist ein GNU-ARM Plug-In für Eclipse. Das CDT-Projekt bietet eine voll funktionsfähige C und C++ integrierte Entwicklungsumgebung auf Basis der Eclipse-Plattform, Dies ermöglich die Erstellung und Verwaltung von ARM-Projekten mit dem Eclipse-Framework. Es läuft unter Windows, Linux und Mac OS X [Ecl].

2.4.4 Die GNU ARM-basierte-Toolchain

Eine Cross-Compiler Toolchain wird immer für die Entwicklung von Mikrocontroller-Firmware die auf einem PC-Host stattfindet benötigt. Der Cross-Compiler wird für eine Zielplattform spezialisiert. Er übersetzt den C-Source-Code auf der Host-Plattform (Windows oder Linux PC) für den Maschinen-Code der Zielplattform (z.B. für das AT91SAM Board).

Im Internet stehen mehrere solcher Toolchains für Windows oder Linux zu Verfügung. Beispiele dafür können unter anderem YAGARTO [Gnu07], GNU ARM-Toolchain [GNU11] oder CodeSourcery G++ Lite for ARM EABI sein. In dieser Diplomarbeit wurde der CodeSourcery G++ Lite for ARM EABI als Entwicklungsumgebung gewählt.

Der CodeSourcery ist eine GNU GCC Toolchain für die ARM-Prozessoren von Mentor Graphics®, die auf Windows, GNU/Linux und Mac OS X laufen. Die Firma Mentor Graphics bietet diese Toolchain zusätzlich zum kommerziellen Angebot als CodeSourcery G + + Lite für ARM EABI als kostenlose Version an. Dies ist eine perfekte Ergänzung für das GNU-ARM Eclipse Plug-In [Men]. Die SourceryCode Bench Lite Edition beinhaltet folgende Komponente:

- GNU C und C + + Compiler
- GNU Assembler und Linker
- C und C + +-Laufzeitbibliotheken
- GNU-Debugger

3 Die AT91SAM9263 Peripherie

In diesem Kapitel wird mit Hilfe eines Beispiels (eine Basic-Anwendung) gezeigt, wie die meist genutzten Peripheriegeräte in den folgenden Kapiteln programmiert und verwendet werden.

3.1 Das Beispiel Programm

Das Beispielprogramm veranlasst zwei LEDs auf der Platine AT91SAM9263EK mit einer festen Rate zu blinken. Diese Frequenz für die erste LED wird durch die Verwendung eines Timers erzeugt. Bei der zweiten LED wird eine Wait-Funktion auf der Grundlage eines 1ms Tick vorangestellt. Das Blinken kann mithilfe von zwei Tasten (eine für jede LED) angehalten werden [Get07].

Obwohl diese Software ziemlich einfach aussieht, verwendet sie mehrere Peripheriegeräte und ist somit ein guter Ausgangspunkt um die Eigenschaften des AT91SAM9263 Mikrocontroller darzustellen.

3.1.1 Die Peripheriegeräte

Damit die LEDs wie in Kapitel 3.1 beschrieben funktionieren, verwendet die Software folgende Peripheriegeräte: [Get07]

- Parallel Input/Output (PIO) Controller
- Timer Counter (TC)
- Periodic Interval Timer (PIT)
- Advanced Interrupt Controller (AIC)

Die LEDs und die Tasten auf der Platine sind mit den Standard Input/Output-Pins des Chips verbunden die durch einen PIO-Controller verwaltet werden. Eine Veränderung des Status der Pins ermöglicht die Steuerung eines Interrupts. Die Tasten werden so konfiguriert, dass sie dieses Verhalten zeigen [Get07].

Der TC und PIT werden verwendet um zwei Mal eine Takt-Base zu erzeugen. Dadurch wird sichergestellt, dass die LEDs mit gleichmäßiger Rate blinken. Beide LEDs werden im Interrupt-Modus betrieben. Das TC löst dabei ein Interrupt mit einer festen Rate aus, so dass die LEDs jedes Mal umschalten. Die PIT löst in jeder Millisekunde einen Interrupt aus und inkrementiert eine Variable. Die Wait-Funktion überwacht dabei diese Variable um die genaue Verzögerungszeit zum Umschalten der zweiten LED sicherzustellen. Diese Wait-Funktion kann dabei wie eine Sleep-Funktion in C++ für andere Anwendungen benutzen werden.

Zusätzlich ist ein AIC erforderlich um die Interrupts zu verwalten. Er ermöglicht die Konfiguration eines separaten Vektors für jede Quelle. Es werden drei verschiedene Funktionen verwendet um die PIO, TC und PIT Interrupts auszuführen.

3.1.2 AT91SAM9263 Evaluation Kit

3.1.2.1 Die Tasten

Das AT91SAM9263 Evaluation Kit verfügt über zwei Tasten, die mit den Pins PC4 und PC5 (siehe in [ATM09, S.3-9]) verbunden sind. Nach der Aktivierung erzeugen sie einen logischen Low-Pegel auf dem entsprechenden PIO Pin.

3.1.2.2 Die LEDs

Es gibt zwei grüne Allzweck-LEDs auf dem AT91SAM9263-EK Board. Sie sind mit den Pins PB8 und PC29 (siehe in [ATM09, S.3-8, 3-9]) verschaltet. Das setzen eines logischen Low-Pegel auf dem entsprechenden Pin schaltet die LED ein. Umgekehrt bewirkt das setzen eines logischen High-Pegel auf den Pin das ausschalten der LED.

3.2 Die Implementierung

Wie bereits erwähnt, erfordert das in Kapitel 3.1 definierte Beispiel die Verwendung von mehreren Peripheriegeräten. Außerdem muss auch der notwendige Code zur Inbetriebnahme des Mikrocontrollers implementiert und bereitgestellt werden. Diese beiden Aspekte werden detailliert in den folgenden Abschnitten beschrieben [Get07].

3.2.1 Nutzung der Peripheriegeräte

3.2.1.1 Die Initialisierung

Die meisten Peripheriegeräte werden durch drei Aktionen initialisiert:

1. Die Aktivierung der Peripherie-Clock in der PMC
2. Aktivieren der Steuerung der Peripheriegeräte PIO Pins
3. Konfigurieren der Interrupt-Quellen der Peripheriegeräte in der AIC

Die meisten Peripheriegeräte werden nicht mit einer Standard Rate getaktet, dies macht es möglich den Stromverbrauch des Systems beim Start zu verringern. Es erfordert jedoch, dass der Programmierer explizit die Peripherie Clock einschaltet. Dies wird in dem Power Management Controller (PMC) gemacht. Eine Ausnahme gibt es für den System Controller (der aus mehreren verschiedenen Controllern besteht), da er von Anfang an und zu jeder Zeit getaktet wird [Get07].

Für Peripheriegeräte, die an einem oder mehreren Pins des Chips als externe Input/Output Geräteangeschlossen sind, muss der parallele Input/Output Controller zuerst konfiguriert werden.

Nachdem der Interrupt von einem Peripheriegerät erzeugt wurde, muss die Quelle korrekt im Advanced Interrupt Controller konfiguriert werden. Für weitere Informationen siehe Kapitel 3.2.2.3.

3.2.2 Die Verwendung des Advanced-Interrupt-Controller(AIC)

3.2.2.1 Das Grundprinzip

Der AIC (Abbildung 3-1) verwaltet alle internen und externen Interrupts des Systems. Er ermöglicht die Definition von einem Handler für jede Interrupt-Quelle. Eine Funktion wird dabei aufgerufen, wenn das entsprechende Ereignis eintritt [Get07]. Interrupts können auch einzeln aktiviert oder ausgeblendet werden. Zusätzlich können sie bis zu 8 verschiedene Prioritätsstufen haben. Detaillierte Informationen über den AIC und seine Registern findet man in [ATM11, S.377].

In der Beispiel-Software ist die Verwendung des AIC erforderlich, da mehrere Interrupt-Quellen vorhanden sind (siehe Kapitel 3.1.1).

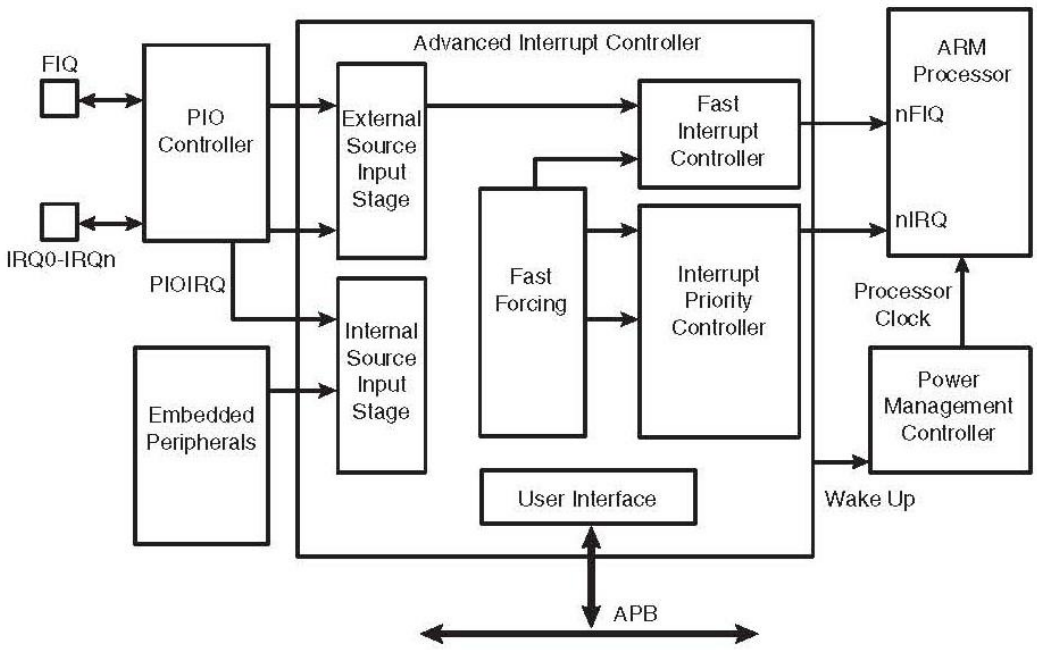

Abbildung 3-1: Advanced Interrupt Controller Block Diagramm [ATM11]

3.2.2.2 Die Initialisierung

Anders als bei den meisten anderen Peripheriegeräten wird der AIC immer getaktet und kann nicht abgeschaltet werden. Daher besteht keine Notwendigkeit den Peripherie Takt (Clock) in der PMC zu aktivieren.

Die einzige zwingende Aktion die an dieser Stelle auszuführen ist, ist das deaktivieren und löschen aller Interrupts. Dies wird mit den folgenden beiden Anweisungen erledigt:[Get07]

```
// Disable all interrupts (Deaktieren)

AT91C_BASE_AIC->AIC_IDCR = 0xFFFFFFFF;

// Clear all interrupts (Löschen)

AT91C_BASE_AIC->AIC_ICCR = 0xFFFFFFFF;
```

3.2.2.3 Das Konfigurieren eines Interrupt

Die Konfiguration einer Interrupt Quelle erfolgt in folgenden fünf Schritten:

1. Deaktivierung des Interrupts.
2. Konfiguration des Interrupt Source Mode Registers.
3. Konfiguration des Interrupt Source Vektor Registers.
4. Aktivierung des Interrupt auf der peripherie Ebene.
5. Aktivierung des Interrupt bei der AIC

Der erste Schritt besteht darin die Interrupt-Quelle zu deaktivieren, damit kein Interrupt zur gleichen Zeit ausgelöst wird. Denn das Lesen eines Mode- oder Vektor-Register kann in ein unvorhersehbares Verhalten des Systems münden. Die Interrupt Disable Command Register (IDCR) der AIC muss mit der Interrupt Quellen-ID beschrieben werden, um Interrupts des Peripheriegeräts zu maskieren [Get07]. Dabei ist das entsprechende Datenblatt der Peripherie IDs zu beachten (siehe in [ATM11, S.33]).

Es gibt zwei Parameter die in dem Source-Modus-Register festzulegen sind. Zum einen die Interrupt-Priorität und zum anderen der Trigger-Modus. Der Interrupt kann eine Priorität zwischen 0 (niedrigste) und 7 (höchste) haben. Interne Interrupts (die von Peripheriegeräten kommen) müssen immer als Level-Sensitive konfiguriert werden. Die externen Interrupts werden abhängig davon wie sie auf dem Chip verdrahtet sind konfiguriert. Das Source Vector Register enthält die Adresse der Handler-Funktion für den Interrupt [Get07].

Am Ende kann die Interrupt-Quelle aktiviert werden. In der Regel wird das sowohl in einem Mode-Register wie auch in dem Interrupt Enable Command Register (IECR) des AIC gemacht. Zu diesem Zeitpunkt sind die Interrupts vollständig konfiguriert und betriebsbereit.

3.2.3 Die Verwendung des Timer-Counter

3.2.3.1 Das Grundprinzip

Der Timer Counter (TC) enthält drei identische 16-Bit-Timer Counter-Kanäle (Abbildung 3-2). Die Kanäle können unabhängig voneinander programmiert werden, damit eine große Auswahl an Funktionen wie Frequenzmessung, Ereigniszählung, Impulserzeugung, Delay-Timing und Puls-Width-Modulation (PWM) durchgeführt werden kann [ATM11].

Jeder Kanal hat drei externe Takteingänge, fünf interne Takteingängen und zwei Mehrzweck-Input/Output-Signale die vom Benutzer frei konfiguriert werden können. Jeder Kanal treibt ein internes Interrupt-Signal. Dieses wird so programmiert dass Prozessor-Interrupts generiert werden können.

Der Timer Counter Block hat zwei globale Register, die auf allen drei Kanälen TC bereitstellen. Das Block-Controller-Register erlaubt dass die drei Kanäle gleichzeitig mit dem gleichen Befehl gestartet werden. Ebenso definiert das Block-Mode-Register den externen Takteingang für jeden Kanal, so dass sie verkettet werden können. [ATM11] Für weitere Information siehe [ATM11, S.733].

In diesem Abschnitt wird ein einziger Timer Counter Kanal verwendet, um eine feste Zeitverzögerung bereitzustellen. Ein Interrupt wird jedes Mal erzeugt, wenn der Timer abläuft. Der dadurch erzeugte Interrupt ruft einen Interrupt-Handler auf. Aufgrund dessen die LEDs mit einer festen Rate blinken.

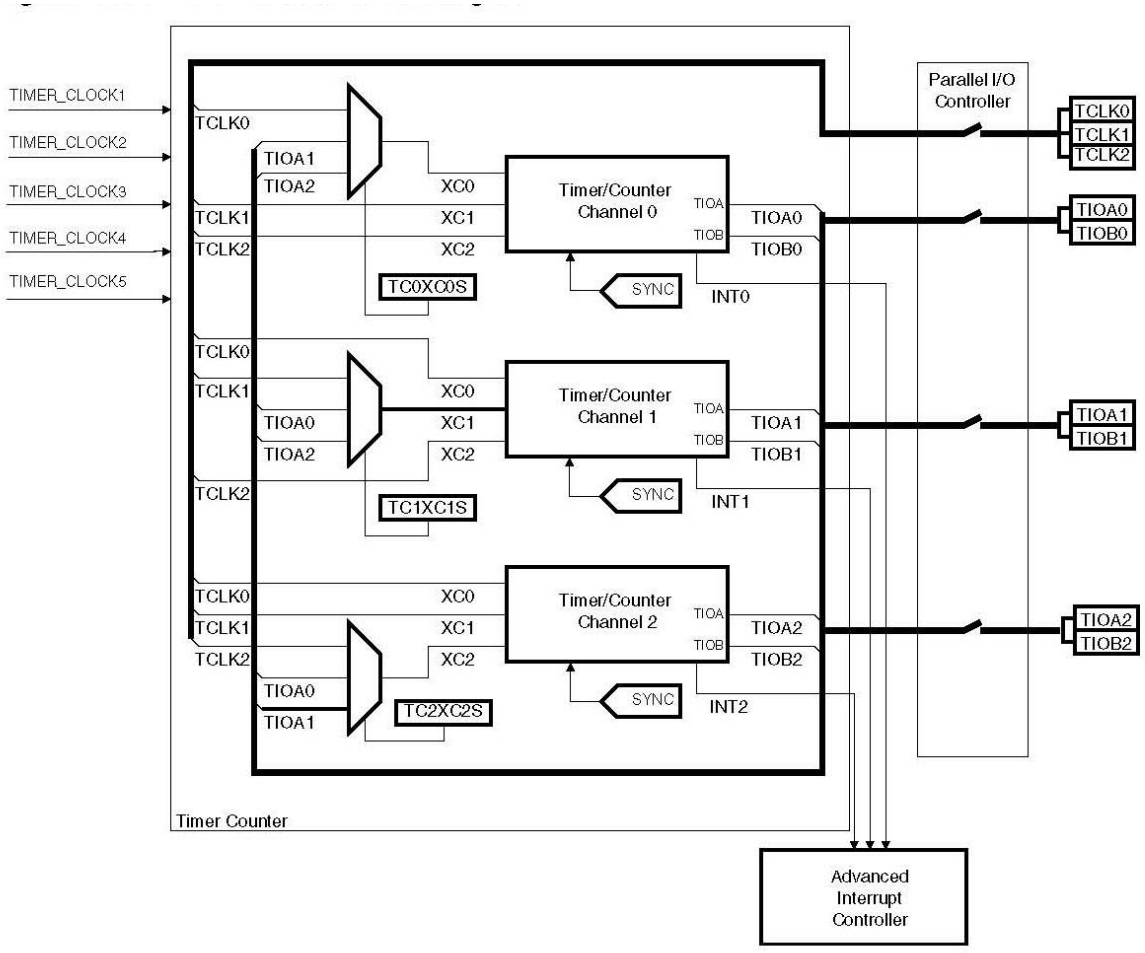

Abbildung 3-2: Timer Counter Block Diagramm [ATM11, S.734]

3.2.3.2 Die Initialisierung

Der erste Schritt bei der Initialisierung eines Peripheriegerätes ist die Aktivierung seines Taktes (Clock). Dafür muss die Timer-Counter-ID (siehe in [ATM11, S.33]) in die Peripherie Clock Enable Register (PCER) des Power Management Controllers (PMC) geschrieben werden.

Für die Aktivierung des TC wird der CLKDIS Bit in dem entsprechenden Channel Control Register (CCR) (z.B. Timer-Channel 0) gesetzt.

TC Kanäle können in zwei verschiedenen Modi arbeiten: [Get07]

- Capture-Mode: dieser Modus wird normalerweise für die Messung von Eingangssignalen verwendet.
- Waveforme-Mode: dieser Modus ermöglicht die Erzeugung von Impulsen.

Der nächste Schritt wird benötigt, um das Channel Mode Register (CMR) zu konfigurieren. In diesem Beispielprogramm, ist der Zweck der TC, einen Interrupt mit einer festen Rate zu erzeugen. Eine solche Operation ist sowohl im Capture- als auch im Waveform-Modus möglich.

Beim Einstellen des CPCTRG Bits des CMR wird der Timer zurückgesetzt. Der Timer wird dabei jedes Mal neu gestartet, wenn sein Zähler den Wert des programmierbaren Registers C (RC) in der TC erreicht hat. Das Erzeugen einer bestimmten Verzögerung erfolgt somit, indem der richtige Wert für das RC berechnet wird. Es ist auch möglich, zwischen verschiedenen Input-Takten für den Kanal auszuwählen. Das macht in der Praxis den vorangestellten MCK möglich. Der Timer arbeitet mit einer Auflösung von 16 Bit, deshalb kann man einen höheren Faktor als Voreinstellung verwenden, um eine größere Verzögerung einzustellen [Get07].

Betrachten wir folgendes Beispiel: Der Timer muss eine 500ms (0.500 Sekunde) Verzögerung mit einer 48MHz Main Clock Frequency (MCK) generieren. Der RC muss dann die gleiche Anzahl von Taktzyklen, die während der Verzögerungsperiode erzeugt werden, haben. Im Folgenden werden die Ergebnisse mit verschiedenen Vorskalierungsfaktoren dargestellt:[Get07]

$$Clock = \frac{MCK}{2}\ , RC = 24000000 \times 0.500s = 12000000$$

$$Clock = \frac{MCK}{8}\ , RC = 6000000 \times 0.500s = 3000000$$

$$Clock = \frac{MCK}{128}\ , RC = 375000 \times 0.500s = 187500$$

$$Clock = \frac{MCK}{1024}\ , RC = 46875 \times 0.500s = 23437.5$$

$$Clock = 32\ kH\ , RC = 32768 \times 0.500s = 16834$$

Der maximale Wert für ein RC ist 65535. Aus diesen Ergebnissen ist deutlich ersichtlich, dass um eine lange Verzögerungen (etwa 1s) zu erzeugen, die Verwendung eines MCK geteilt durch 1024 oder ein langsamerer interner Takt erforderlich ist.

Im letzten Schritt der Initialisierung muss der Interrupt konfiguriert werden. Sobald der Zähler den Wert in RC erreicht, wird ein Interrupt erzeugt[Get07]. Für weitere Informationen zur Konfiguration der Interrupts im AIC siehe Kapitel 3.2.3.3.

3.2.3.3 Der Interrupt-Handler

Die erste Aktion des Interrupt Handler ist das Acknowledge-Bit im Status-Register für den entsprechenden Timer-Counter zu löschen. Der Rest des Interrupt-Handler schaltet einfach den Zustand (an oder aus) einer blinkenden LED.

3.2.4 Die Verwendung des Periodic-Interval-Timer

3.2.4.1 Das Anwendungsziel

Das primäre Ziel des Peripherie Intervall Timers (PIT) ist es, periodische Interrupts zu generieren. Dies wird oft benutzt, um den Basis Tick für das Betriebssystem bereitzustellen. Die PIT verwendet MCK geteilt durch 16 als Eingabe Takt (clock) sowie einen 20-Bit-Zähler. Jedes Mal wenn der Zähler einen programmierbaren Wert erreicht, wird ein Interrupt erzeugt und ein zweiter Zähler inkrementiert. Letzteres macht es möglich, nie einen Tick zu verpassen, auch wenn das System überlastet ist [Get07]. Dadurch erhöht sich die Zuverlässigkeit des Systems. Abbildung 3-3 zeigt ein PIT Block Diagramm.

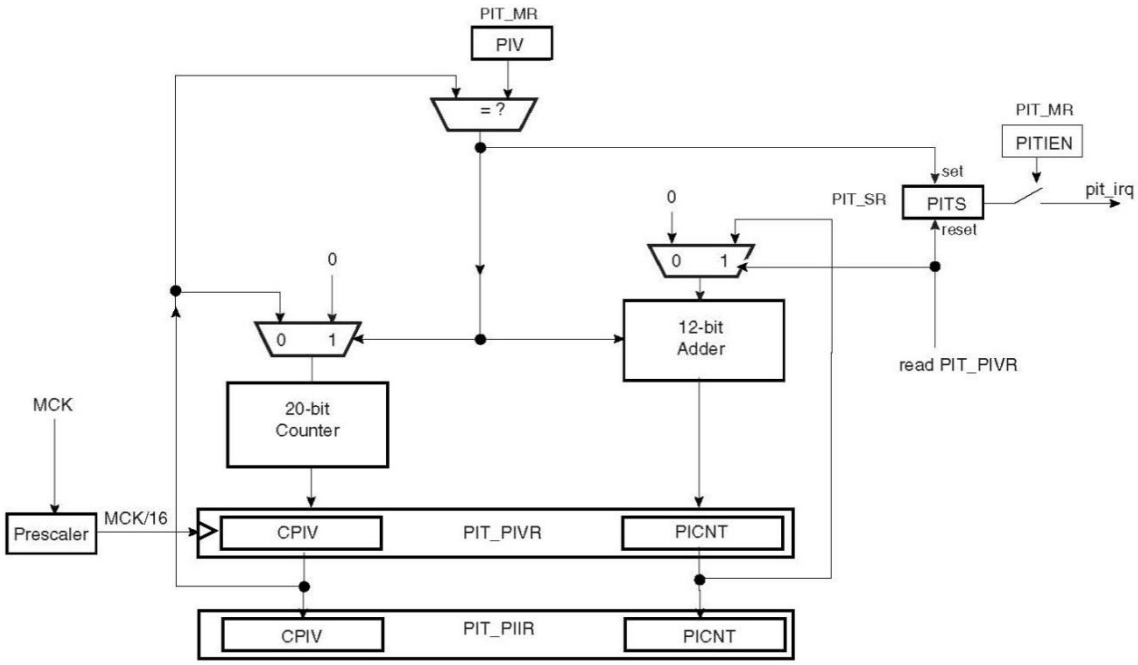

Abbildung 3-3: Periodic Interval Timer (PIT) [ATM11, S.127]

In dieser Diplomarbeit wird der PIT eine Zeitbasis von 1ms liefern. Jedes Mal, wenn der PIT-Interrupt ausgelöst wird, wird ein 32-Bit-Zähler erhöht. Eine Wait-Funktion verwendet diesen Zähler damit eine Anwendung für einen bestimmten Zeitraum ausgesetzt werden kann. Diese Wait-Funktion wird für verschiedene Warte-Praktiken benutzt.

3.2.4.2 Die Initialisierung

Der PIT ist ein Teil des System-Controllers der kontinuierlich getaktet wird. Aus diesem Grund ist es nicht notwendig, einen peripherie Takt (clock) in der PMC zu aktivieren.

Das Mode-Register enthält den Periodic Interval Value (PIV), die dem PIT anzeigt, wann der interne Zähler zurückgesetzt werden muss. Es muss durch die Anzahl der Ticks (MCK/16) in einer Millisekunde programmiert werden:[ATM11, S.128]

$$PIV = \frac{MCK}{16} \times 0{,}001$$

Vor dem Start des Timer, muss der Interrupt im AIC konfiguriert werden. Weitere Informationen über diesen Schritt findet man im Kapitel 3.2.2.3. Sobald die AIC Konfiguration abgeschlossen ist, kann der Interrupt im PIT-Mode Register durch Setzen des PITIEN-Bit[ATM11, S.131] aktiviert werden. Analog kann er auch im gleichen Arbeitsgang durch Setzen des PITIEN-Bit gestartet werden[Get07].

3.2.4.3 Der Interrupt-Handler

Der Interrupt wird beim Lesen des PIT-Registers implizit. Dieses Register enthält zwei Werte: [ATM11]

- CPIV (20-Bit Periodic Interval Timer Counter, 20-Bit): hier steht der aktuelle Wert des internen Zählers.
- PICNT (12-Bit Periodic Interval Counter): hier steht die Anzahl der Ticks, die seit dem letzten Lesen der PIVR erzeugt worden sind.

Ein zweites Register (PIT-Image Register) enthält die gleichen Werte. Dadurch kann man das PIT-Image Register ohne Löschen aller ausstehende Interrupts lesen [ATM11].

Der Interrupt-Handler für das PIT ist wie folgt realisiert. Zuerst wird der PIVR Wert ausgelesen, um den PICNT abzurufen. Dann wird eine globale Variable mit der Anzahl der gelesenen Takte inkrementiert. Hierbei gilt es zu beachten, dass es notwendig ist zu prüfen, ob wirklich ein Interrupt auf der PIT ansteht. Der System-Controller Interrupt kann von mehreren Peripheriegeräten genutzt werden, es kann dadurch irgendeiner von ihnen der Auslöser sein. Dies wird durch das Lesen der Status Register des PIT verifiziert. Das PITS Bit wird gesetzt, wenn ein Interrupt ansteht [Get07].

Je nachdem, wie lange das System im Betrieb sein soll kann ein 32-Bit-Zähler nicht immer geeignet sein. Zum Beispiel wird durch einen 1ms Tick Zähler nach etwa 50 Tagen der Zähler überlaufen und dies kann evtl. nicht genug für eine spezifische Anwendung sein [Get07]. In diesem Fall kann ein größerer Zähler implementiert werden.

3.2.4.4 Wait-Funktion

Mit dem globalen Zähler ist eine Wait-Funktion mit einer Millisekunde als Parameter sehr einfach zu implementieren.

Beim Aufrufen der Funktion wird zuerst der aktuelle Wert des globalen Zählers in einer lokalen Variable gespeichert. Dann wird die angeforderte Anzahl von Millisekunden, die als Argument gegeben sind, aufaddiert. So entsteht eine Schleife, bis der globale Zähler gleich oder größer als der berechnete Wert ist [Get07].

3.2.5 Die Verwendung des Parallel Input/Output Controller

3.2.5.1 Das Grundprinzip

Die Pins des AT91SAM kann man entweder durch eine Peripherie Funktion oder direkt als allgemeine Input/Output Pins verwenden. Alle diese Pins werden von einem oder mehreren parallelen Input/Output (PIO) Controllern verwaltet [ATM11].

Ein PIO-Controller (Abbildung 3-4) ermöglicht es dem Programmierer sowohl den Pin als auch die zugehörige Peripherie oder alles als allgemeine IO zu konfigurieren. Im zweiten Fall kann man das Niveau des Pins als Input oder Output mit mehreren Registern des PIO Controllers einstellen. Für jeden Pin kann auch ein einzelner interner Pull-Up aktiviert werden [Get07].

Darüber hinaus kann der PIO Controller eine Statusänderung auf einem oder mehreren Pins erkennen und gegebenenfalls einen Interrupt auslösen.

In diesem Beispiel verwaltet der PIO Controller zwei LEDs und zwei Tasten. Die Tasten werden so konfiguriert, dass ein Interrupt ausgelöst wird, wenn sie gedrückt werden. Für weitere Information über Funktionen und Initialisierung von verschiedenen Registern siehe [ATM11, S.425].

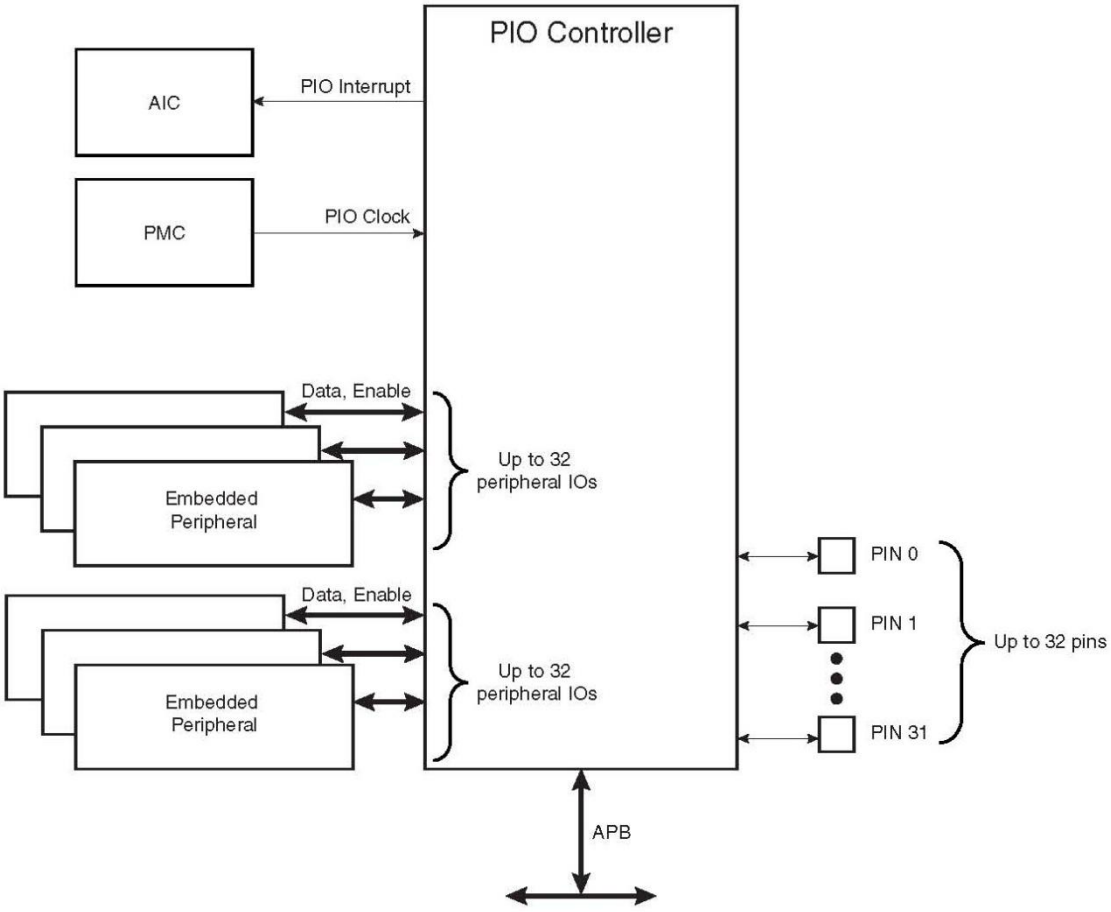

Abbildung 3-4: PIO Controller Block Diagramm [ATM11]

3.2.5.2 Die Initialisierung

Als einzige Initialisierungsoperation muss der Peripheriegeräte-Takt des PIO Controllers in der PMC aktiviert werden.

3.2.5.3 Die LED Konfiguration

Die beiden über PIO angeschlossenen Pins an den LEDs müssen als Ausgänge (Output) konfiguriert werden, damit sie aktiviert oder deaktiviert werden können. Zuerst muss der PIOC Controller und der PIOB Controller im PIO Enable Register (PER) durch schreiben des Werts „1“ über die entsprechenden Bits (PB8 und PC29) aktiviert werden.

Die Richtung von PIO wird über zwei Register kontrolliert. Zum einen das Output Enable Register (OER) und zum anderen das Output Disable Register (ODR). In dem Falle das beide PIOs als Output geschaltet sind, wird der gleiche Wert wie in OER geschrieben.

Es ist zu beachten, dass es individuelle interne Pull-Ups auf jeden PIO-Pin gibt. Diese Pull-Ups sind standardmäßig aktiviert. Da sie nutzlos für die Ansteuerung von LEDs sind, sollten sie deaktiviert werden. Dies wird durch die Pull-Up Disable-Register (PUDR) des PIOC und PIOB gemacht.

3.2.5.4 Die Ansteuerung von LEDs

Die LEDs werden an- oder ausgeschaltet. Dies geschieht durch Veränderung der Werte auf den PIOs an denen sie angeschlossen sind. Nachdem die PIOs konfiguriert wurden, können ihre Ausgangswerte durch das Schreiben der Pin-IDs in die Set Output Data Register (SODR) und die Clear Output Data Register (CODR) der PIO Controller gewechselt werden. Das Pin Data Status Register (PDSR) zeigt den aktuellen Status an jedem Pin an [Get07].

3.2.5.5 Das Konfigurieren der Tasten

Wie bereits erwähnt, werden die beiden PIO-Pins mit den Schaltern auf der Platine verbunden, und sind dann als Eingänge konfiguriert. Außerdem wird ein Interrupt für beide Tasten konfiguriert, damit wird ein Interrupt ausgelöst, sobald eine Taste gedrückt oder losgelassen wird.

Nachdem die PIOC Kontrolle auf der PIO (durch das Schreiben PER) aktiviert ist, werden sie als Eingänge durch das Schreiben ihrer IDs in ODR konfiguriert. Im Gegensatz zu den LEDs, ist es hierbei nötig die Pull-Ups zu aktivieren.

Die Interrupt Aktivierung auf die beiden Pins wird einfach über die Interrupt Enable Register (IER) gemacht. Allerdings muss der PIO-Controller Interrupt wie in Kapitel 3.2.2.3 beschrieben, konfiguriert werden [Get07].

3.2.5.6 Der Interrupt Handler

Der Interrupt-Handler für die PIO-Controller muss zuerst überprüfen, welche Taste gedrückt wurde. Die PDSR zeigt den Pegel für jeden Pin an, hierdurch kann man feststellen, ob und welche Taste derzeit gedrückt ist. Alternativ kann das Interrupt Status Register (ISR) berichten ob die PIOs ihren Status seit dem letzten Lesen des Registers verändert haben [Get07].

In der Beispiel Software wurden die beiden Möglichkeiten kombiniert, um eine Zustandsänderung des Interrupt sowie ein bestimmtes Niveau an den Pins zu erkennen. Dies entspricht sowohl dem „Drücken“ als auch dem „Loslassen“ als Aktion auf der Taste.

Wie schon in der Anwendungsbeschreibung erwähnt, ist es möglich, dass durch jede der Tasten das Blinken einer LED aktiviert oder deaktiviert wird. Zwei Variablen werden als boolesche Werte verwendet, um anzuzeigen wenn eine LED blinkt. Wenn der Status der LED die durch die Timer-Counter umgeschaltet werden geändert ist, wird der TC-Takt entweder angehalten oder durch die Interrupt Handler neugestartet.

4 USB

USB ist die Abkürzung von "Universal Serial Bus" und wurde von Intel entwickelt. Mithilfe von USB kann man verschiedene Peripheriegeräte an den Computer anschließen. USB ist ein serielles Bussystem, das bedeutet die Daten werden bitweise nacheinander übertragen. Die Daten laufen in beiden Richtungen auf denselben Leitungen. Mit Hilfe von USB können theoretisch bis zu 127 Peripheriegeräte an den Computer angeschlossen werden. Damit Jedes angeschlossene Gerät gezielt angesprochen werden kann, erhält jedes Gerät eine eigene Adresse. Der Abstand vom PC zum Peripheriegerät kann über eine USB-Verbindung maximal 5 Meter betragen. Über USB können die Daten mit bis zu 12 MBit/s (USB 3.0 bis zu 500 MByte/s) übertragen werden [Sch01].

4.1 USB Grundlagen

Die physikalische Struktur von USB (Abbildung 4-1) wird als Stern-Topologie bezeichnet. Dieses Netz besteht aus verschiedenen Geräten (Devices) und einem Controller. Alle Geräte werden durch einen Controller (z.B. im PC oder einem Mikrocontroller) zentral verwaltet. Der Controller steuert die Kommunikation zwischen sich selbst und den Devices. Es kann Daten in einen Pufferspeicher (Endpunkt) des Device schreiben und auslesen. Die Geräte mit niedriger Geschwindigkeit besitzen maximal zwei Endpunkte und die Geräte mit höherer Geschwindigkeit besitzen bis zu 16 Endpunkte [Sch06].

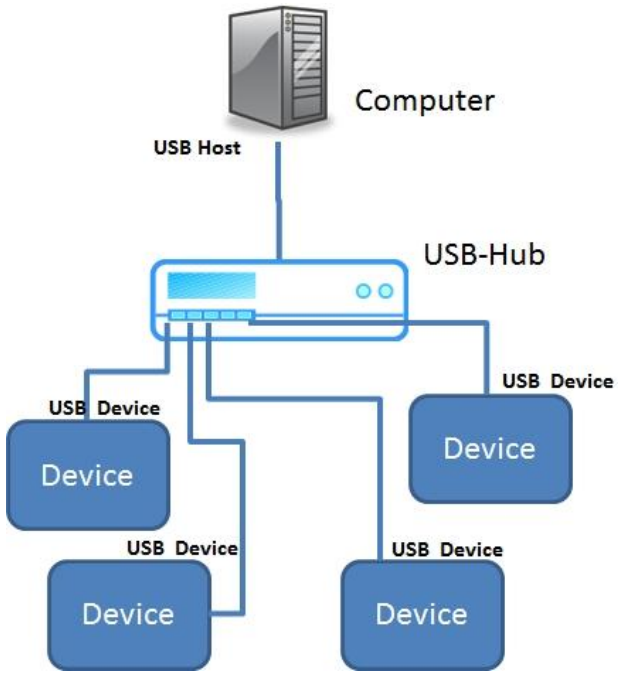

Abbildung 4-1: der sternförmige Aufbau des USB mit einem Hub

Für die Übertragung werden bei USB mehrere logische Datenkanäle eingerichtet, die so genannten Pipes. Jedes Pipe verbindet einen Endpunkt des Host mit einem Endpunkt des Device. Wenn ein USB Gerät mehrere Endpunkte hat, werden auch mehrere Pipes unterstützt. Ein Endpunkt ist physikalisch ein Speicherplatz, der als FIFO Buffer aufgebaut ist. Durch die Endpunkte können die USB Geräte Daten senden und empfangen.

Beim Anschließen eines USB Gerätes an den Host (Enumeration, siehe Abschnitt 4.3) werden folgende Aktionen durchlaufen. Jedes Device meldet sich zunächst beim Controller an, wenn der USB Bus neu gestartet, oder wenn das Device an den Bus angeschlossen wird. Dadurch bekommt jedes Device eine eindeutige Adresse durch den Controller zugewiesen. Der Controller liest dann die Konfigurationseinstellung vom Endpunkt 0 aus, um die Verbindung korrekt herstellen zu können [Sch07].

4.1.1 Aufbau des USB-Kabels und der Signalpegel

Das USB-Kabel besteht aus folgenden vier elektrischen Leitungen:

- VCC: diese Leitung hat +5 Volt Spannung und dient der Stromversorgung des USB-Geräts
- GND: Masseleitung für die Stromversorgung des USB-Geräts
- D-: Datenleitung
- D+: Datenleitung

Auf den Datenleitungen werden die Daten in binärer Form übertragen. Zwei definierte logische Zustände J und K, die sich aus den einzelnen Zuständen der zwei Signalleitungen D+ und D- zusammensetzen, werden wie in Tabelle 4-1 gezeigt zusammengesetzt [Sch06].

Tabelle 4-1: Signalpegel [Sch06]

Lowspeed					Fullspeed				
Signal	**Pegel**	**D+**	**D-**	**Bemerkung**	**Signal**	**Pegel**	**D+**	**D-**	**Bemerkung**
J	0	H	L	(D-)-(D+)>200 mV	J	1	L	H	(D+)-(D-)>200 mV
K	1	L	H	(D+)-(D-)>200 mV	K	0	H	L	(D-)-(D+)>200 mV

4.1.2 USB Protokoll (Paketformat)

Bei USB werden die Daten Paketweise übertragen. Jeder Paket Typ besteht aus verschiedenen Feldern. Die Felder in einem USB Paket gliedern sich weiter in verschiedene Typen auf: [Sch06]

- SYNC: jedes Paket einer Datenübertragung fängt mit einem SYNC Feld an. Der Zweck des SYNC Feldes ist die Synchronisation zwischen Empfänger und Sender. Das Feld hat

8 oder 32 Bit. Die letzten beiden Bits zeigen den Beginn des PID Feldes an. Das SYNC-Feld wird somit in allen Pakettypen gebraucht.

- PID (Paket ID): jeder Paket Typ wird durch eine Paket ID gekennzeichnet. Das PID Feld hat 8 Bit. Die ersten vier Bit sind die Paket ID, die letzten vier Bit sind erneut aus der Paket ID zusammengesetzt dieses Mal jedoch in invertierter Reihenfolge, siehe hierzu Abbildung 4-2.

PID0	PID1	PID2	PID3	$\overline{PID0}$	$\overline{PID1}$	$\overline{PID2}$	$\overline{PID3}$

Abbildung 4-2: PID Feld

- Adresse: durch ein 7 Bit großes Adressfeld lassen sich 128 Geräte, inklusive dem Host, adressieren.
- Endpunkt: da manche Geräte mehr als einen Endpunkt haben, wird eine Endpunkt-Nummer genutzt. Durch ein 4-Bit Endpunkt-Feld werden bis zu 16 verschiedene Endpunkte in USB Geräten adressiert.
- Framenummer: dieses Feld hat 11 Bit und enthält die Frame (Paket) Nummer.
- Daten: das Daten Feld besteht aus 0 bis 1024 Bit. Die Feldgröße hängt vom Transfertyp und der Busgeschwindigkeit ab.
- CRC: je nach Paket Typ besteht das CRC Feld entweder aus 5-Bits oder aus 16-Bits. Das CRC Feld wird für Fehlerüberprüfung verwendet.

Das USB Protokoll hat folgenden Paket Aufbau: [Sch07]

Der Ablauf einer Bus Transaktion besteht aus bis zu drei Paketen. Die Transaktion fängt mit einem Token-Paket an, das Paket wird an den Host Controller gesendet, damit wird die Art und die Richtung der Transaktion sowie die Geräteadresse und die Endpunktnummer bestimmt. Mit dem darauffolgenden Datenpaket können Daten zwischen Host und einem Gerät ausgetauscht werden. Mit Hilfe eines Handshakepaketes kann der Empfänger antworten, ob die Übertragung erfolgreich war oder nicht.

- Start-of-Frame Paket: wie in Abbildung 4-3 gezeigt ist, beinhaltet dieses Paket eine PID, eine Framenummer und eine Prüfsumme(CRC).

PID (8-Bits)	**Frame Nummer** (11-Bits)	**CRC** (5-Bits)

Abbildung 4-3: Start-of-Frame Paket

- Token Paket: das Token-Packet (Abbildung 4-4) hat folgende vier Felder: PID, Adresse, Endpunkt und CRC.

PID (8-Bits)	**ADDR** (7-Bits)	**ENDP** (4-Bits)	**CRC** (5-Bits)

Abbildung 4-4: Token Paket

- Daten Paket: das Daten Paket (Abbildung 4-5) fängt mit einer PID an, darauf folgen die Daten und schließlich endet es mit einer Prüfsumme.

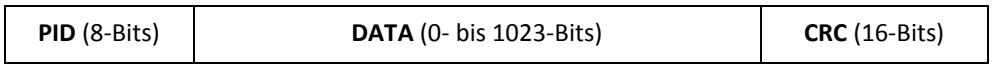

Abbildung 4-5: Daten Paket

- Handshake Paket: wie in Abbildung 4-6 gezeigt besteht dieses Packet nur aus einem PID Feld.

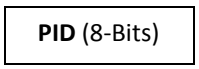

Abbildung 4-6: Handshake Paket

4.2 USB Device Port des AT91SAM9263 Board

4.2.1 Überblick

Der USB Device Port (UDP) ist kompatibel mit dem Universal Serial Bus (USB) in Version 2.0. Jeder Endpunkt kann in einer von mehreren USB Übertragungsarten konfiguriert werden. Er wird direkt dem RAM zugeordnet, um die aktuellen Daten zu speichern. Wenn zwei Bänke verwendet werden, wird eine DPR Bank für das Lesen und die Andere für das Schreiben zugeordnet. Während eine Bank durch den Prozessor gelesen oder geschrieben wird, kann die andere von dem USB Peripheriegerät gelesen oder geschrieben werden. Durch diese Technik wird die maximale Bandbreite (z.B. 1 MByte / s) sichergestellt. In Tabelle 4-2 werden die USB Endpunkte genauer erklärt [ATM11].

Tabelle 4-2: USB-Endpoint Beschreibung[ATM11, S. 859]

Endpoint Number	Mnemonic	Dual-Bank	Max.Endpoint Size	Endpoint Type
0	EP0	Nein	64	Control/Bulk/Interrupt
1	EP1	Ja	64	Bulk/Iso/Interrupt
2	EP2	Ja	64	Bulk/Iso/Interrupt
3	EP3	Nein	64	Control/Bulk/Interrupt
4	EP4	Ja	256	Bulk/Iso/Interrupt
5	EP5	Ja	256	Bulk/Iso/Interrupt

4.2.2 Block Diagramm

Abbildung 4-7 zeigt ein UDP Blockdiagramm. Der Zugriff auf den UDP wird über die APB-Bus-Schnittstelle sichergestellt. Lese- und Schreibzugriffe auf die Daten (FIFO) werden physikalisch durch Lesen und Schreiben von APB Registern ausgeführt.

Das UDP Peripheriegerät erfordert zwei Takte: der eine Takt stammt vom Peripheriegerät und ergibt die Master Clock-Domain (MCK) und einen weiteren 48-MHz-Takt (UDPCK). Ein USB 2.0 Full Speed Transceiver ist ein Embedded-System und wird durch das Serial Interface Engine (SIE) gesteuert[ATM11].

Das Signal „external_resume“ ist optional. Es ermöglicht dem UDP Peripheriegerät wenn es sich im System-Modus befindet, aufzuwachen. Der Host wird dann benachrichtigt dass, das Gerät eine Wiederaufnahme der Verbindung angefordert hat. Diese optionale Funktion muss mit dem Host während der Enumeration ausgehandelt werden[ATM11].

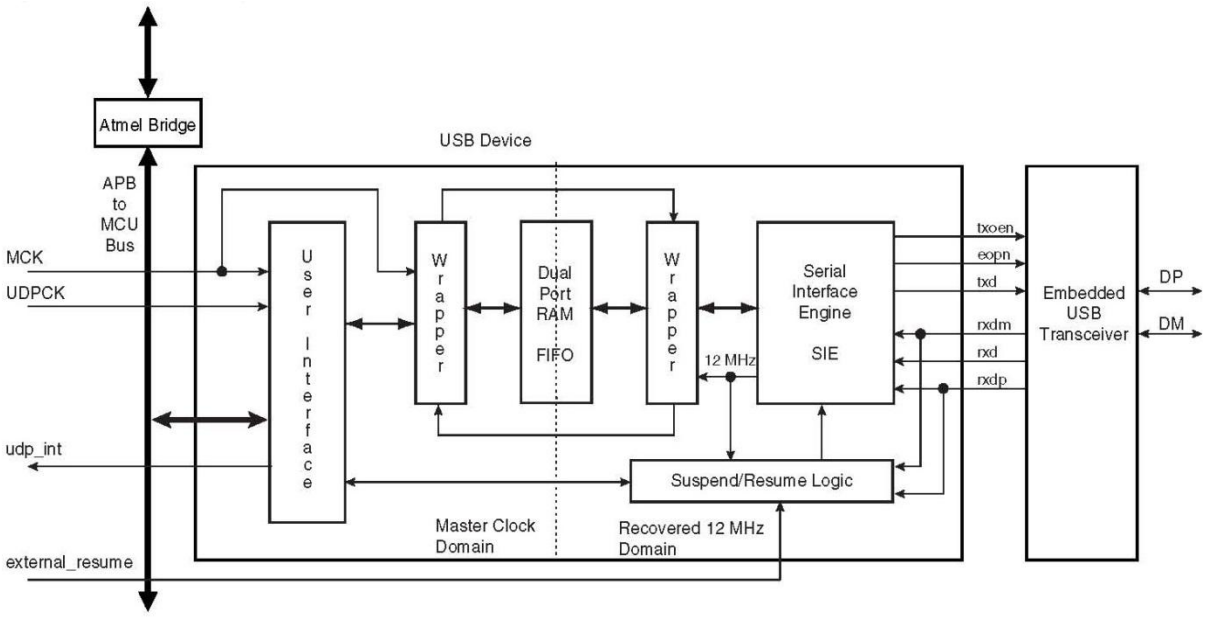

Abbildung 4-7: USB Device Diagramm [ATM11, S.860]

4.3 USB Device Zustandsdiagramm (Enumeration)

Ein USB Gerät verfügt über mehrere mögliche Zustände. Abbildung 4-8 zeigt das zugehörige Zustandsdiagramm.

Übergänge von einem Zustand in den anderen hängen von dem USB-Bus Zustand ab oder werden durch gesendete Standard-Anfragen z.B. durch Kontrolltransaktionen über den Standard-Endpunkt (Endpunkt 0) veranlasst.

Nach einer gewissen Inaktivität des Busses, tritt das USB-Gerät in den so genannten Suspend-Modus. Die Einschränkungen im Suspend-Modus für Bus gesteuerte Anwendungen sind sehr streng [ATM11].

Peripheriegeräte im Suspend-Modus können vom Host des Gerätes aufgeweckt werden indem er eine „Resume“ Signal (eine Bus Aktivität) sendet. Im folgenden Absatz werden die einzelnen Zustände genauer erklärt. [ATM11, S875]

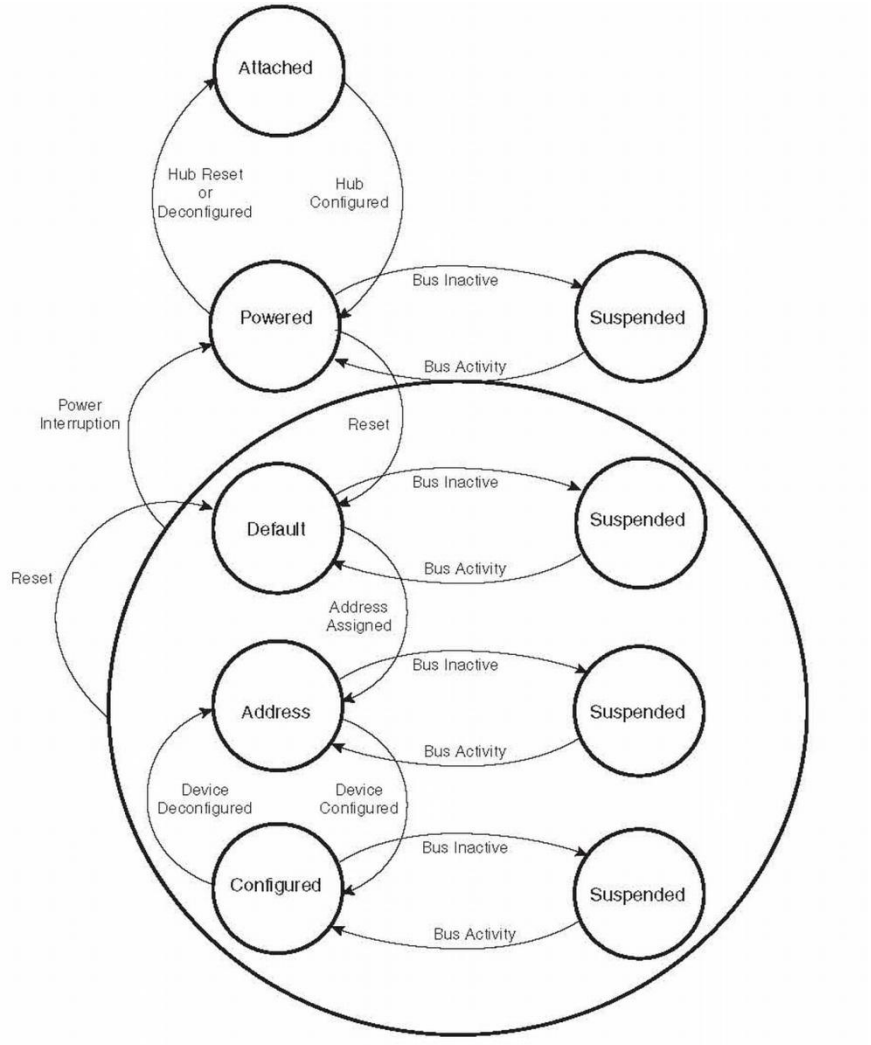

Abbildung 4-8: USB Device Zustandsdiagramm [ATM11, S. 874]

- **„Attached“ Zustand:** Wenn ein Gerät angeschlossen oder der Strom beim Systemstart eingeschaltet wird, geht der UDP in den „Attached“ Zustand. Nach der Verbindung geht das Gerät in den „Powered“ Zustand über.
- **Vom „Powered“ Zustand in den „Default“ Zustand:** Nach der Verbindung mit einem USB Host wartet das USB Gerät auf einen „End Of Bus Reset“. Die Flagge „ENDBUSRES“ wird in dem Register UDP_ISR gesetzt und dadurch ein Interrupt

ausgelöst. Sobald der „ENDBUSRES“ Interrupt ausgelöst wurde, wechselt das Gerät in den „Default“ Zustand.

- **Vom „Default“ Zustand in den „Address“ Zustand:** Nachdem die Adresse durch eine Standardgerät Anfrage festgelegt wurde, wechselt das Gerät in den „Address“ Zustand.
- **Vom „Address“ Zustand in den „Configured“ Zustand:** Sobald eine gültige Standard Konfigurationsanfrage entgegen genommen wurde und anerkannt ist, wechselt das Gerät in den „Configured“ Zustand.
- **Der „Suspend“ Zustand:** Wenn keine Bus-Aktivität auf dem USB Bus erkannt wird, geht das Gerät in den „Suspend“ Zustand.

Bemerkung: Lese und Schreib-Operationen auf die UDP Register sind nur zulässig, wenn für den MCK UDP ein Peripheriegerät aktiviert ist.

- **Empfangen eines „Host Resume“:** Im „Suspend“ Modus wird ein „Resume“ Event asynchron auf dem USB Bus erkannt, dadurch werden Transceiver und Takt (Clock) deaktiviert.

Bemerkung: Lese und Schreib-Operationen auf die UDP-Register sind nur zulässig, wenn für den MCK UDP ein Peripheriegerät aktiviert ist.

- **Senden eines „Device-Remote-Wakeup“:** Während sich ein Gerät im „Suspend“ Modus befindet, kann der Host das Gerät aufwecken, indem er ein „Resume“ Signal (Bus Aktivität) sendet.

4.4 USB Framework

Dieser Abschnitt beschreibt ein USB-Framework, das für Atmel®AT91ARM® Thumb®-basierte Mikrocontroller entwickelt worden ist. Es ermöglicht die schnelle Entwicklung von USB-kompatiblen Treibern, wie die Communication Device Class (CDC)[ATM06].

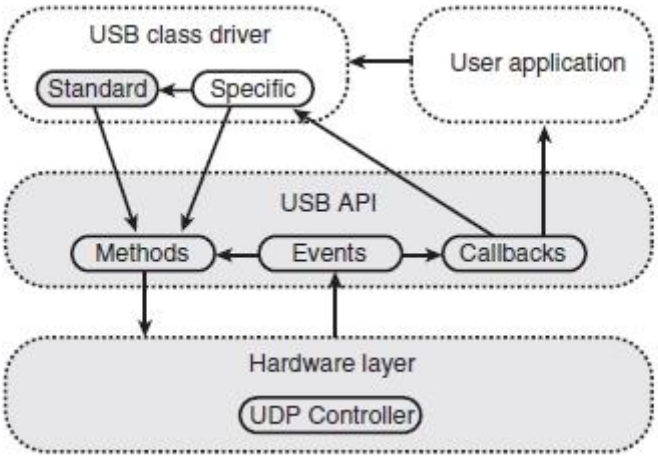

Abbildung 4-9: USB Framework Architektur [ATM06]

Die meisten Mikrocontroller der AT91-Familie betten einen USB-Controller ein. Da jedoch mehrere verschiedene Steuerungen in den Geräten verwendet werden, bietet das Framework eine Abstraktion der Hardware-Schicht. Dies bedeutet, dass eine Applikation die dieses Framework verwendet leicht auf jedes AT91 Gerät portiert werden kann. Abbildung 4-9 zeigt eine USB Framework Architektur. Die Komponenten in grau, sind Bestandteil des Frameworks [ATM06].

Abbildung 4-9 zeigt die folgenden dreistufigen Schichten: [ATM06]

- Eine Hardware-Schicht, die Low-Level-Operationen für den USB-Controller bereitstellt
- Die USB-API bietet hardware-unabhängige Methoden und Strukturen.
- Der Applikation-Layer, besteht aus einer USB-Klasse für den Treiber und der eigentlichen Benutzeranwendung.

Das Framework beinhaltet die USB-API, die Hardwareschicht, sowie einen Standard Anfragen (request) Handler. Die Applikationsschicht ist auf die Gerätefunktionalität aufgebaut.

Es muss eine Form der Kommunikation zwischen der USB-API und der Applikationsschicht sichergestellt sein. Dies wird durch die Verwendung von „Callbacks“ bewerkstelligt. „Callbacks“ sind Funktionen, die automatisch von der USB-API aufgerufen werden, um bestimmte Operationen auszuführen [ATM06].

4.4.1 Framework Beschreibung

Das Framework hat folgende Komponenten: [ATM06]

- Standard-USB-Strukturen: Die folgenden Standard-Strukturen wurden in dem USB-Framework umgesetzt:
- Setup Anfrage für Daten: S_usb_request
- Geräte Deskriptor: S_usb_device_descriptor
- Konfigurations Deskriptor: S_usb_configuration_descriptor
- Interface Deskriptor: S_usb_interface_descriptor
- Endpunkt Deskriptor: S_usb_endpoint_descriptor
- Gerätekennzeichungs Deskriptor: S_usb_device_qualifier_descriptor
- String Deskriptor Null: S_usb_language_id

- USB-API Struktur: Von der USB-API werden mehrere spezifische Strukturen verwendet um verschiedene Operationen, wie den Aufruf von „Callbacks“ oder den Zugriff auf den USB-Controller auszuführen. Es gibt die vier folgenden Strukturen:
  - S_usb
  - S_usb_driver
  - S_usb_endpoint
  - S_usb_callbacks

- USB-API Methoden: Die USB-API bietet mehrere Methoden, um die folgenden Operationen durchzuführen: [ATM06]
  - Änderung des Gerätezustands
  - Behandlungen von Ereignissen, die (Handling events) von dem USB-Controller gesendet werden.
  - Änderung des Verhaltens eines Endpunkts
  - Übertragen von Daten („USB_Read“, „USB_Write“)
  - Weitere spezielle Funktionen

- Callback-API: Die Callback-API ist ein Vermittler der Kommunikation zwischen Benutzer und Anwendung der USB-API. Wenn bestimmte Operationen durchgeführt werden müssen, fordert der USB-Treiber mehrere externe Funktionen an. Man nennt diese Funktionen „Callback“. Wenn diese Funktion aufgerufen wird, wird diese die Benutzeranwendung über ausstehende Ereignisse benachrichtigen. Die Definition von allen „Callbacks“ ist nicht zwingend erforderlich. Zum Beispiel, wenn das Gerät nicht in Low-Power-Modus gehen darf, dann darf kein „Suspend-Callback“ angeboten werden. Ob ein Callback obligatorisch ist, kann in der Beschreibung [ATM06] nachgelesen werden.

- Standard-Request-Handler: Die USB-Spezifikation 2.0 definiert eine Reihe von Standard-Anfragen, die von allen Geräten implementiert werden müssen. Da die meisten Klassen für Treiber diese Anfragen dem Standard nach behandeln, bietet das USB-Framework eine Möglichkeit zur einfachen Anwendung [ATM06].

4.5 AT91 USB CDC Treiber Entwicklung

Die Kommunikation Device Class (CDC) ist ein universeller Weg, um alle Arten von Kommunikation über den Universal Serial Bus (USB) zu ermöglichen. Diese Klasse macht es möglich Telekommunikationsgeräte, wie digitale oder analoge Telefone, sowie Netzwerkgeräte wie ADSL-oder Kabelmodems, zu verbinden[AT09].

Ein CDC-Gerät ermöglicht die Implementierung von sehr komplexen Geräten, kann aber auch als eine sehr einfache Methode für die Kommunikation über den USB-Port verwendet werden. Zum Beispiel kann sich ein CDC Gerät als virtuelle COM-Schnittstelle ausgeben, welche die Anwendungsprogrammierung auf der Host-Seite stark vereinfacht[AT09].

4.6 Implementierung der USB Test Programme auf dem PC und dem Mikrocontroller

In diesem Abschnitt wird mithilfe der Implementierung von zwei Test Programmen (eines auf dem PC und das Andere auf dem Mikrocontroller) gezeigt, wie ein Mikrocontroller und ein PC über USB miteinander kommunizieren können.

4.6.1 Aufgabenerklärung

- **Testprogramm auf dem Mikrocontroller**: das Programm läuft auf dem Mikrocontroller und wartet auf gesendete Nachrichten des PCs auf dem USB-Port. Das Programm sollte den UDP des Mikrocontrollers und zugehörige Peripheriegeräte konfigurieren. Es liest die gesendeten Nachrichten des PCs über UDP. Wenn das Programm Nachrichten bekommt, muss es diese erkennen und verarbeiten. Je nachdem welche Nachrichten Typen empfangen werden, werden folgende Aufgaben ausgeführt:
- Ein Handshake mit dem PC
- Konfiguration der angeforderten LED und Benachrichtigung
- Konfiguration der angeforderten Rate mit Hilfe einer Wait-Funktion und erneute Benachrichtigung
- LEDs mit der angeforderten Anzahl blinken lassen und wieder benachrichtigen
- **Test Programm auf dem PC**: dieses Programm läuft auf dem PC und kommuniziert mit dem Mikrocontroller über USB. Dieses Programm sendet verschiedene Nachrichten um die zwei LEDs auf dem Mikrocontroller mit der angeforderten Taktrate und gewünschten Häufigkeit blinken zu lassen.

4.6.2 Handshake Prinzip

Für die Kommunikation zwischen PC und Mikrokontroller wird ein zwei Phasen Handshake implementiert. Das heißt, der PC schickt einen Befehl an den Mikrokontroller und wartet auf die Antwort des Mikrokontrollers. Wenn als Antwort eine positive Bestätigung zurückkommt, schickt der PC den nächsten Befehl an den Mikrokontroller, ansonsten wiederholt der PC den vorherigen Befehl.

Der Mikrocontroller wartet immer auf Nachrichten des PCs. Wenn der Mikrocontroller eine Nachricht bekommt und verarbeitet hat führt er den entsprechenden Befehl aus und schickt eine positive Bestätigung an den PC, sonst schickt er eine Fehlernachrichte an den PC.

Wie in Abbildung 4-10 gezeigt, schickt der PC zuerst eine „HostReady“ Nachricht an den Mikrocontroller. Dadurch weiß der Mikrocontroller, dass der PC und das Anwendungsprogramm in Betrieb sind und schickt eine „MikroReady“ Nachricht an den PC zurück. Der PC sendet als nächstes eine „LedConfig“ Nachricht und der Mikrocontroller konfiguriert mithilfe dieser Nachricht die angeforderte LED. Nach der Konfiguration schickt der

Mikrocontroller „LedIsConfigured“, in diesem Falle die positive Bestätigung, dadurch weiß der PC dass der Mikrocontroller bereit für den nächsten Schritt ist. Der PC sendet seinen letzten Befehl „LedStart“, um die LED mit der angeforderten Anzahl blinken zu lassen. Nachdem der Mikrokontroller diesen Befehl empfängt, startet er die LED und lässt sie mit der angeforderten Anzahl blinken und schickt ein „LedToggleStart“ an den PC zurück. Am Ende, wenn der Mikrocontroller mit dem Blinken fertig ist, sendet der Mikrokontroller eine „LedToggleFinish“ Nachricht an den PC. Anschließend endet das Programm auf dem PC.

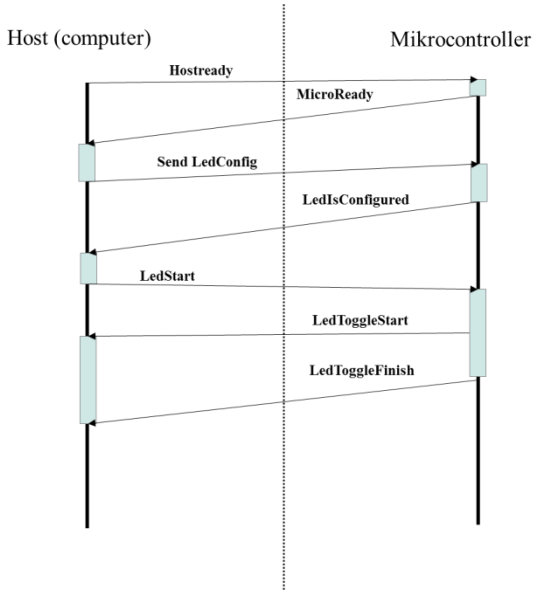

Abbildung 4-10: Handshake Prinzip

4.6.3 Nachrichten Paket Format

Wie bereits in Abschnitt 4.2.1 erklärt, gibt es bei USB verschiedene Paket Formate und eines davon ist das Daten Paket. Für die verwendete Nachricht wird das Daten Feld des Daten Pakets um eigene Anpassungen, wie in Abbildung 4-11 gezeigt, erweitert.

Für diese Implementierung werden verschiedene Nachrichtentypen mit folgendem Paketaufbau verwendet:

- **„Acknowledge“ Nachrichten**: Diese Nachricht gibt verschiedene Auskunft von Mikrocontroller und PC und wird für Handshake verwendet. Wie in Abbildung 4-11 gezeigt wird, besteht diese Nachricht aus den drei folgenden Feldern.
- Nachrichten Länge: dieses Feld hat 4 Byte, diese stehen für die Länge der Nachrichten-ID und der Daten.
- Nachrichten ID: Dieses Feld hat 1 Byte und steht für Nachrichten Identifier. Diese Feldlänge ermöglicht bis zu 256 verschiedene Nachrichten-IDs. Jede Nachricht hat eine eigene ID.

- Data: dieses Feld hat bis zu 59 Bytes. Dieses Feld ermöglicht es den Fehlertext als String zu übertragen.

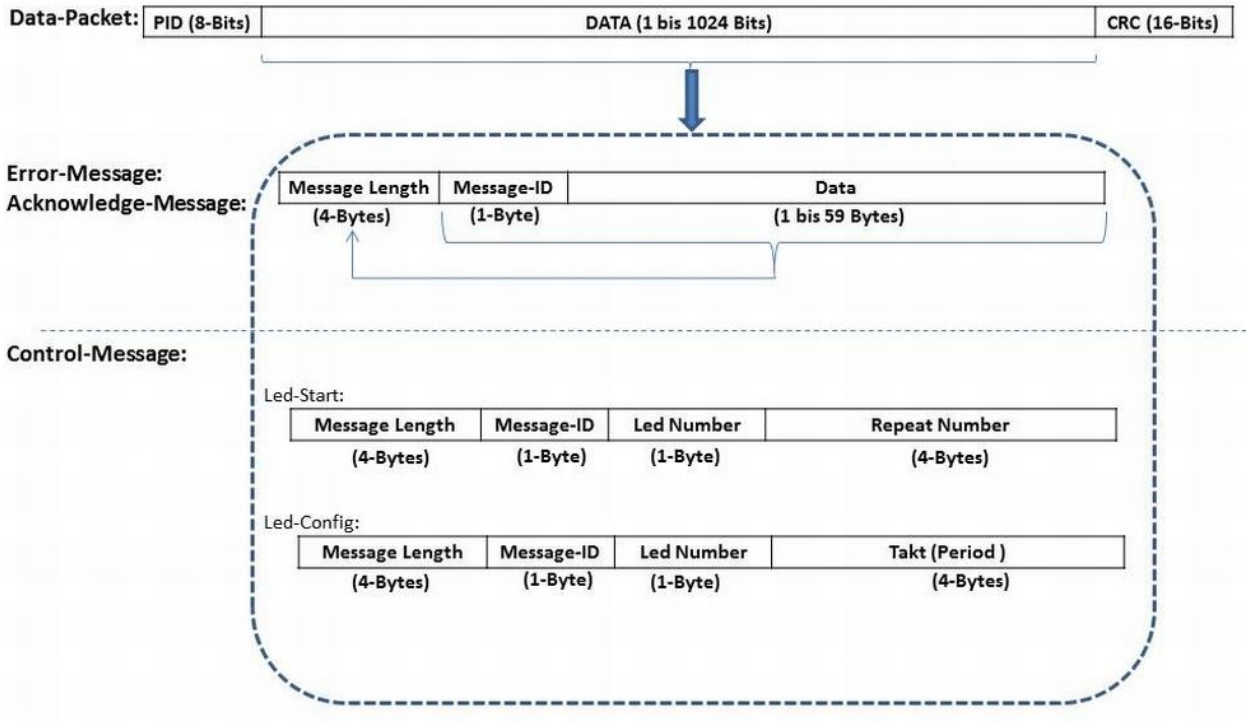

Abbildung 4-11: Nachrichten Packet Format

- **Fehler Nachrichten**: Diese Nachricht enthält alle Fehlerereignisse auf dem Mikrocontroller und auf dem PC. Der Paket Aufbau ist analog zu den „Acknowledge“ Nachrichten.

- **Kontroll-Nachrichten**: Das Test Programm auf dem PC kann mithilfe dieser Nachrichten verschiede Kontrollbefehle an den Mikrocontroller senden. In Abbildung 4-11 sind zwei Beispiele von Kontrollbefehlen („Led-Start“, „Led-Config“) dargestellt.

4.7 Implementierung des Test Programms auf dem Mikrocontroller

In diesem Abschnitt wird zuerst der Zusammenhang zwischen dem Test Programm auf dem Mikrokontroller und dem USB Framework, dem CDC Serial Driver und der Hardware durch eine Software Architektur erklärt und schließlich wird die Architektur und die Implementierung mittels eines Test Programms aufgezeigt.

4.7.1 Software Architektur des AT91 USB Framework und des CDC Serial Driver

Die Architektur besteht aus den folgenden vier Schichten:

- **Hardware Schicht**: der blaue Bereich in Abbildung 4-12 ist die Hardware Schicht. Diese Schicht ist in Abschnitt 4.2 bereits erklärt. Alle Low Level Operation finden bei diesem UDP–Controller in dieser Schicht statt.

- **USB Framework Schicht**: Der grüne Bereich in Abbildung 4-12 ist die USB Framework Schicht. Das Framework beinhaltet die USB-API und den Standard USB Klassen Treiber. Wie bereits in Abschnitt 4.4.1 erklärt, ermöglicht dies die schnelle Entwicklung von USB-kompatiblen Treibern, wie die Communication Device Class (CDC) [ATM06]. Diese Framework Applikation hat Atmel für AT91 ARM basierte Mikrocontroller geschrieben und wird in diesem Test Programm auf dem Mikrocontroller verwendet.

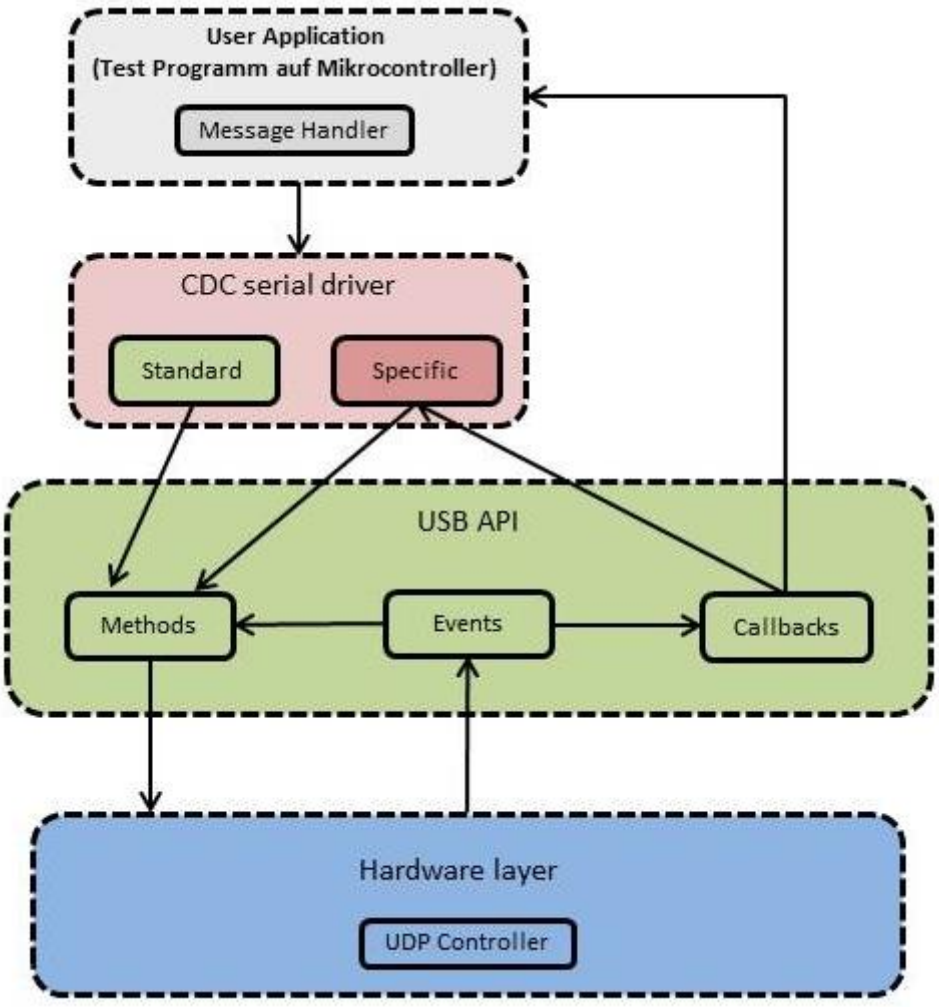

Abbildung 4-12: Architektur des Testprogramms[ATM06]

- **USB CDC Driver**: Der rote Bereich in Abbildung 4-12 ist der CDC Driver. Diese Schicht wurde bereits in Abschnitt 4.4 erklärt. Die Kommunikation Device Class (CDC) ist ein universeller Weg, um alle Arten von Kommunikation über den Universal Serial Bus (USB) zu ermöglichen [AT09]. Die verwendete CDC Driver Applikation hat Atmel speziell für AT91 ARM basierte Mikrocontroller geschrieben und wird ebenso in diesem Test Programm auf dem Mikrocontroller eingesetzt.

- **User Applikation:** Der graue Bereich ist die Benutzer Anwendung. Im folgenden Abschnitt wird diese Schicht detaillierter erklärt und es wird aufgezeigt wie die Architektur und die Implementierung dieser Schicht mit den anderen Schichten zusammen arbeitet.

4.7.2 Flussdiagramm der System-Schleife des Testprogramms auf dem Mikrocontroller

Wie bereits erwähnt, erfordert die oben definierte Aufgabe, siehe Abschnitt 4.6.1, die Verwendung von mehreren Peripheriegeräten. Daher muss auch der notwendige Code zur Inbetriebnahme des Mikrocontrollers bereitgestellt werden.

Die Abbildung 4-13 zeigt das Flussdiagramm des Test Programms. Im Folgenden wird jede Komponente des Flussdiagramms genauer beschrieben.

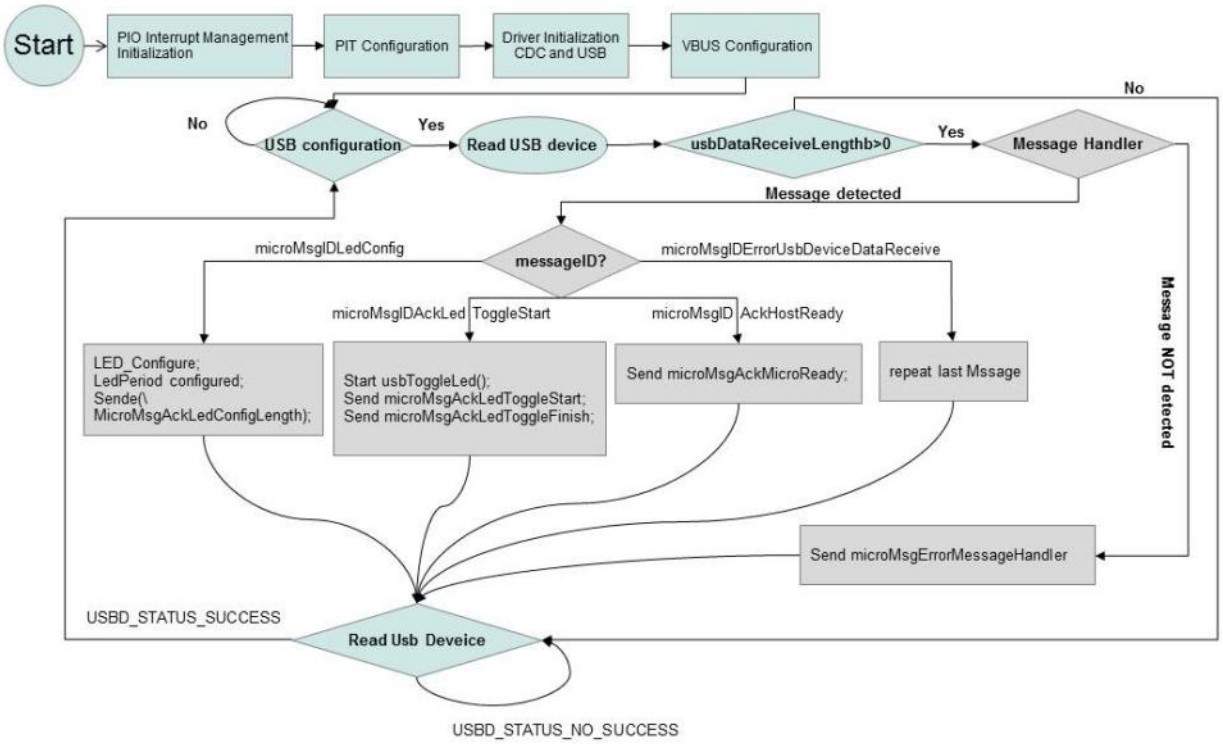

Abbildung 4-13: Flussdiagramm der System-Schleife des Test Programms auf dem Mikrocontroller (User Applikation)

4.7.2.1 PIO Interrupt Management Initialisierung

Die Funktion „PIO_InitializeInterrupts(priority)“ initialisiert das PIO-Interrupt-Management und sorgt für die gewünschte Priorität des PIO-Interrupts. Der mehrmalige Aufruf dieser Funktion führt zu einem „Reset“ der aktuellen Interrupt Konfiguration.

4.7.2.2 PIT Konfiguration

Die „ConfigurPit(void)“ Funktion, konfiguriert den „Periodic Interval Timer“, um in jeder Millisekunde einen Interrupt erzeugen zu können. Die PIT Konfiguration ist ausführlich In Abschnitt 3.2.4 erklärt.

4.7.2.3 Treiber Initialisierung

Die „CDCDSerialDriver_Initialize()“ Funktion sorgt für die Initialisierung des USB Device CDC Serial Treiber und des USB Device Port Treibers. Mithilfe dieser Funktion wird die Initialisierung in den folgenden Schritten durchgeführt:

- Initialisierung des Control Model-Attribut: die „CDCLineCoding_Initialize()“ Funktion initialisiert die Bitrate, die Anzahl der Stopp-Bits, die Paritätsprüfung und die Anzahl der Datenbits eines „CDCLineCoding“ Objektes.
- Initialisierung des Standard Treibers: durch die Funktion „USBDDriver-Initialize()“ wird ein USB-Treiber beispielweise mit einer Liste der Deskriptoren initialisiert. Wenn Schnittstellen mehrere alternative Einstellungen haben, muss ein Array zu Verfügung gestellt werden, um die aktuellen Einstellungen für jede Schnittstelle speichern zu können.
- Initialisierung des USB Treibers: Die „USBD_Init()“ Funktion führt folgende Schritte durch, um die USB Treiber zu initialisieren.
  - Die Endpoint Struktur wird zurückgesetzt.
  - Pull-up wird auf D+ konfiguriert.
  - Der Device Zustand wird auf „Attached“ gesetzt.
  - Aktivierung des Taktes (Clock) des UDP Peripheriegeräts.
  - Aktivierung des USB Takts (48 Mhz).
- USB Controller Interrupt: Der USB-Controller des Peripheriegeräts erzeugt einen Interrupt, wenn ein Ereignis eintritt. Da dieses Ereignis an die USB_Handler Methode weitergeleitet werden muss, wird eine Interrupt Service Routine installiert.
- Interrupt Konfiguration: Die „USBDCallbacks_Intialized()“ Funktion konfiguriert den UDP- und UDHPS-Interrupt. In dieser Funktion wird mithilfe von „AIC_ConfigureIT()“ eine Interrupt Handler Funktion „USBD_InterruptHandler()“ für den UDP Interrupt konfiguriert. Dieses Verfahren wurde bereits in Abschnitt 3.2.2 erwähnt. Für weitere Information siehe [ATM06], [AT09], [ATM11].
- USB Device Interrupt Handler (UDP Interrupt Service Routine): Mehrere Ereignisse können auf der USB-Controller Ebene auftreten:[ATM06]
  - Ende des Bus-Reset

- Empfang eines SETUP-Pakets
- Änderung der Bus-Aktivität (aktiv -> Idle -> aktiv ...)
- Bereitstellung eines Endpunkts
- usw.

Immer, wenn ein solches Ereignis eintritt, muss es an die USB-API weitergeleitet werden, und in angemessener Weise behandelt werden. Diese Funktionalität wird durch den USB Device Handler durchgeführt. Der Controller Interrupt muss konfiguriert werden, um den USB Device Handler „USBD_InterruptHandler()“ ausführen zu können.

4.7.2.4 VBUS Konfiguration

Die VBUS Überwachung ist erforderlich, siehe Abbildung 4-14, um Host-Verbindungen zu erkennen. Die VBUS Überwachung erfolgt über einen Standard-PIO mit Deaktivierung des internen Pullups. Wenn der Host ausgeschaltet ist, muss dies als Trennung berücksichtigt werden. Der Pullup muss deaktiviert werden, um die Stromversorgung des Host durch den Pullup-Widerstand zu verhindern [ATM11]. Abbildung 4-13 zeigt, wo die VBUS Verbindungen im Kontext liegen.

Mithilfe der „VBus_configure()“ Funktion und dem „ISR_Vbus()“ Interrupt Handler wird die VBUS Überwachung implementiert.

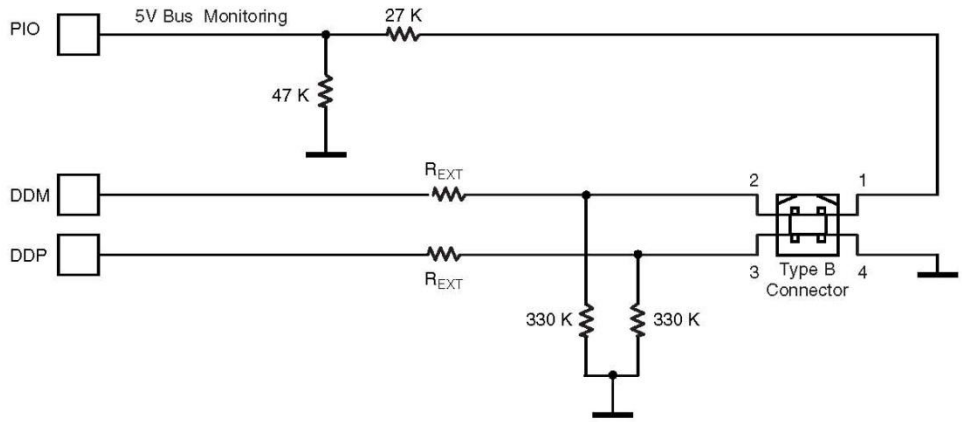

Abbildung 4-14: Board Schaltplan zur Periphere Interface Device [ATM11, S. 862]

4.7.2.5 USB Konfigurations Test

Wie in Abschnitt 4.3 bereits erwähnt, kann sich das USB-Device in verschiedenen Zuständen befinden. Im Main Programm wird mithilfe der Funktion „USBD_GetState()“überprüft, in welchem Zustand sich das USB-Device befindet. Wenn das USB-Device konfiguriert ist, wird der nächste Schritt des Programms ausgeführt ansonsten wird die Funktion „USBD_connect()“ aufgerufen und gewartet bis das USB-Device konfiguriert ist.

Im Folgenden werden die 3 wichtigsten Methoden genauer erläutert:

- **USBD_GetState():** Diese Methode gibt den aktuellen Zustand des USB-Controllers zurück. Die ersten sechs Bits des Rückgabewerts legen fest, welcher Zustand aktiv ist: [ATM06]
- SB_STATE_ATTACHED (Bit 0): Ist immer gesetzt, da die physikalische Bindung oder Trennung von dem Bus nicht erkannt werden kann.
- USB_STATE_POWERED (Bit 1): wenn der VBus vorhanden ist, wird dieses Bit gesetzt. Er wird gelöscht, sobald der VBus getrennt wird.
- USB_STATE_DEFAULT (Bit 2): Dieses Bit wird gesetzt, wenn der Bus-Reset-Vorgang abgeschlossen ist. Er wird gelöscht, wenn die Funktion „USB_Disconnect“ aufgerufen wird.
- USB_STATE_ADDRESS (Bit 3): Dieses Bit wird gesetzt, wenn eine SET_ADDRESS Anfrage mit einem nicht null „wValue“ empfangen wird. Es wird gelöscht, wenn ein „SET_ ADDRESS“ Anfrage mit einem null Wert im „wValue“ Feld empfangen wird.
- USB_STATE_CONFIGURED (Bit 4): dieses Bit wird gesetzt, wenn eine „SET_ CONFIGURATION“ Anfrage mit einem nicht null „wValue“ empfangen wird. Es wird gelöscht, wenn ein „SET_CONFIGURATION“ Anfrage mit einem null Wert im „wValue“ Feld empfangen wird.
- USB_STATE_SUSPENDED (Bit 5): Dieses Bit wird gesetzt, wenn der Bus in den Idle Modus geht. Gelöscht wird es, wenn der Bus wieder aktiv wird. Die anderen Bits zeigen den Zustand des Gerätes an, bevor es in den „Suspend“ Zustand geht.
- **USB_Connect(), USB_Disconnect():** Diese beiden Methoden steuern den Zustand des D + Pull-up. Durch diese Methoden ist es möglich, dass ein Gerät bei Bedarf eine Verbindung herstellen oder trennen kann [ATM06].

4.7.2.6 Read und Write Methoden von USB Geräten

Datentransfer (IN oder OUT) an einen Endpunkt, kann durch den Aufruf von zwei Methoden, „USB_Write()“ und „USB_Read()“, durchgeführt werden: [ATM06]

- **USB_Write():** Die „USB_Write“ Funktion sendet die Nutzdaten auf einen bestimmten Endpunkt. Sind diese Daten gleich oder größer als die maximale Paketgröße des Endpunkts es zulässt, dann sind mehrere IN-Transaktionen erforderlich. Diese Methode sollte nur auf einen IN- oder Control-Endpunkt aufgerufen werden.
  Der Schreibvorgang wird asynchron durchgeführt, das heißt. die Funktion gibt sofort eine Rückgabe ohne Wartezeit auf das Ende der Übertragung. Wenn die Übertragung abgeschlossen ist, kann optional eine vom Benutzer angegebene Callback-Funktion aufgerufen werden. Die „USB_Write“ Methode wird in der „CDCDSerialDriver_Write“ Funktion verwendet, um die Daten an den Host zu senden. In diesem Test Programm wird keine „Callback“ Funktion für diese Methode verwendet.

- **USB_Read():** Diese Funktion liest eingehende Daten auf einen entsprechenden Endpunkt. Die Übertragung stoppt, entweder wenn der Puffer voll ist, oder ein kurzes Paket (Größe geringer als die maximale Paketgröße des Endpunkts) empfangen wird. Diese Methode darf nur auf einem OUT- oder Control-Endpunkt aufgerufen werden.
  Das Lesen wird ebenso wie das Schreiben asynchron durchgeführt. Auch hierbei kann nach dem Abschluss der Übertragung eine optionale Callback-Funktion aufgerufen werden. Die „USB_Read“ Methode wird in der „CDCDSerialDriver_Read“ Funktion verwendet, um die gesendeten Daten vom Host zu lesen. In diesem Test Programm wird eine Callback-Funktion für die „USB_Read“ Methode verwendet. Diese Callback-Funktion heißt „UsbDataReceived“ und gibt die Größe der empfangenen Daten in Byte zurück um Fehler bei der Übertragung erkennen zu können.

4.7.2.7 Nachrichten Handler

Der graue Bereich in Abbildung 4-13 zeigt den Nachrichten Handler. Der Nachrichten Handler prüft zuerst ob die empfangenen Daten eine gültige Nachricht ist. Erkennt der Handler eine ungültige Nachricht, dann schickt er eine Fehler Nachricht („microMsgError-NachrichtenHandler“) an den Host. Wenn der Host diese Nachricht empfängt, sendet er die gerade gesendete Nachricht erneut an den Mikrocontroller.

Je nachdem welche, der folgenden Nachrichten-IDs erkannt wird, werden verschiedene Schritte abgearbeitet. Es gibt folgenden Nachrichten-IDs:

- microMsgIDErrorDataReceive: Wenn diese Nachrichten ID erkannt wird, sendet der Mikrocontroller die vorherige Nachricht erneut an den Host.

- microMsgIDAckHostRedy: wenn der Mikrocontroller diese Nachrichten ID empfängt, sendet der Mikrocontroller die Nachricht „microMsgAckMicroReady“ an den Host. Das heißt, dass der Mikrocontroller in Betrieb ist und bereit für den nächsten Schritt.

- microMsgIDLedConfig: nach dem Empfang dieser Nachricht wird der Nachrichten-handler mithilfe der von der „Led_Configure“ Funktion angeforderten Led Nummer konfiguriert. Wie eine LED in dieser Funktion konfiguriert wird, wurde in Abschnitt 3.2 erläutert. Im nächsten Schritt wird mithilfe einer globalen Variablen („ledPeriod“) für die Wait-Funktion der angeforderte Takt für das Blinken gesetzt. Schließlich wird eine Acknowledge-Nachricht („micro-MsgLedConfigLength“) an den Host geschickt. Dadurch weiß der Host, dass die angeforderte LED und der Takt konfiguriert ist und der Mikrocontroller bereit steht für den nächsten Schritt.

- microMsgIDLedToggLedStart: wenn der Mikrocontroller diese Nachrichten-ID empfängt, wird die LED mit der angeforderten Anzahl zum Blinken gebracht. In diesem Schritt werden zwei „Acknowledge“ Nachrichten an den Host gesendet. Die erste Nachricht („microMsgAckLedToggleStart“) zeigt dem Host, dass die LED blinkt. Die

zweite Nachricht („microMsgAckLedToggleFinish“) zeigt dem Host, dass die LED nicht mehr blinkt und die Host Anforderung für den Mikrocontroller ist somit abgearbeitet.

4.7.3 Implementierung des Testprogramms auf dem PC

Das Testprogramm auf dem Host hat einen analogen Verlauf und eine Architektur wie das Testprogramm auf dem Mikrocontroller. In Abbildung 4-15 wird das Flussdiagramm der Systemschleife des Testprogramms auf dem Host gezeigt. Im folgenden Abschnitt wird, das Flussdiagramm genauer erklärt.

Die Klasse „CISerial“ öffnet den USB Port als eine serielle Schnittstelle. Mithilfe der „initializeCom()“ Methode wird das USB Device initialisiert. Die Member Variable „deviceStatus“ der „CIUsb“ Klasse gibt den USB Device Zustand zurück. Wenn das Device Status auf „Connected“ steht, wird eine Acknowledge-Nachricht (microMsgAckHostReady) durch die „serialUsbWriteStr“ Methode der Klasse „CIUsb“ an den Mikrocontroller gesendet. Dadurch bekommt der Mikrocontroller die Information, dass das Hostprogramm jetzt in Betrieb ist und auf die nächste Nachricht wartet. Das Anwendungsprogramm empfängt die Nachrichten über den USB Port mit Hilfe die „serialUsbReadStr()“ Methode.

Die grauen Bereiche in Abbildung 4-15 zeigen den Nachrichten Handler. Wenn eine Nachricht gelesen wird, wird diese zunächst vom Nachrichtenhandler empfangen und verarbeitet. Wenn der Nachrichtenhandler eine ungültige Nachricht erkennt, sendet er eine Fehlernachricht („msgErrorUsbDeviceData-Receive“) an den Mikrocontroller. Dadurch erkennt der Mikrocontroller den fehlerhaften Transfer und kann die Nachricht erneut senden. Erkennt der Nachrichtenhandler eine gültige Nachricht, werden die folgenden Schritte, je nach Nachrichten ID, ausgeführt. Es gibt die folgenden Nachrichten IDs:

- msgErrorUsbDataReceive: Wenn diese Nachrichten ID erkannt wird, sendet der Host die vorherige Nachricht erneut an den Mikrocontroller.
- microMsgIDAckMicroReady: wenn der Host diese Nachrichten ID empfängt, sendet er eine Kontrollnachricht („CINachrichtenSetup“) an den Mikrocontroller. Mit dieser Nachricht erkennt der Mikrocontroller, welche der LED mit welchem Takt angesprochen werden soll.
- microMsgIDAckLedConfigLength: nach dem Empfang dieser Nachricht sendet der Nachrichtenhandler eine Kontrollnachricht („CINachrichtenLesStart“). Diese Nachricht zwingt den Mikrokontroller die LED mit der angeforderten Anzahl blinken zu lassen.
- microMsgIDAckLedToggleFinish: wenn der Host diese Nachricht empfängt, endet das Programm auf dem Host.

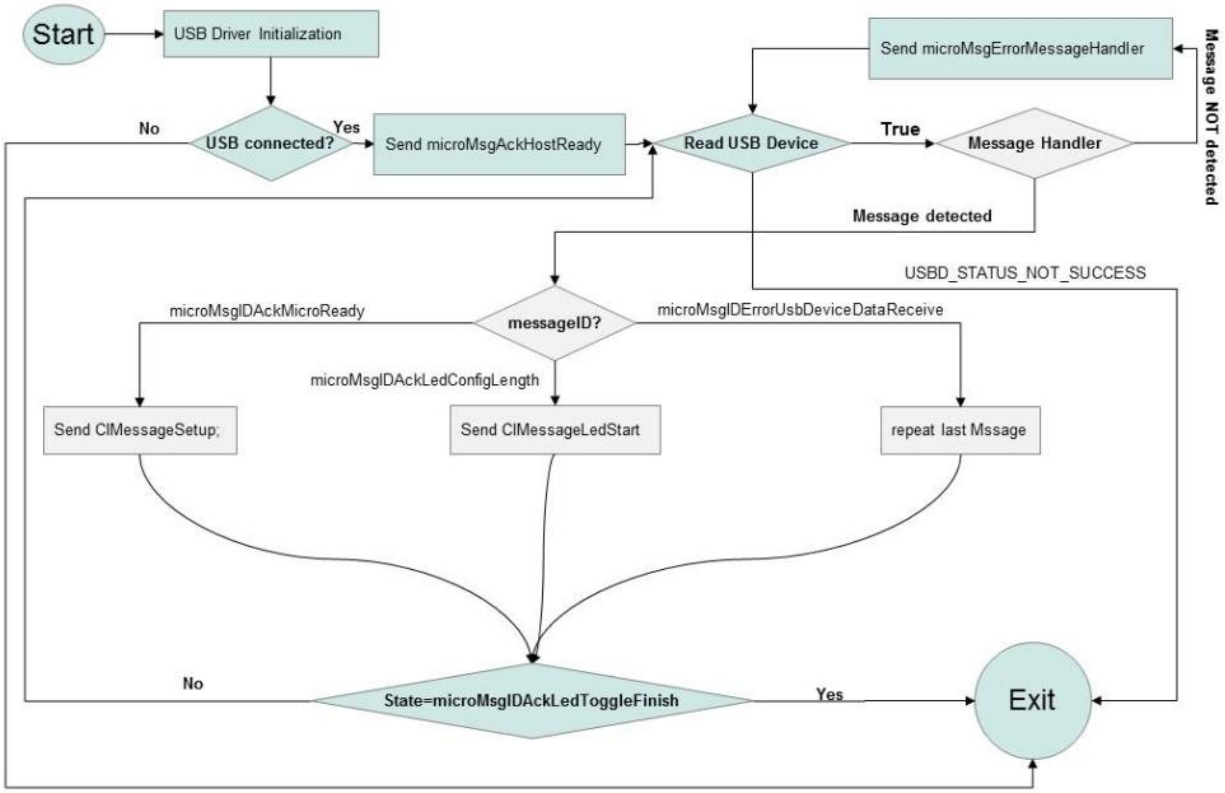

Abbildung 4-15: Flussdiagramm der System-Schleife des Testprogramms auf dem Host (User Applikation)

5 CAN

In diesem Kapitel werden zuerst die allgemeinen Grundlagen des Controller Area Network (CAN) sowie dessen Übertragungsverfahren, die Übertragungsart und mögliche Leitungslängen, die Topologie und der Frameaufbau erläutert. Im Anschluss wird das CAN Peripheriegerät des AT91SAM9263 genauer erklärt.

5.1 Grundlagen

Das Controller Area Network (CAN) ist ein serieller Kommunikations-Bus. Der CAN-Bus wurde entwickelt um eine einfache, effiziente und stabile Kommunikation für Vernetzung von Steuergeräten in Automobilen (In-Vehicle-Netzwerke) zu bieten. Es wurde von der Robert Bosch GmbH im Jahr 1983 entwickelt. Im Jahr 1987 wurden die ersten CAN Controller Chips von Intel (82526) und Philips (82C200) veröffentlicht. In den 1990er Jahren entwickelte Bosch die CAN-Spezifikation für die Standardisierung nach ISO-Standard für CAN (11898). Die erste Veröffentlichung des Standards war im Jahr 1993 [Dav07].

5.1.1 Physikalische Bitübertragung

Die serielle Bitübertragung beim CAN-Bus wird im Allgemeinen über zwei verdrehte Leitungen (twisted pair) realisiert. Wie in Abbildung 5-1 gezeigt, wird die Leitung an jedem Ende mit einem 120 Ohm Widerstand abgeschlossen. Die Widerstände verhindern eine mögliche Reflexion des Bitstromes an dem Leitungsende [Har11]. Der rezessive Pegel beträgt 2,5 V auf beiden Leitungen. Der dominante Pegel 3,5 V ist für CAN_H und 1,5 V für CAN_L.

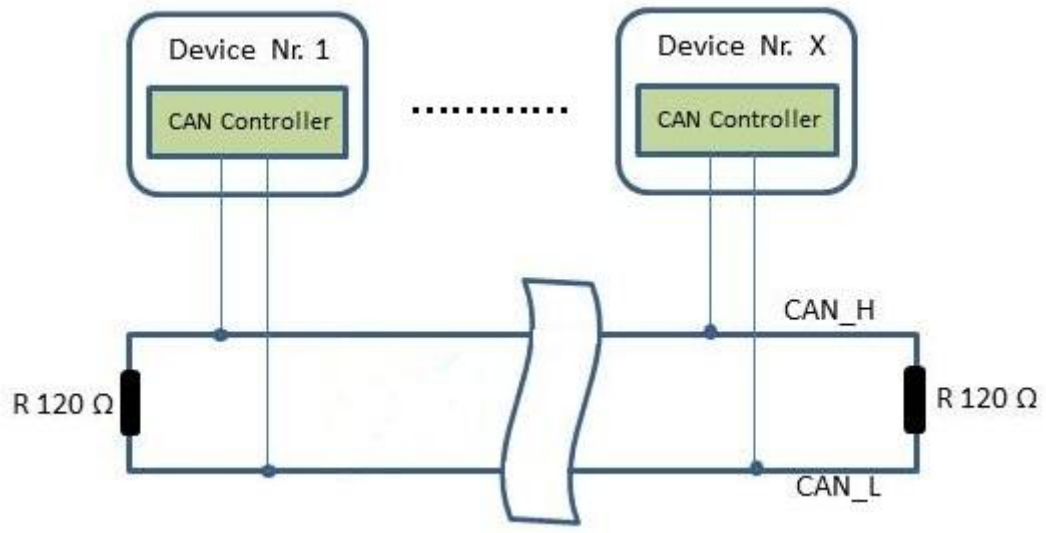

Abbildung 5-1: Das Prinzip eines Controller Area Network

Der CAN-Bus verwendet das CSMA/CR[Nat98] (Carrier Sense Multiple Access / Collision Resolution) Verfahren, um eine Kollision aufzulösen. Die Daten werden nach dem NRZ-L Verfahren kodiert und mithilfe der zyklischen Redundanzprüfung (CRC) wird die Korrektheit eines Pakets geprüft. Für die Synchronisierung zwischen Busteilnehmern wird das Verfahren des Bitstopfens (bit stuffing) verwendet. Alle Bus Teilnehmer eines CAN Busses sind gleichberechtig (Multi Master Prinzip), das heißt jeder Teilnehmer kann auf den Bus zugreifen sobald der Bus frei ist.

5.1.2 Übertragungsrate und Leitungslänge

Es gibt zwei Arten von Bussen, den Highspeed- und den Lowspeed Bus. Die maximale Übertragungsrate beträgt für den Highspeed-Bus 1 Mbit/s und für den Lowspeed Bus 125 kbit/s. die maximale Leitungslänge hängt von der Übertragungsrate ab, siehe hierzu auch Tabelle 5-1).

Tabelle 5-1: Abhängigkeit zwischen maximaler Leitungslänge und Bitrate [Mes]

Bitrate	Kabellänge
10 kbits/s	6,7 km
20 kbits/s	3,3 km
50 kbits/s	1,3 km
125 kbits/s	530 m
250 kbits/s	270 m
500 kbits/s	130 m
1 Mbits/s	40 m

5.1.3 Topologie

Im folgenden Abschnitt werden zwei Topologien von CAN vorgestellt:

- Linientopologie (Bustopologie): Bei dieser Topologie wird kein zentrales Steuerelement zur Kommunikation benötigt. Alle CAN Knoten werden mit einer kurzen Stichleitung mit der Busleitung verbunden (siehe Abbildung 5-1). Alle Knoten können die auf dem Bus übertragenen Informationen empfangen. Trotzt Ausfall eines CAN Knoten können die anderen Knoten weiterhin Information austauschen.
- Sterntopologie: mithilfe einer zentralen Station ist es möglich, eine Sterntopologie aufzubauen. Wenn die zentrale Station ausfällt, werden keine weiteren Daten mehr weitergeleitet. Bei dieser Aufbaustruktur ist die Berechnung des Leitungswellen-wiederstandes deutlich aufwendiger.

In dieser Diplomarbeit wird die eben vorgestellte Bustopologie verwendet.

5.1.4 Frame Typen

Der Nachrichten-Transfer innerhalb des CANs wird durch die vier folgenden Frame-Typen gesteuert: [Bos91]

- Data Frame: dieses Frame überträgt die Daten von einem Sender zum Empfänger.
- Remote Frame: Ein Ziel Knoten kann Daten von der Quelle beantragen, wenn er einen Remote Frame mit einem passenden Identifier sendet.
- Error Frame: dieses Frame wird von jedem Gerät bei Erkennung eines Fehlers gesendet.
- Overload Frame: Ein Overload Frame wird verwendet, um eine zusätzliche Verzögerung zwischen Daten-Frame und Remote-Frame bereitzustellen.

Daten-Frame und Remote-Frame sind immer vom vorhergehenden Frame durch ein Inter-Frame getrennt.

5.1.5 Data-Frame Aufbau

CAN Nachrichten werden in einem so genannten CAN Frame verpackt und gesendet. Ein Data-Frame hat zwei verschiedene Spezifikationen. Die beiden folgende Spezifikationen unterscheiden sich nur in der Länge des Identifiers: [Bos91]

- Data Frame mit CAN-Spezifikation 2.0A: dieses Data Frame hat 11 Bit für den Identifier reserviert. Dadurch ermöglicht dieses Identifier Feld 2032 verschiedene logische Adressen zu kodieren. Dieses CAN Frame wird als Standard Frame bezeichnet. Abbildung 5-2 zeigt den Aufbau eines Standard Frames.

Start	Identifier	RTR	DIE	r0	DLC	DATA	CRC	ACK	EOF+IFS
1 Bit	11 Bit	1 Bit	1 Bit	1 Bit	4 Bit	0 bis 4 Byte	15 Bit	2 Bit	10 Bit

Abbildung 5-2: Standard Frame

- Data Frame mit CAN Spezifikation 2.0B: dieses Data Frame hält 29 Bit für den Identifier bereit. Durch diese Identifier Erweiterung werden Kodierungen von logischen Adressen bis zu 536.870.912 ermöglicht. Dieser CAN Frame kann man demnach auch als Extended Frame bezeichnet. Abbildung 5-3 zeigt den Aufbau des Extended Frames.

Start	Identifier	SRR	IDE	Identifier	RTR	r0	R1	DLC	DATA	CRC	ACK	EOF+IFS
1 Bit	11 Bit	1 Bit	1 Bit	18 Bit	1 Bit	1 Bit	1Bit	4 Bit	0 bis 4 Byte	15 Bit	2 Bit	10 Bit

Abbildung 5-3: Extended-Frame

Im Folgenden werden alle Felder des Data Frames erklärt:[Bos91] [Mes]

- Start of Frame: dieses Feld ist dient der Synchronisation zwischen den CAN Devices in einem CAN-Bus und besteht aus einem dominant wirkenden Bit.
- Identifier: das Feld besteht aus 11 oder 29 Bit je nach Data-Frame Typ (Standard oder Extended) und dient der Objektidentifikation.
- RTR (Remote Transmission Request Bit): das RTR-Bit muss in Daten-Frames dominant wirken. innerhalb eines Remote Frames wirkt das RTR-Bit hingegen rezessiv.
- IDE (Identifier Extention): wenn das Bit dominant wirkt, folgen im Data Frame weitere Daten.
- r0, r1: dieser Bit ist für zukünftige Erweiterungen reserviert.
- DLC (Data Length Code): mithilfe dieser 4 Bit wird die Länge des nachfolgenden Daten Feldes codiert.
- Data: dieses Feld enthält die eigentlichen Daten der Nachricht.
- CRC: dieses Feld beinhaltet die Prüfsumme für das vergangene Feld und wird für die Fehlererkennung verwendet.
- ACK: wenn ein Daten-Frame von einem anderen Teilnehmer korrekt empfangen wird, wird der Empfänger dieses Feld dominieren.
- EOF (End of Frame): Das Feld besteht aus 7 Bit und dient der Erkennung des Endes von einem Frame.
- IFS (Inter Frame Space): Das Feld hat 3 Bit und zeigt den Übertragungszeitraum für eine korrekt empfangene Nachricht.
- SRR (Substitute Remote Request): dieses Feld existiert für den Extended Frame und hat keine weitere Bedeutung.

5.2 CAN auf dem Mikrocontroller AT91SAM9263

5.2.1 Überblick

Der CAN-Controller bietet alle benötigten Funktionen, um das serielle Kommunikations-protokoll der Robert Bosch GmbH zu implementieren. Die CAN-Spezifikation wird durch ISO/11898A (v2.0 Teil A und v2.0 Teil B, siehe Abschnitt 5.1.5), für hohe Geschwindigkeiten und durch ISO/11519-2 für niedrige Geschwindigkeiten, definiert. Der CAN-Controller ist in der Lage, alle Arten von Frames (Data, Remote, Error und Overload) zu behandeln und erreicht dabei eine Datenrate von bis zu 1 MBit/s.

Abbildung 5-4 zeigt das CAN Block Diagramm. Der CAN-Controller greift durch vorkonfigurierte Register auf den Bus zu. 16 unabhängige Nachrichten Objekte (Mailboxen) sind implementiert. Jede Mailbox kann als Empfangspuffer (auch nicht-konsekutiven Puffer) programmiert werden. Für den Empfang von Nachrichten können eine oder mehrere Nachrichten Objekte, ohne Teilnahme an der Puffer-Funktion, maskiert werden. Ein Interrupt wird erst erzeugt, wenn der Puffer voll ist. Gemäß der Mailbox Konfiguration kann die

Nachricht im Register von CAN-Controllern gesperrt werden, bis die Anwendung die Nachricht erkennt, oder diese Nachricht kann von neuen empfangenen Nachrichten verworfen werden, je nach Implementierung.

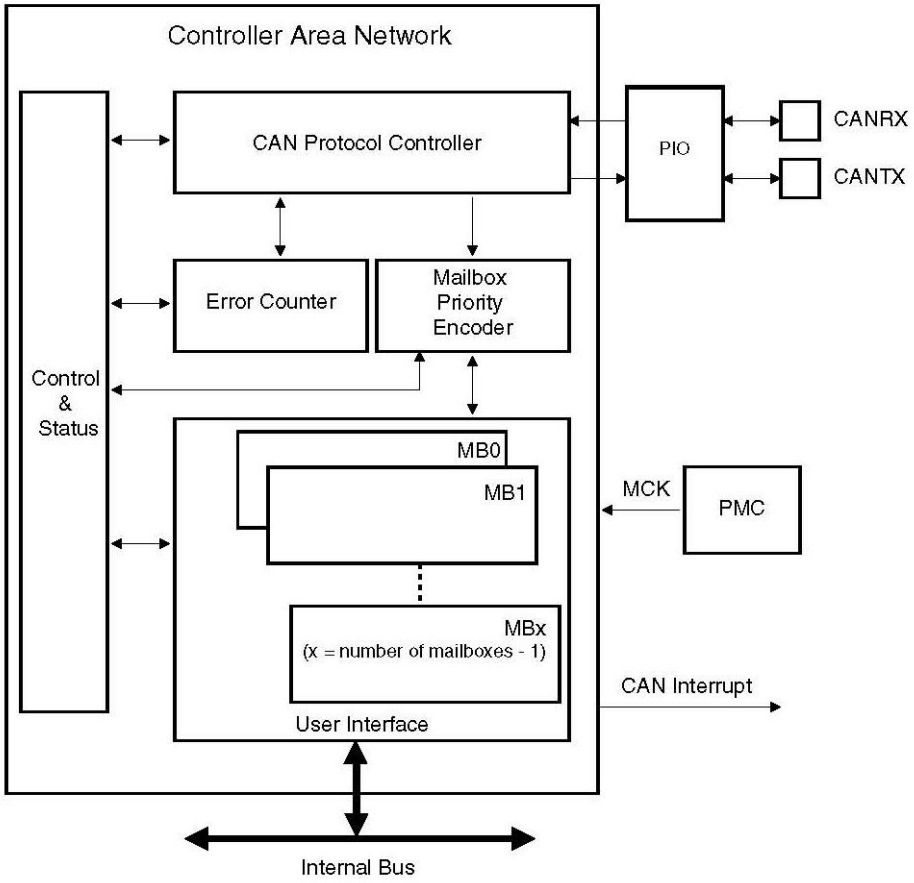

Abbildung 5-4: CAN Block Diagramm[ATM11, S. 650]

Jede Mailbox kann für die Übertragung einzeln programmiert werden. Mehrere Mailboxen können im Empfangsmodus, in der gleichen Zeit aktiviert werden. Eine Priorität kann für jede Mailbox unabhängig definiert werden.

Ein interner 16-Bit-Timer wird verwendet, um jede empfangene und gesendete Nachricht zu stempeln. Dieser Timer beginnt zu zählen, sobald der CAN-Controller aktiviert wird. Er kann durch die Anwendung aber jederzeit zurückgesetzt werden, oder automatisch nach dem Empfangen einer Nachricht in dem letzten Postfach im Time Trigger-Modus.

Der CAN-Controller bietet optimierte Funktionen, um das Time Trigger Communication (TTC)-Protokoll unterstützen.

5.2.2 CAN Controller Merkmale

5.2.2.1 Mailbox Organisation

Das CAN-Modul verfügt über 16 Puffer, die auch Kanäle oder Mailboxen genannt werden. Ein Identifier, der einem CAN Identifier entspricht, wird für jede aktive Mailbox definiert. Nachrichten-IDs können mit dem Standard-Frame oder dem Extended-Frame übereinstimmen. Diese Definition des Identifiers kann während der ersten CAN Initialisierung definiert werden, aber durch eine neue dynamische Konfiguration zu einem späteren Zeitpunkt, kann die Mailbox einen neuen Nachrichten Typ behandeln. Mehrere Mailboxen können auch mit der gleichen ID konfiguriert werden.

Jede Mailbox kann für den Empfangsbetrieb oder den Sendebetrieb unabhängig voneinander konfiguriert werden. Der Typ von der Mailbox wird in dem „MOT“ Feld des „CAN_MMRx“ Registers [ATM11 S.696] definiert.

5.2.2.1.1 Nachrichtenannahmeverfahren (Maskierung)

Wenn das „MIDE“ Feld in dem CAN_MIDx Register eingestellt ist, kann die Mailbox einen Extended-Frame Identifier lesen oder im anderen Fall das Standard-Frame.

Das Nachrichtenannahmeverfahren wird mithilfe eines Flussdiagramms, wie in Abbildung 5-5 gezeigt erstellt. Sobald eine neue Nachricht empfangen wird, wird seine ID mit dem Wert CAN_MAMx maskiert. Ebenso wird der vorgegebene Wert von CAN_MIDX mit dem CAN_MAMx Wert maskiert. Das Ergebnis von beiden Maskierungen wird verglichen, wenn dieses gleich ist, wird die Nachrichten ID in das CAN_MIDx Register kopiert[ATM11].

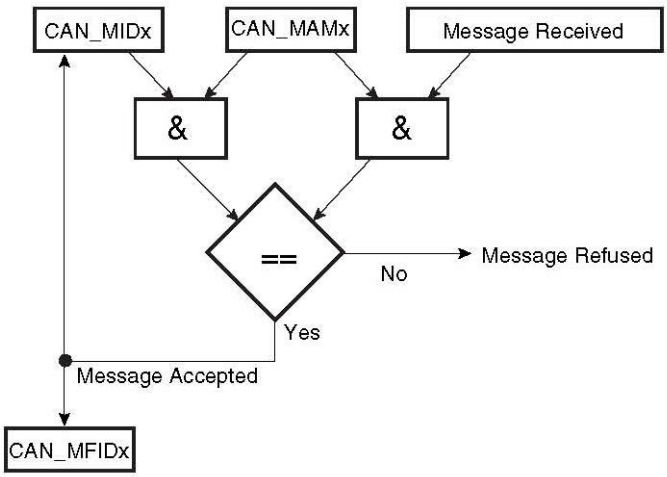

Abbildung 5-5: Flussdiagramm des Nachrichtenannahmeverfahrens [ATM11, S. 653]

Ein Beispiel: Die folgenden Nachrichten IDs aus Tabelle 5-2 werden durch dasselbe Postfach behandelt:

Tabelle 5-2: Beispiel für Familien IDs

Nachrichten ID Name	Nachrichten ID (Hex)	Nachrichten ID (Binär)
ID6	0x666	110 0110 0110
ID7	0x667	110 0110 0111
ID14	0x66E	110 0110 1110
ID15	0x66F	110 0110 1111

Die Register CAN_MIDx und CAN_MAMx der Mailbox x müssen mit den entsprechenden Werten, die den folgenden Regeln entsprechen, initialisiert werden:

- Wenn die Ziffern der Nachrichten IDs in Tabelle 5-2 mit einander verglichen werden, sind manche Ziffern in alle Nachrichten IDs fest (ohne Änderung) und manche Ziffern in allen Nachrichten IDs verändert. Die festen Ziffern sind mit schwarzer Farbe, die geänderten mit roter Farbe dargestellt.
- CAN_MIDx Initialisierung: die schwarzen Ziffern werden ohne Änderung an die gleiche Stelle in CAN_MIDX geschrieben und anstatt roten Ziffern wird an gleicher Stelle eine „0“ geschrieben. CAN_MIDX sieht dann wie folgt aus:
- CAN_MIDx=110 0110 0110=0x666 h
- CAN_MAMx Initialisierung: anstatt schwarzer Ziffern werden an gleicher Stelle Einsen in CAN_MAMx geschrieben und anstatt roter Ziffern wird an gleicher Stelle eine 0 geschrieben. CAN_MAMx sieht dann wie folgt aus:
- CAN_MAMx=111 1111 0110=0x7f6 h

Wenn CAN_MIDX und CAN_MAMx für diese Beispiel initialisiert sind, sieht man in Tabelle 5-3 das Nachrichtenannahmeverfahren für einige Nachrichten IDs durchgeführt:

Tabelle 5-3: Beispiel für das Nachrichtenannahmeverfahren

Nachricht Name	Nachricht ID (Hex)	Nachricht ID (Binär)	CAN_MAMx	CAN_MIDx	Nachricht-Receive & CAN_MAMx	CAN_MAMx & CAN_MIDx	Accept-ed?	CAN_MFIDX
ID0	0x660	110 0110 0000	111 1111 0110	110 0110 0110	110 0110 0000	110 0110 0110	No	---------
ID6	0x666	110 0110 0110	111 1111 0110	110 0110 0110	110 0110 0110	110 0110 0110	Yes	00
ID7	0x667	110 0110 0111	111 1111 0110	110 0110 0110	110 0110 0110	110 0110 0110	Yes	01
ID8	0x668	110 0110 1000	111 1111 0110	110 0110 0110	110 0110 0000	110 0110 0110	No	---------
ID13	0x66D	110 0110 1101	111 1111 0110	110 0110 0110	110 0110 0100	110 0110 0110	No	---------
ID14	0x66E	110 0110 1110	111 1111 0110	110 0110 0110	110 0110 0110	110 0110 0110	Yes	10
ID15	0x66F	110 0110 1111	111 1111 0110	110 0110 0110	110 0110 0110	110 0110 0110	Yes	11

Die blaue Zeile in Tabelle 5-3 zeigt die Nachrichten IDs, die akzeptiert und verarbeitet werden, an. In der orangenen Zeile sind die Nachrichten IDs die nicht akzeptiert werden.

Wenn Mailbox x eine Nachricht mit ID14 erhält, wird nach dem Nachrichtenannahmeverfahren die Nachricht Akzeptiert. CAN_MFIDx und CAN_MIDx sind wie folgt belegt:

CAN_MIDx=110 0110 1110=0x66E h

CAN_MFIDx=000 0000 0010=0x002 h

Wenn die Anwendung einen Handler für jede Nachrichten-ID einrichtet, kann sie ein Array von Zeigern auf Funktionen definieren: [ATM11]

```
void (*pHandler[8])(void);
```

Wenn eine Nachricht empfangen wird, kann der entsprechende Handler mit dem Wert des CAN_MFIDx Registers aufgerufen werden und es besteht keine Notwendigkeit, die maskierten Bits zu überprüfen:[ATM11]

```
unsigned int MFID0_register;

MFID0_register = Get_CAN_MFID0_Register();

// Get_CAN_MFID0_Register() returns the value of the CAN_MFID0 register

pHandler[MFID0_register]();
```

5.2.2.1.2 Mailbox in Empfangsbetrieb

Wenn das CAN Modul eine Nachricht empfängt, sucht es nach der ersten verfügbaren Mailbox mit der niedrigsten Zahl und vergleicht die empfangene Nachrichten-ID mit der Mailbox-ID. Wenn eine passende Mailbox gefunden wird, dann wird die Nachricht in das Daten-Register gespeichert. Eine Mailbox im Empfangsbetrieb hat die folgenden Modi:[ATM11]

- **Empfangs Modus:** Die erste empfangene Nachricht wird im Mailbox-Daten-Register gespeichert. Daten sind bis zur nächsten Übertragungsanfrage erhältlich.
- **Empfang mit überschriebenem Modus:** Die letzte empfangene Nachricht wird im Daten-Register der Mailbox gespeichert. Die nächste Nachricht überschreibt immer die vorherige. Die Anwendung kann prüfen, ob die neue Nachricht währende dem Lesen des Daten Registers nicht überschrieben wurde.
- **Consumer-Modus:** In diesem Modus wird nach jeder Übertragungsanfrage automatisch ein Remote Frame gesendet. Die erste empfangene Antwort wird in den Daten-Registern der entsprechenden Mailbox gespeichert.

Es können auch mehrere Mailboxen verkettet werden, um einen Puffer zu erhalten. Alle Mailbox außer der letzten müssen mit der gleichen ID im Empfangs Modus konfiguriert sein, die letzte Mailbox sollte im Überschreibmodus konfiguriert werden. Dadurch kann mit der letzten Mailbox ein Pufferüberlauf detektiert werden.

5.2.2.1.3 Mailbox im Sendebetrieb

Eine Mailbox im Sendebetrieb hat folgenden Modus:[ATM11]

- **Sende Modus:** Bei der Übertragung einer Nachricht werden die Länge der Nachricht und die Daten zu der Sende-Mailbox mit der richtigen ID geschrieben. Jede Sende-Mailbox wird eine Priorität zugewiesen. Der Regler sendet die Nachricht mit der höchsten Priorität zuerst. Diese Nachrichtenpriorität wird mithilfe des „PRIO“ Feldes im CAN_MMRx eingestellt.
- **Producer Modus:** In diesem Modus werden, sobald ein Remote Frame empfangen wird, automatisch die Mailbox-Daten gesendet.

5.2.2.2 CAN Bit Timing Konfiguration

Alle CAN Teilnehmer auf einem CAN-Bus müssen dieselbe Bitrate und Bit Länge haben. Bei verschiedenen Taktfrequenzen der einzelnen CAN Teilnehmer, muss die Bit- Rate als Zeit Segment angepasst werden.

Die CAN-Protokoll-Spezifikation partitioniert den nominale Bit-Time in vier verschiedene Segmente, siehe Abbildung 5-6:[ATM11]

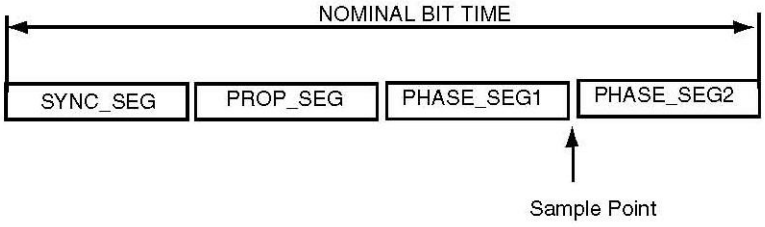

Abbildung 5-6: Partition des CAN-Bit-Time [ATM11, S. 656]

- TIME QUANTUM: Das Time-Quantum (TQ) ist eine feste Zeiteinheit und aus der MCK Periode abgeleitet. Die Gesamtzahl der Time-Quantums in der programmierten Bit-Time liegt zwischen 8-25.
- SYNC SEG (SYNChronization Segment): Dieser Teil der Bit-Time wird verwendet, um die verschiedenen Knoten auf dem Bus zu synchronisieren. Es wird eine Kante erwartet, die in diesem Segment liegt. Es ist ein TQ lang.
- PROP SEG (PROPagation Segment): Dieser Teil der Bit-Time wird verwendet, um die physikalischen Verzögerungszeiten innerhalb des Netzwerks zu kompensieren. Es ist das Doppelte der Summe der Signal-Laufzeit auf dem Bus (Eingangskomparator Verzögerung, und Ausgangstreiber Verzögerung). Es hat die Länge von 1 bis zu 8 TQ.
- PHASE SEG1, PHASE SEG2 (PHASE Segment 1 und 2): Die Phasen-Buffer-Segmente werden verwendet, um Randphasenfehler zu kompensieren. Diese Segmente können durch Neu-synchronisierung verlängert (PHASE SEG1) oder verkürzt werden (PHASE SEG2).

Phase-Segment2 Länge muss mindestens so lang wie die Information-Processing-Time (IPT) sein und darf nicht länger als die Phase Segment 1 sein.

- Information-Processing-Time (IPT): Die Information-Processing-Time (IPT) ist die erforderliche Zeit, um die Bit-Ebene eines Sample-Bit festzulegen. Die IPT beginnt an dem Sample-Punkt und wird in TQ gemessen. Die IPT Länge ist für den Atmel CAN Controller 2 TQ.
- SAMPLE Punkt: Der Sample Punkt ist der Zeitpunkt, an dem die Bus-Ebene als der Wert des jeweiligen Bits gelesen und interpretiert wird. Er ist am Ende des PHASE_SEG1 lokalisiert.
- SJW (ReSynchronization Jump Width): Dieses Segment definiert die Grenze für den Umfang der Verlängerung oder Verkürzung der Phasen-Segmente.

In dem CAN-Controller wird die Länge eines Bits über den CAN-Bus durch die Parameter BRP, PROPAG, PHASE1 und PHASE2 bestimmt.

$$t_{BIT} = t_{CSC} + t_{PRS} + t_{PHS1} + t_{PHS2}$$

Das Time-Quantum wird wie folgt berechnet:[ATM11]

$$t_{CSC} = (BRP + 1)/MCK$$

$$t_{PRS} = t_{CSC} \times (PROPAG + 1)$$

$$t_{PHS1} = t_{CSC} \times (PHASE1 + 1)$$

$$t_{PHS2} = t_{CSC} \times (PHASE2 + 1)$$

Die SJW Zeit wird wie folgt berechnet:[ATM11]

$$t_{SJW} = t_{CSC} \times (SJW + 1)$$

Beispiel: Bit-Timing Berechnung für CAN mit 500 Kbit/s Baudrate[ATM11, S. 658]

```
MCK = 48MHz
CAN baudrate= 500kbit/s => bit time= 2us
Delay of the bus driver: 50 ns
Delay of the receiver: 30ns
Delay of the bus line (20m): 110ns
TCSC= 1 time quanta= bit time/16 = 125 ns => BRP=(TCSC x MCK)-1 = 5
TPRS= 2 * (50+30+110)ns =380 ns =3 TCSC => PROPAG=(TPRS/TCSC)-1 = 2
TPHS1+TPHS2= bit time - Tcsc - TPRS= (16 - 1 - 3) TCSC
TPHS1+TPHS2=12 TCSC
TPHS2=TPHS1+TCSC
TPHS1=TPHS2=(12/2) Tcsc=6 TCSC => PHASE1=PHASE2=(Tphs1/TCSC)-1=5
TSJW=Min(4 Tcsc,Tphs1)=4 TCSC => SJW=(Tsjw/TCSC)-1=3
CAN_BR = 0x00053255
```

6 CAN-USB Adapter Implementierung

In diesem Kapitel wird die Implementierung eines CAN-USB Adapters anhand eines Beispielszenarios veranschaulicht. Die USB Funktionen, das Framework und die Bibliotheken, die in Kapitel 4 erwähnt sind, werden in dieser Implementierung verwendet. Die für den CAN Controller benötigten Funktionen, wurden in Kapitel 5 implementiert und erklärt.

6.1 Beispielszenario

In diesem Szenario wird der PC Nr.1 und ein Mikrocontroller durch die CAN Schnittstelle über einen CAN-Bus angeschlossen (siehe Abbildung 6-1). Der PC Nr.2 wird durch eine USB Schnittstelle an dem USB Device des Mikrocontrollers angeschlossen. Der PC Nr.3 wird als Monitoring Gerät an den CAN-Bus angeschlossen. Da die PCs direkt keine eigene CAN Schnittstelle besitzen, wird ein USB-CAN Adapter verwendet. Dieser Adapter ermöglicht es, dass ein PC über einen CAN-Bus verbunden werden kann.

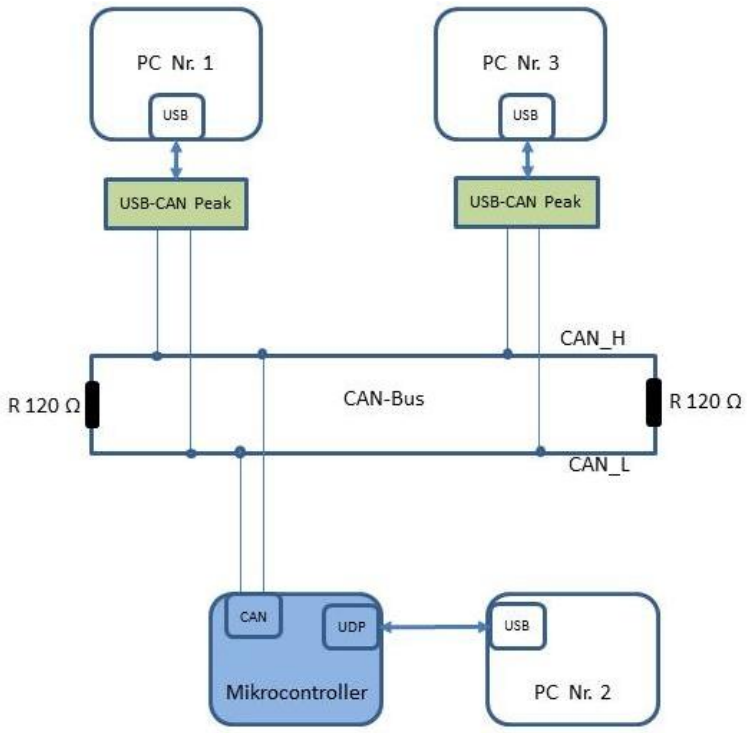

Abbildung 6-1: Schema des Beispielszenarios

Der PC Nr.1 simuliert einen CAN Teilnehmer, der CAN-Pakete auf dem CAN-Bus senden oder empfangen kann. Der PC Nr.2 simuliert einen USB Host, der durch seine USB Schnittstelle USB-Pakete senden oder empfangen kann. Auf dem Mikrocontroller läuft das CAN-USB Adapterprogramm, dessen Implementierung in diesem Kapitel erläutert wird. Das CAN-USB

Adapterprogramm hat die Aufgabe, dass gesendete CAN-Pakete über die CAN Schnittstelle von PC Nr.1 zu einem USB-Paket konvertiert werden und an PC Nr. 2 durch die USB-Device Schnittstelle weitergeleitet werden. Die andere Richtung ist ebenso möglich, also alle gesendeten USB-Pakete des PCs Nr.2 zu einem CAN-Paket konvertieren und über den CAN-Bus an PC Nr.1 weiterzuleiten. Auf dem PC Nr. 3 läuft das Programm PCAN-Viewer, mit dessen Hilfe kann über den PCAN-USB Adapter auf den CAN-BUS zugegriffen werden und alle gesendeten Daten eines anderen CAN-Bus Teilnehmers angezeigt werden.

6.2 PCAN-USB von Peak System

Dieses Interface ermöglicht es, dass ein PC über USB mit einem CAN-Bus verbunden werden kann. Die maximale Baudrate beträgt für dieses Device 1 MBit/s. Abbildung 6-2 zeigt das USB-CAN-Interface [PEA11].

Abbildung 6-2: PCAN-USB Device [PEA11]

Dieses Geräte ist für den Einsatz an einem PC und am Laptop geeignet ist. Das Gerät wird mit einem Treiber und dem Programm „PCAN-View“ vom Hersteller, Peak System, geliefert. Das Programm PCAN- View ermöglicht zum einen das Monitoring eines CAN Busses, und zum anderen auch das Senden und Empfangen einer CAN Nachricht über das PCAN-USB Gerät. Abbildung 6-3 zeigt einen Screenshot des Hauptfensters.

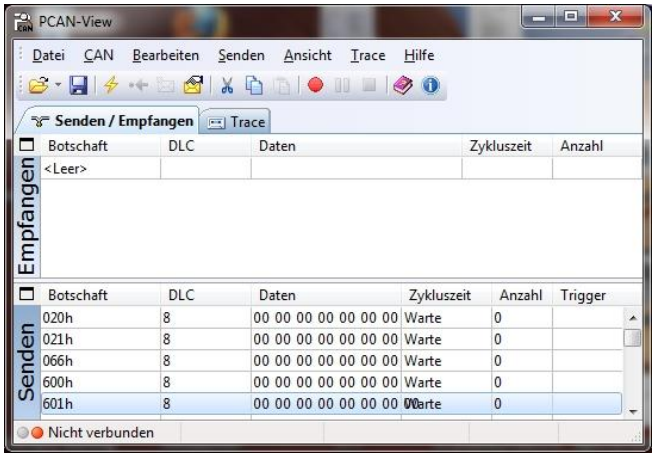

Abbildung 6-3: Hauptfenster von PCAN-View

In diesem Beispielszenario wird das Programm PCAN-View auf PC Nr.3 unter Windows installiert. Es ermöglicht, dass alle laufende CAN Nachrichten des CAN-Busses anschaulich gemacht werden. Dadurch kann man die Korrektheit der implementierten Programme auf den PCs Nr1 und Nr.2 sowie auf dem Mikrokontroller überprüfen.

Auf dem PC Nr.1 wird ein Programm implementiert, damit der PC als ein CAN-Bus Teilnehmer fungieren kann. Dafür wird ein PCAN-USB Gerät verwendet. Damit das implementierte Programm unter dem eingesetzten Linux Betriebssystem auf das Gerät zugreifen kann, wird der PCAN-USB Treiber („pcan.o“ oder „pcan.ko“) installiert. Ein Anwenderprogramm kann mit dem Treiber auf drei verschiedene Arten kommunizieren: [Pes11]

- ASCII-formatierte Daten können über eine „read()“ oder „write()“-Schnittstelle akzeptiert werden. Diese Daten enthalten Informationen aus empfangenen oder gesendeten Nachrichten oder über die Parameter der Channel Initialisierung.
- Mit eine so genannte „ioctl()“-Schnittstelle. Über diese Schnittstelle können Benutzerprogramme eine CAN Nachricht senden oder empfangen. Auch ist es möglich, Informationen über den Channel-Status zu erhalten und den CAN-Kanal zu initialisieren.
- Als Netzwerk-Gerät mit gängigen Netzwerk-Socket-Aufrufen. Diese Art von Interface ist eine Alternative zu den bisherigen. Es wird als „netdev“ Treiber-Schnittstelle bezeichnet.

Für weitere Information über Installation und Verwendung siehe [Pes11].

6.3 Handshake Prinzip für das Beispielszenario

Wie in Abbildung 6-4 zu sehen, wird für die Kommunikation zwischen PC Nr.1, PC Nr.2 und Mikrokontroller ein zwei Phasen Handshake implementiert. Im Folgenden wird das Handshake Prinzip auf PC Nr.1, PC Nr.2 und dem Mikrocontroller erklärt:

- **Programm auf PC Nr.1:** Das Programm auf PC Nr.1 sendet zuerst eine CAN Nachricht mit der CAN-ID 0x666 und CAN-Data 0x0000 auf den CAN-Bus, dann wartet es auf eine CAN Nachricht mit der CAN-ID 0x66F als Antwort. Wenn das Programm die CAN Nachricht mit der entsprechenden CAN-ID empfängt, liest es das CAN-Data Feld von der angekommenen Nachricht aus, inkrementiert dieses Data-Feld und schickt diese mit der CAN-ID 0x666 weiter über den CAN-Bus an den Mikrocontroller. Dieser Vorgang wiederholt sich in einer Schleife. Wenn Das Programm eine ungültige Nachricht bekommt, sendet es eine Error-Nachricht an den Mikrocontroller. Der Mikrocontroller leitet diese Fehlermeldung über USB an PC Nr.2 weiter, damit PC Nr. 2 die vorherige Nachricht erneut senden kann.
- **CAN-USB Adapter Programm auf dem Mikrocontroller:** Das CAN-USB Adapter Programm auf dem Mikrocontroller hat eine Main Schleife. In dieser Schleife prüft es, ob eine CAN-Nachricht oder eine USB-Nachricht vorhanden ist.

Wenn eine CAN-Nachricht vorhanden ist, liest es die CAN-ID- und das CAN-Data-Feld des CAN-Pakets aus und konvertiert die beiden Felder in ein USB-Paket. Dann sendet es das vorbereitete USB-Paket über den UDP an PC Nr.2.
Wenn eine USB-Nachricht vorhanden ist, liest es die Msg-ID- und das USB-Data-Feld des USB-Pakets aus und konvertiert analog zur Gegenrichtung die beiden Felder in ein CAN-Paket und sendet das so vorbereitete CAN-Paket über den CAN-Controller an PC Nr.1.

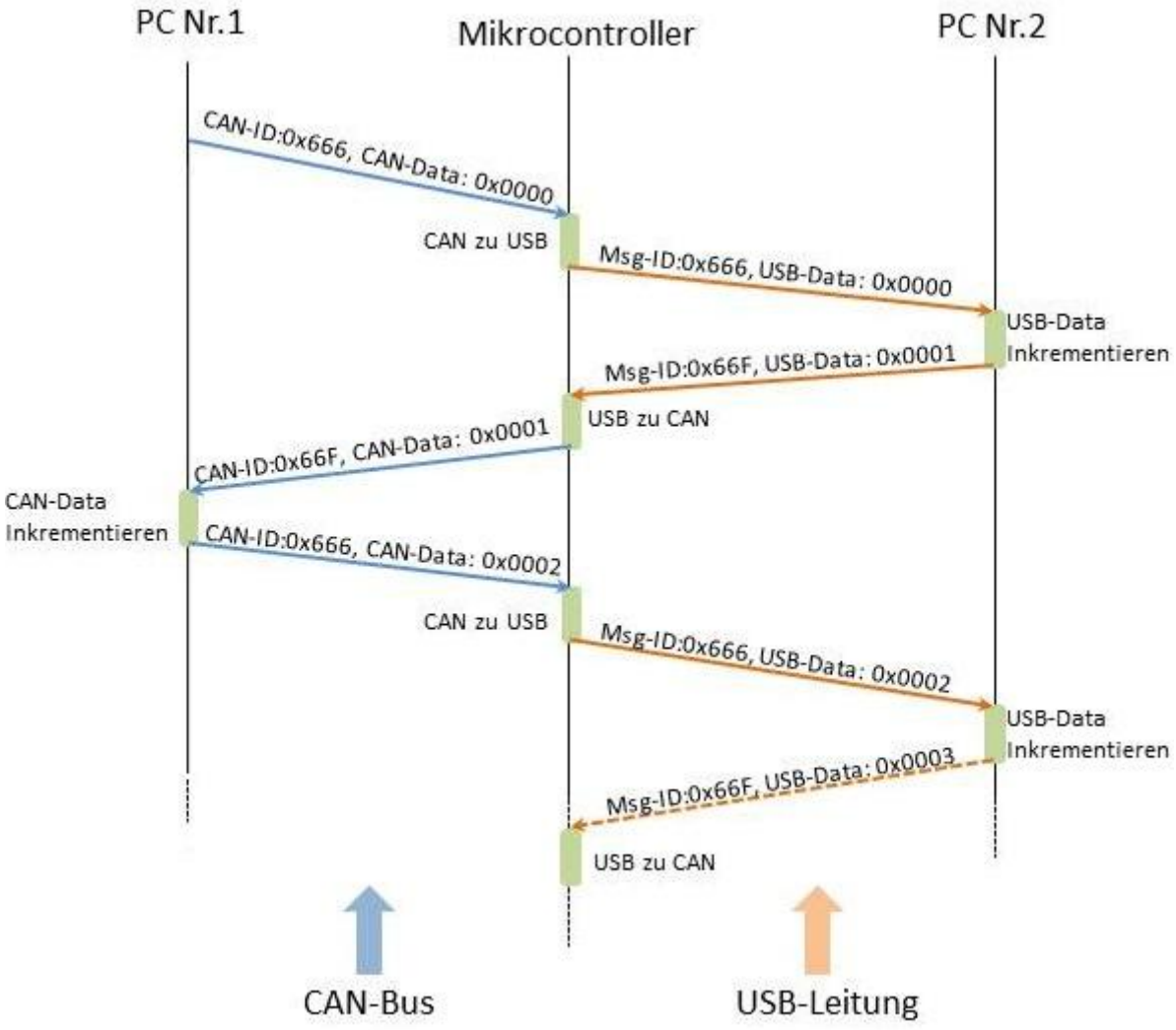

Abbildung 6-4: Handshake Prinzip für das Beispielszenario

- **PC Nr2:** Das Programm auf PC Nr.2 wartet immer auf USB Nachrichten mit der Msg-ID 0x666. Wenn das Programm diese Nachricht empfangen hat, dann inkrementiert es das Data-Feld dieser Nachricht und sendet dieses inkrementierte Data-Feld mit der Msg-ID ox66F über USB an den Mikrocontroller zurück. Wenn Das Programm eine ungültige Nachricht bekommt, sendet es eine Fehlernachricht an den Mikrocontroller. Der Mikrocontroller leitet diese Fehler Meldung über den CAN-Bus an PC Nr.1 weiter, um

ein erneutes Senden der Nachricht zu forcieren. Dieser Vorgang wiederholt sich ebenso in einer Schleife.

6.4 Nachrichten Paket Format

In diesem Beispielszenario werden das USB-Paket und das CAN-Paket verwendet.

CAN-Paket: das CAN-Paket ist in Abschnitt 5.1.1 erklärt.

USB-Paket: Wie vorher in Abschnitt 4.2.1 erwähnt, hat USB verschiedene Paket Formate, eines davon ist das Data-Paket. Diese Daten Paket wird Für die Kommunikation zwischen dem Mikrocontroller und PC Nr.2 verwendet. Da dieses USB Daten Paket für unser Anwendungsprogramm auf dem Mikrocontroller mit dem CAN Data-Paket teilweise übereinstimmen muss, werden zwei Typen vom Data-Paket-Format abgeleitet. Wie in Abbildung 6-5 zu sehen ist, wird im Data-Feld des Data-Pakets zwei unterschiedliche Strukturen definiert. Dadurch existieren zwei Typen des Data-Pakets nämlich das Standard- und das Extended-Data-Paket. Die beiden Data-Paket Typen unterscheiden sich nur in der Länge des Message-ID Feldes. Das Message-ID Feld hat im Standard Data-Paket eine Länge von 2 Byte und im Extended Data-Paket eine Länge von 4 Byte.

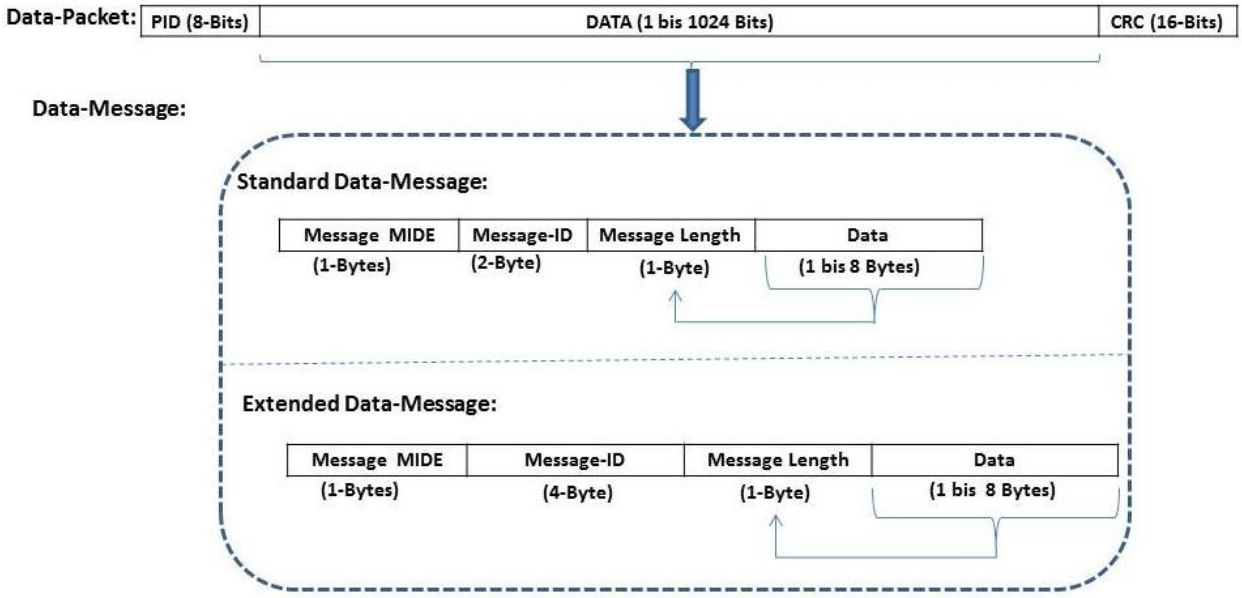

Abbildung 6-5: USB Data-Paket Format

In folgenden werden die einzelnen Felder des Data Frame von USB erklärt:

- **PID:** Dieses Feld entspricht dem PID Feld im USB Data Paket, wie es in Abschnitt 4.1.2 erwähnt ist.
- **Message MIDE:** Dieses Feld entspricht dem IDE Feld im CAN Data Paket. Wenn es 1 ist, ist die USB Data Message vom Extended Typ, sonst folgt der Standard Typ

- **Message-Length:** dieses Feld besteht aus 1 Byte und wird verwendet um die Länge des Data Feld zu speichern.
- **Nachrichten-ID:** das Feld hat je nach Data Paket Typ 2 oder 4 Byte Länge und steht für den Nachrichten Identifier.
- **Data:** dieses Feld hat bis zu 8 Bytes Länge und enthält die eigentlichen Daten.

6.5 Implementierung des USB-CAN Adapters auf dem Mikrocontroller

Das USB-CAN Schnittstellenprogramm für den Mikrocontroller hat die folgenden Aufgaben:

- Konfiguration und Steuerung des CAN Kontroller auf dem Mikrocontroller
- Konfiguration und Steuerung des UDP (USB Device Port) Kontroller auf dem Mikrocontroller
- Empfang der CAN Nachrichten
- Konvertierung von CAN Paketen zu USB Paketen und deren Weiterleitung über UDP an den PC Nr.2
- Empfang der USB Nachrichten
- Konvertierung von USB Paketen zu CAN Paketen und deren Weiterleitung über den CAN Controller an PC Nr.1

6.5.1 Flussdiagramm des Main-Programms des USB-CAN Adapters

Die oben definierte Aufgabe erfordert die Verwendung von mehreren Peripheriegeräten, dafür ist es notwendig den entsprechenden Code zur Inbetriebnahme der Peripheriegeräte zu implementieren.

Abbildung 6-6 zeigt das Flussdiagramm des Main-Programms des USB-CAN Adapters auf dem Mikrocontroller. Dieses Flussdiagramm besteht aus vielen Komponenten mit 3 verschiedenen Farben. Die Komponenten in orangener Farbe sind in Abschnitt 4.7 erwähnt und werden hier nicht erneut erklärt. Die Komponenten in blauer und grauer Farbe werden im folgenden Abschnitt genauer erläutert.

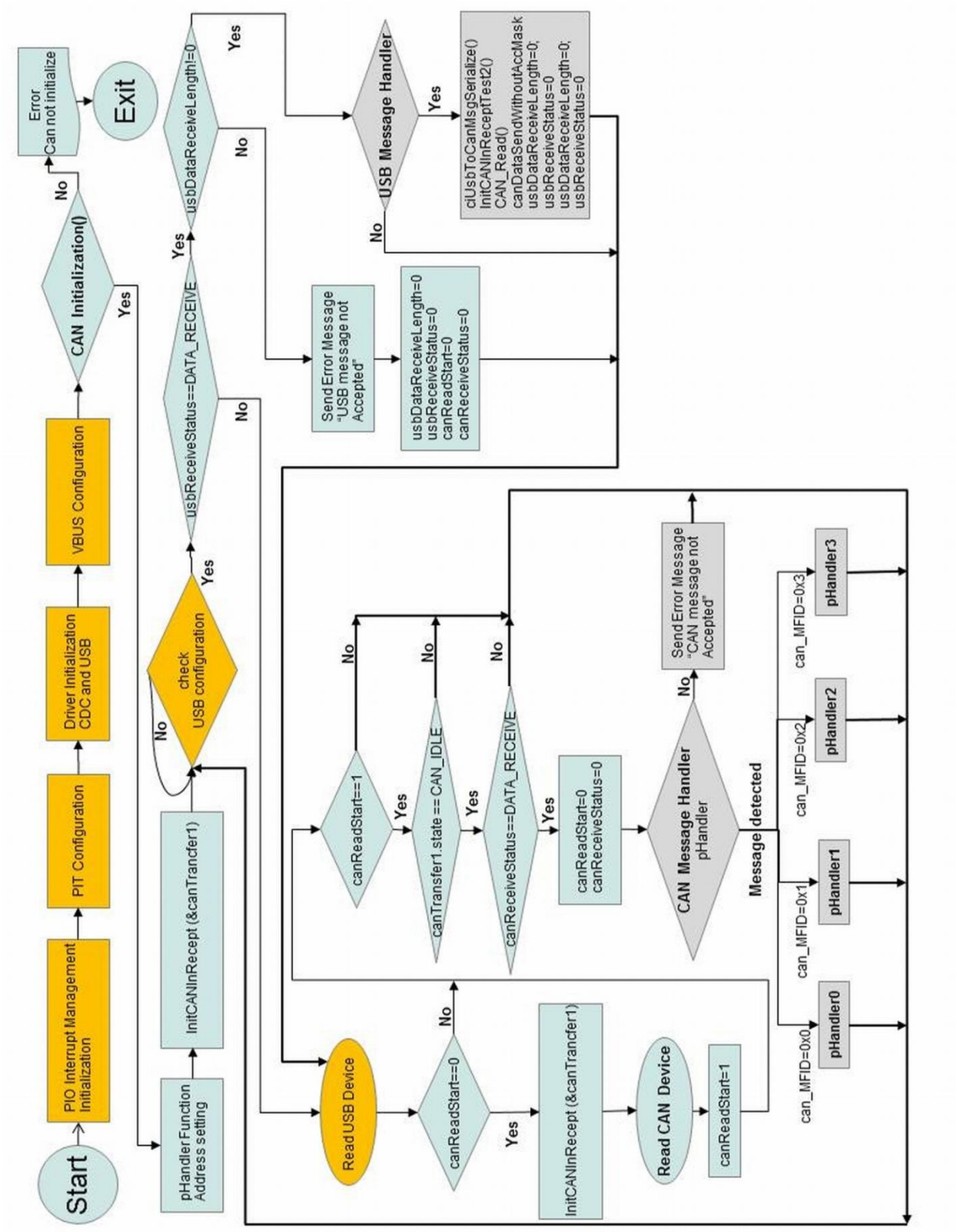

Abbildung 6-6: Flussdiagramm des Main-Programms des USB-CAN Adapters

- CAN-Transfer: Um eine Mailbox in verschiedenen Sende oder Empfangs Modi zu organisieren, werden die folgenden acht Register verwendet, (siehe Kapitel 5.2.2.1):[ATM11, S.677]
- Mailbox Mode Register (CAN_MMR)
- Mailbox Acceptance Mask Register (CAN_MAM)
- Mailbox ID Register (CAN_MID)
- Mailbox Family ID Register (CAN_MFID)
- Mailbox Status Register (CAN_MSR)
- Mailbox Data Low Register(CAN_MDL)
- Mailbox Data High Register(CAN_MDH)
- Mailbox Control Register(CAN_MCR)

Damit man die Mailbox einfacher verwalten kann, wird eine Struktur Namens „CanTranfer“ definiert. Jede Variable in dieser Struktur entspricht einer der obengenannten Register. Wenn eine Nachricht empfangen wird, wird der CAN-Interrupt Handler das obengenannte Mailbox- Register lesen und in der „CanTransfer“ Struktur speichern. Wenn eine Nachricht gesendet wird, wird die Komponente (member) von der „CanTransfer“ Struktur gelesen und in das zugehörige Mailbox-Register gespeichert. Im Folgenden ist die genaue Definition der „CanTranfer“ Type Struktur aufgelistet.

```
typedef struct
{
    volatile unsigned char state;
    volatile unsigned char can_number;
    volatile unsigned char mailbox_number;
    volatile unsigned char test_can;
    volatile unsigned int  mode_reg;
    volatile unsigned int  acceptance_mask_reg;
    volatile unsigned int  identifier;  // CAN Message-ID Register
    volatile unsigned int  data_low_reg;
    volatile unsigned int  data_high_reg;
    volatile unsigned int  control_reg;
    volatile unsigned int  mailbox_in_use;
    volatile unsigned int  can_MFID; // CAN Message Family Register
    volatile int size; // Mailbox Data Length in CAN Message Status Register
} CanTransfer;
CanTransfer Cantransfer1;
```

- „CAN_Init()“: In dieser Funktion werden die folgenden Schritte abgearbeitet:

1- Wie in Abbildung 5-4 gezeigt ist, werden die folgenden verbundenen PIO Pins mit dem CAN Kontroller durch die „PIO_Configure()“ Funktion konfiguriert:

- CAN Transmit Serial Data (CAN TXD)
- CAN Receive Serial Data (CAN RXD)
- CAN RS
- CAN RXEN

2- Die Aktivierung der PIO-Clock des CAN in der PMC.

3- Die Aktivierung der Clock für den CAN0-Controller in der PMC

4- Alle Interrupts des CAN0-Controllers werden mithilfe des CAN-Interrupt-Disable-Registers deaktiviert.

5- Der Advanced-Interrupt-Controller wird für die CAN Interrupts durch die Funktion „AIC_ConfigureIT()“ konfiguriert.

6- Die CAN Interrupts werden durch die Funktion „AIC_EnableIT()“ direkt im Interrupt-Controller aktiviert.

7- Die Bit-Timings für den CAN-Controller werden durch die Funktion „CAN_BaudRateClculate()“ mit der angeforderten Baudrate berechnet. Für weitere Information zur Bit-Timing Berechnung siehe Kapitel 5.2.2.2.

8- In der Funktion „CAN_ResetALLMailbox()“werden alle Mailboxen zurückgesetzt.

9- Alle Fehler die der CAN-Controller detektiert und einen Interrupt provozieren, werden mithilfe des CAN Interrupt Enable Register aktiviert.

10- Mithilfe der Funktion „CAN_Synchronisation()“ wird in diesem Schritt auf die CAN Synchronisation gewartet. Wenn die Synchronisationsphase fertig ist, gibt diese Funktion den Wert 1 zurück, ansonsten 0. Es gibt zwei Arten von Synchronisation, diese werden in "harte Synchronisation", am Anfang eines Rahmens, und „Neusynchronisierung“, innerhalb eines Rahmens, unterschieden. Nach einer harten Synchronisation wird die Bit-Time mit dem Ende des „SYNC_SEG“ Segment, unabhängig von dem Phasenfehler, neu eingestellt. Neusynchronisation führt zu einer Verringerung oder Erhöhung der Bit-Time, so dass die Position des Sample-Punktes in Bezug auf die detektierte Flanke verschoben wird.

- CAN Interrupt Handler: Es gibt zwei verschiedene Typen von Interrupts. Die eine Art von Interrupts sind Nachrichten Interrupts und die anderen sind System Interrupts, die entsprechende Fehler oder System Interrupts behandeln. Die Nachricht Interrupts und die System Interrupts sind durch die Funktion „CAN_Handler()“ und „CAN_ErrorHandling()“ in „can.h“ von Atmel implementiert.
  - CAN_Handler(): Diese Funktion behandelt folgende Mailbox Ereignisse:
    - Im Empfangsmodus wird eine erhaltene Nachricht durch die CanTransfer-Struktur für das Anwendungsprogramm verfügbar gemacht.
    - Im Sendemodus wird eine Nachricht erfolgreich übertragen.
  - CAN_ErrorHandling(): Diese Funktion behandelt folgende Mailbox Ereignisse:
    - Bus-off-Interrupt: Das CAN-Bus-Modul tritt in den „bus-off” Zustand, dh. Das Modul wird vom CAN-Bus getrennt.

- Error-passive-Interrupt: Das CAN-Modul tritt in den Error-Passiv-Modus und kann dadurch den Bus nicht mehr mit weiteren Fehlernachriten blockieren.
- Error-Active-Modus: Das CAN-Modul ist weder im Error Passive Modus noch im „Bus-Off“ Modus.
- Warn-Limit-Interrupt: Das CAN-Modul befindet sich aktiv im „Error-Modus“ und zumindest einer ihrer Fehlerzähler übersteigt den Wert 96.
- Wake-up-Interrupt: Dieser Interrupt wird nach einem Wake-Up und einer Bussynchronisation generiert.
- Sleep-Interrupt: Dieser Interrupt wird erzeugt, nachdem ein Low-Power-Modus aktiviert wurde und wenn alle ausstehenden Nachrichten der Übertragung gesendet wurden.
- Internal-Timer-Counter Overflow Interrupt: Dieser Interrupt wird generiert, wenn der interne Timer überläuft.
- Timestamp-Interrupt: Dieser Interrupt wird nach dem Empfang oder der Übertragung eines „Start-of-Frame“ oder eines „End-of-Frame“ erzeugt und der Wert des internen Zählers in das „CAN_TIMESTP“ Register kopiert.

- CAN_RestTransfer(): Die CanTransfer-Struktur wird durch diese Funktion zurückgesetzt. D.h. alle Komponenten (members) der CanTransfer-Struktur werden auf 0 (null) gesetzt.
- CAN_InitMailboxRegisters(): Diese Funktion konfiguriert die Register der angeforderten Mailbox, als Übergabeparameter dient die CanTransfer-Struktur.
- InitCanInRecept(): diese Funktion kann angeforderte Mailboxen im Empfangsmodus initialisieren und die erwünschte CAN-ID maskieren. Das Nachrichtenannahmeverfahren (Maskierung) ist schon genauer in Kapitel 5.2.2.1.1 erklärt.
- CAN_Read(): Diese Funktion gibt als Rückgabe den CAN Status und als Übergabeparameter hat sie die „CanTransfer“ Struktur. Diese Funktion ändert den CAN-Zustand in „CAN_SENDING“, wenn der vorherige CAN_Zustand „CAN_IDLE“ war, sonst gibt sie als Rückgabe „CAN_STATUS-LOCKED“ zurück. Dann wird der Interrupt von der zugehörigen Mailbox im CAN Interrupt Enable-Register (CAN_IER) aktiviert. Wenn die Mailbox eine Nachricht empfängt, wird der Interrupt erzeugt. Dadurch wird der CAN Interrupt-Handler aufgerufen und die empfangene Nachricht in der CanTransfer-Struktur gespeichert.
- CAN_Write(): Diese Funktion gibt als Rückgabe den CAN Status, und als Übergabeparameter besitzt sie die CanTransfer-Struktur. Diese Funktion ändert den CAN-Zustand in „CAN_RECEIVING“, wenn der vorherige CAN_Zustand „CAN_IDLE“ war, ansonsten gibt es „CAN_STATUS-LOCKED“ als Rückgabe. Dann wird der Interrupt von der zugehörigen Mailbox im CAN Interrupt Enable Register (CAN_IER) aktiviert. Dazu wird das zugehörige Feld der Mailbox im CAN Transfer Command Register aktiviert. Das heißt, dass die Nachricht in der Mailbox bereit ist und dadurch sendet der CAN-Controller so schnell wie möglich die Nachricht.

- ciCanToUsbMsgSerialize(): Mithilfe dieser Funktion wird die empfangene CAN Nachricht (siehe Kapitel 5.1.4) zu einer USB-Nachricht (USB Frame siehe in Kapitel 6.4) konvertiert.
- ciUsbToCanMsgSerialize(): Mithilfe diese Funktion wird die empfangene USB Nachricht zu einer CAN Nachricht konvertiert.
- canDataSendWithoutAccMask(): Diese Funktion sendet die vorbereitete Nachricht in der CanTransfer-Struktur ohne die Acceptance-Mask. In dieser Funktion wird zuerst das Acceptance-Mask-Register auf null gesetzt. Anschließend wird mithilfe des „CAN_InitMailboxRegisters()“ und „CAN_Write()“ die CAN Nachricht gesendet.
- pHandler(): Wenn die Anwendung einen Handler für jede CAN-Nachrichten-ID verbindet, kann es ein Array von Zeigern auf Funktionen definieren (siehe Kapitel 5.2.2.1.1). Wenn eine CAN Nachricht empfangen wird, kann der entsprechende Handler mit dem Wert des „CAN_MFIDx“ Registers aufgerufen werden. In Jeder „pHandler()“ Funktion wird die empfangene CAN Nachricht durch „ciCanToUsbMsgSerialize()“ Funktion zu einer USB Nachricht konvertiert und durch den USB Device Port weiter gesendet.
- usbMessageHandler(): In dieser Funktion wird die empfangene USB Nachricht durch die „ciUsbToCanMsgSerialize()“ Funktion zu einer CAN Nachricht konvertiert und durch den CAN Device Port weitergeleitet.

6.5.2 Die Evaluierung des USB-CAN Adapters

6.5.2.1 Die Implementierung des Testprogramms auf den PCs

- Test Programm auf PC Nr.1: Wie bereits in Kapitel 6.2 erwähnt, wird ein PCAN-USB Gerät verwendet, um den PC mit dem CAN-Bus verbinden zu können. Damit das implementierte Programm auf das PCAN-USB Gerät zugreifen kann, wird der PCAN-USB Treiber installiert. Weitere Informationen über den PCAN-USB Treiber sind in [Pes11] zu finden. In diesem Anwendungsprogramm wird die entsprechende Bibliothek von der PEAK System-Technik GmbH verwendet, um auf das PCAN-USB Geräte zugreifen zu können. Die folgenden Methoden der „libpcan.h“ Bibliothek wurden verwendet.

1. initializeCan(): Die übergebenen Parameter für diese Methode sind die CAN Baudrate und der CAN Frame Typ. PCAN-USB Geräte werden mit Hilfe der „initializeCan()“ Methode initialisiert.
2. CAN_Read(): Durch diese Methode wird die empfangene CAN Nachricht gelesen.
3. CAN_Write(): Mit Hilfe dieser Methode wird eine vorbereitete CAN Nachricht über den CAN-Bus gesendet.

In diesem Anwendungsprogramm wird zuerst das PCAN-USB Gerät vermittels der „initializeCan()“ Methode mit der angeforderten CAN Baudrate und dem entsprechenden CAN Frame Typ initialisiert. Danach wird eine CAN Message mit der CAN-ID 0x666 und dem Data Feld 0x0 gesendet.

Wenn eine CAN Nachricht empfangen wird, wird zuerst überprüft, welche CAN-ID diese Nachricht hat. Wenn die empfangene Nachricht die CAN-ID 0x66F besitzt, wird das Data Feld von der Nachricht gelesen und inkrementiert. Das inkrementierte Data Feld wird als neue CAN Nachricht mit der CAN-ID 0x666 gesendet. Wenn das Data Feld von der empfangenen Nachricht den Wert 0xFF erreicht hat, wird das Programm beendet.

- Test Programm auf PC Nr.2: Der PC Nr.2 wird über den USB Port mit dem Mikrocontroller verbunden. In diesem Anwendungsprogramm wird die Klasse „ciSerial“ verwendet. Diese Klasse öffnet eine serielle Schnittstelle, um auf das USB Device zuzugreifen. Die Methode „serialUsbReadStr()“ liest die empfangene Nachricht und „serialUsbWriteStr()“ sendet die USB Nachricht über die USB Schnittstelle.

In dieser Anwendung wird zuerst eine serielle Schnittstelle mit dem entsprechenden Namen des Device Files, der Baudrate und dem Processing-Type geöffnet.
Das Programm wartet immer auf den Empfang einer USB Nachricht. Sobald das Programm eine USB Nachricht empfangen hat, wird das Programm das Message-ID Feld der Nachricht überprüfen. Falls die Message-ID den Wert 0x666 hat, wird das Data-Feld der Nachricht gelesen und inkrementiert. Das Inkrementierte Data Feld wird erneut als USB Nachricht mit der Message-ID 0x666 gesendet.

- PC Nr.3: der PC wird mit Hilfe eines PCAN-USB Geräts mit dem CAN-Bus verbunden. Das Programm PCAN-View (siehe Kapitel 6.2) wird auf dem PC unter Windows installiert. Es ermöglicht, dass alle laufende CAN Nachrichten über den CAN-Bus veranschaulicht werden können.

6.5.2.2 Durchführung der Evaluierung

Für die Durchführung der Evaluierung werden die folgenden Schritte vollzogen:

1. Die PCs und der Mikrocontroller werden wie Abbildung 6-1 zu sehen verbunden.
2. Das USB-CAN Adapterprogramm wird auf den Mikrocontroller kopiert und gestartet.
3. Das Anwendungsprogramm für PC Nr. 2 wird gestartet.
4. Das Programm PCAN-View wird auf PC Nr. 3 gestartet.
5. Das Anwendungsprogramm für PC Nr. 1 wird gestartet.

Jedes Programm auf den PCs Nr.1 und Nr.2 zeigt die empfangenen und gesendeten Nachrichten an. Zur Kontrolle ermöglicht das Programm PCAN- View die laufenden Nachrichten auf dem CAN-Bus zu detektieren. Die Anzahl der CAN Nachrichten mit der CAN-ID 0x666 und 0x66f zeigt die Korrektheit des USB-CAN Adapters an.

Während der Ausführung der Programme auf den 3 PCs, kann man mit Hilfe von PCAN-View einige CAN Nachrichten mit unterschiedlichen CAN-IDs über den CAN-Bus senden, um die Korrektheit des Nachrichtenannahmeverfahrens (Maskierung) des USB-CAN Adapters zu prüfen.

7 CAN-Ethernet Adapter

Wie im vorherigen Kapitel erwähnt wurde, wurde für die Implementierung des CAN-USB Adapters die Low-level Programmierung angewendet. Bei dieser Art der Programmierung gibt es kein Betriebssystem auf dem Mikrokontroller. Die programmierte Anwendung wird mit Hilfe einer Cross-Compiler Toolchain kompiliert, die dadurch erstellte bin-Datei der Anwendung wird direkt auf das SDRAM des Mikrokontrollers kopiert und ausgeführt.

Im Gegensatz zur CAN-USB Adapter Implementierung wird für die Implementierung eines CAN-Ethernet Adapters die Socket Programmierung verwendet. Für die Verwendung dieser Methode muss ein Embedded-Betriebssystem (Embedded-Linux) auf dem Mikrocontroller installiert werden. Dann wird das Anwendungsprogramm in C++ geschrieben. Das kann im Application-Layer geschehen, also eine Abstraktionsebene höher als beim CAN-USB Adapter.

7.1 Toolchain

Die folgende Toolchain wurde verwendet, um den Linux Kernel und die Anwendungs-programme für den Mikrocontroller zu kompilieren.

7.1.1 BitBake

BitBake ist ein von Chris Larson entwickelter Task Executor, der in Python geschrieben ist[Reh07]. BitBake ist ein Werkzeug für die Durchführung von verschiedenen Aufgaben. Es wird am häufigsten verwendet um Pakete zu erzeugen, und wird als Basis des OpenEmbedded Projekts verwendet[Ber12].

In den so genannten „Bitbake-Recipes“, analog zu den Makefiles, stehen die Anweisungen für BitBake, wie ein bestimmtes Paket zu erstellen ist. Es beinhaltet alle Paket-Abhängigkeiten, die Quellen sowie die verschiedenen Konfigurationen und Anweisungen zum Erstellen, Bauen, Installieren und Entfernen. [BerBi].

7.1.2 OpenEmbedded

OpenEmbedded ermöglicht unter Verwendung eines Bitbake-Recipes, das Erstellen einer kompletten Linux Distribution für Embedded-Systeme (Linux-Image) oder eines Software Pakets für ein bestimmtes Geräte [OPE01][OPE02].

7.1.3 Ångström Linux für Embedded Systeme

Ångström wurde von einer kleinen Gruppe von Menschen begonnen, die an den OpenEmbedded, OpenZaurus und OpenSimpad Projekten gearbeitet haben, um eine stabile und

benutzerfreundliche Distribution für Embedded-Geräte, wie Handhelds, Network-Attached Storage-Geräte und mehr, zu vereinheitlichen [Ang].

Im Angstrom Paket gibt es einen Angstrom Cross-Compiler. Mit Hilfe dieses Compilers wird das Anwendungsprogramm und der Linux Kernel kompiliert.

7.2 Konfiguration, Erstellen und Starten des Linux Kernels

Zuerst wird die Source des gewünschten Linux Kernels (z.B. Version 2.6.34) auf den PC heruntergeladen. Anschließend muss der Linux Kernel für die entsprechende Hardware konfiguriert werden. In der Konfiguration muss zusätzlich die Option für „SocketCAN“ (siehe Kapitel 7.3) aktiviert werden. Dann wird der so konfigurierte Linux Kernel kompiliert und ein so genanntes „zImage“ erstellt. Weil U-Boot keine normalen Linux-Kernel-Images wie „zImage“ unterstützt, muss mit dem „mkimage“ Werkzeug eine uImage-Datei aus der zImage-Datei erstellt werden [ATMLI].

U-Boot: U-Boot ist die zweite Stufe des „Bootloaders“. Es ist für die Konfiguration der wichtigsten Schnittstellen und den Start des Linux-Systems zuständig [ATMU].

U-Boot bietet Unterstützung für das Laden von Binärdateien (wie uImage eines Linux Kernels) aus einem entfernten Rechner im Netzwerk über das TFTP-Protokoll. Dafür muss ein TFTP Server auf dem PC installiert sein und auf der U-Boot Seite müssen die Netzwerk Parameter korrekt definiert sein.

Wenn der TFTP Server auf dem PC konfiguriert ist, kann das U-Boot uImage des Linux Kernels durch das Netzwerk in den SDRAM des Mikrocontrollers kopiert werden und davon gebootet werden.

7.3 SocketCAN

SocketCAN ist eine Sammlung von Open-Source-CAN-Treibern. Früher war es auch unter dem Namen Low Level CAN Framework (LLCF) bekannt.

Traditionell basieren CAN-Treiber für Linux auf dem Modell der „Character-Devices“. Typischerweise erlauben sie dem CAN-Controller nur das Senden und Empfangen. Konventionelle Implementierungen dieser Klasse von Gerätetreibern lassen nur einen einzigen Prozess auf das Gerät zugreifen, d.h. dass alle anderen Prozesse in der Zwischenzeit blockiert sind.

Das SocketCAN Konzept auf der anderen Seite nutzt das Modell der Netzwerkgeräte. Diese ermöglichen es, dass mehrere Anwendungen auf ein CAN Gerät gleichzeitig zugreifen können. Des Weiteren ist eine einzige Anwendung in der Lage, auf mehrere CAN-Netzwerke parallel zuzugreifen. Abbildung 7-1 zeigt die typischen CAN Kommunikationsschichten [SOC].

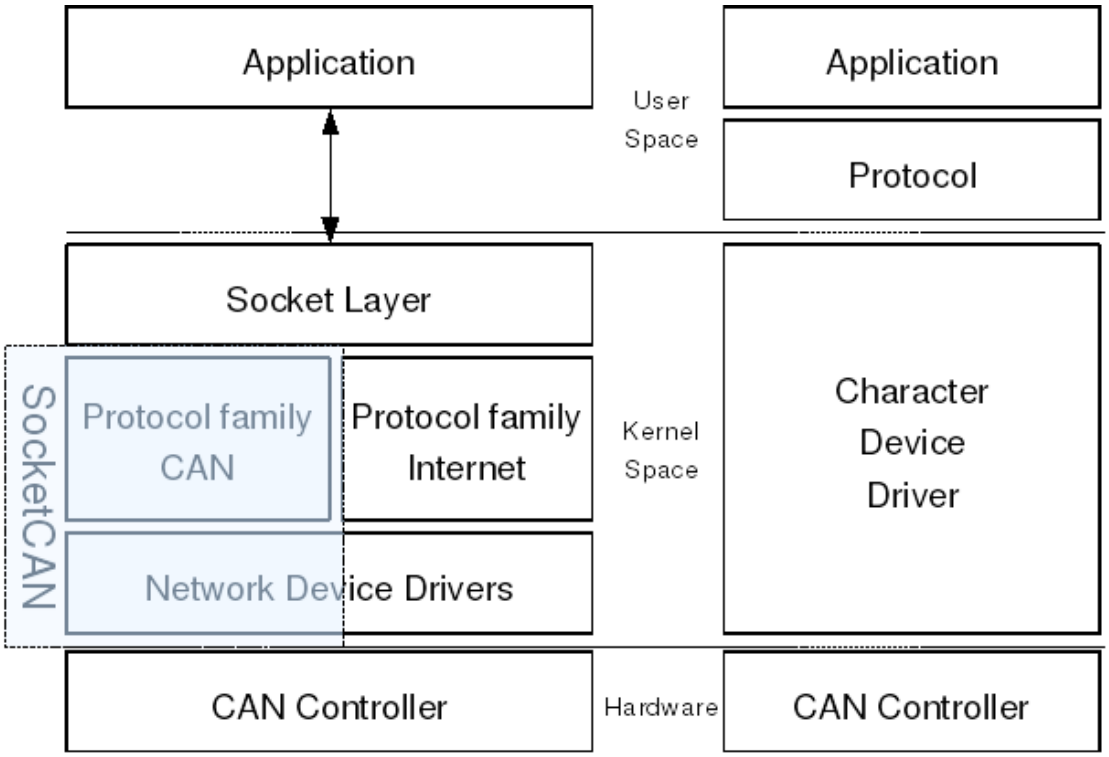

Abbildung 7-1: CAN Kommunikationsschichten für SocktCAN (links) und konventionelle Treiber (recht) [SOC]

7.4 Socket Programmierungsgrundlagen

Ein Socket ist eine Schnittstelle (API) zwischen einem Anwendungsprogramm und einem Transport Protokoll. Durch diese Schnittstelle wird es einem Anwendungsprogramm ermöglicht sich auf einem Computer über diesen Socket mit einem Rechnernetz zu verbinden und mit anderen Computern Daten auszutauschen. Diese Verbindung ist bidirektional. [COD] Ein Computerprogramm fordert einen Socket vom Betriebssystem an, das Betriebssystem hat daraufhin die Aufgabe, alle benutzten Sockets sowie die zugehörigen Verbindungsinformationen zu verwalten.

Ethernet Socket: man unterscheidet zwei verschiedene Ethernet Socket Typen

- Stream Socket: dieser Socket kommuniziert über einen Zeichen-Datenstrom und verwendet das Transmission Control Protocol (TCP).
- Datagram Socket: dieser Socket kommuniziert über einzelne Nachrichten und verwendet das User Datagram Protocol (UDP).

7.4.1 Grund Funktionen der Socket Programmierung:

In diesem Beispiel für die Socket Programmierung wird das Anwendungsprogramm auf 2 PCs, ein PC als Client und der andere als Server, in einem Netzwerk ausgeführt und über einen Socket findet eine Kommunikation zwischen den beiden Rechnern statt. Folgende Grundfunktionen werden in der Socket Programmierung verwendet.[Rad]

- socket(): die Methode erstellt einen Socket mit Domäne, Type und Protokoll als Übergabeparameter
- bind(): ein zuvor erstellter Socket wird an die angeforderte Port Nummer gebunden.
- listen(): Mit listen versetzt man einen Socket in den Listening-Mode, das heißt er wartet auf eingehende Verbindungen.
- accept(): der Server akzeptiert eine Verbindung von einem Client
- connect(): der Client sendet eine Verbindungsanfrage einen Server (Call)
- send(): sendet eine Nachricht über den Socket
- recv(): empfängt eine Nachricht über den Socket
- close(): diese Funktion schließt den Socket.

7.4.2 Ablauf der Socket Kommunikation

Auf dem Server: wie in Abbildung 7-2 gezeigt ist, werden auf dem Server die folgenden Schritte für die Socket Kommunikation durchlaufen.

1. Es wird ein Server-Socket erstellt. (Socket())
2. Der soeben erstellte Server-Socket wird an eine Adresse (Port Nummer) gebunden. (Bind())
3. Auf eine Verbindungsanfrage warten. (listen())
4. Wenn die Verbindungsanfrage akzeptiert wird, wird ein neuer Socket für diesen Client erstellt. (accept())
5. Die Client-Anfragen werden auf dem neuen Client-Socket mit send() und recv() Methoden gesendet oder empfangen.
6. Nach dem Ende der Bearbeitung wird der Client-Socket wieder geschlossen. (close())

Auf dem Client: wie in Abbildung 7-2 zu sehen ist, werden die folgenden Schritte für die Socket Kommunikation auf dem Client abgearbeitet.

1. Es wird ein Client-Socket erstellt. (socket())
2. Der erstellte Socket wird mit der Server Adresse verbunden. (connect())
3. Der Client kann die Daten über den Socket mit send() und recv() Methoden senden oder empfangen.
4. Der Client-Socket wird geschlossen. (close())

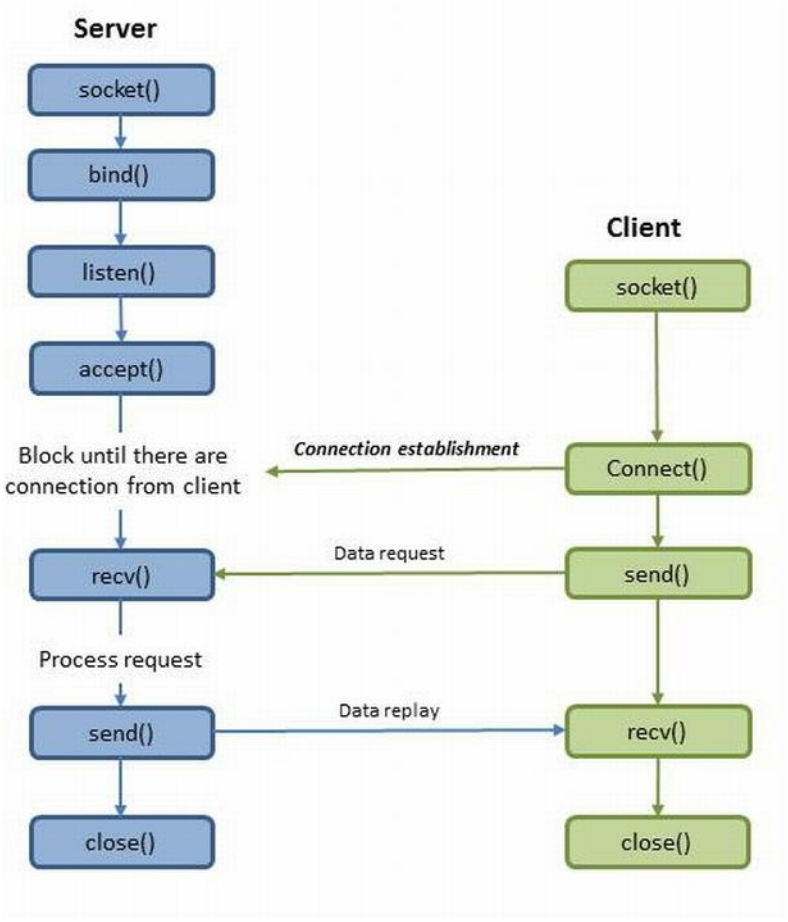

Abbildung 7-2: Client/Server Beziehung der Socket API für TCP [NET]

7.5 CAN-Ethernet Adapter Implementierung

Die CAN-Ethernet-Adapter Anwendung wird durch eine Socket Programmierungstechnik implementiert. Wie in Abbildung 7-3 gezeigt ist, wird zuerst ein Listening Socket (Server Socket) erstellt. Der Socket wird an die angeforderte Port Nummer gebunden und wartet auf eine Verbindungsanfrage. Wenn PC Nr.2 über LAN eine Verbindungsanfrage an den Mikrocontroller gesendet hat, wird die Anfrage, je nach Einstellung des Listening Socket, akzeptiert oder verworfen. Wird die Verbindungsanfrage akzeptiert, wird ein Netzwerk Socket für diesen Client (PC Nr.2) erstellt. Der erstellte Netzwerk Socket wird durch seinen File Deskriptor (fd) an der „Communication-Engine“ im CAN-Ethernet Programm registriert.

Für den CAN-Port wird im CAN-Ethernet Adapter ebenso ein CAN Socket erstellt. Der CAN-Socket wird analog zum Ethernet-Socket durch seinen File Deskriptor an der „Communication Engine“ angemeldet.

In der „Communication-Engine“ [Raj09] wird durch eine „Wait“ Methode mit der angeforderten Wartezeit auf ein Signal vom Ethernet- oder CAN- File Deskriptor gewartet. Durch diese

„Wait“ Methode verbleibt die Adapter Anwendung für die angeforderte Wartezeit in einem „Sleep Mode“. D.h. das CAN-Ethernet Adapterprogramm verwendet keine CPU-Zeit.

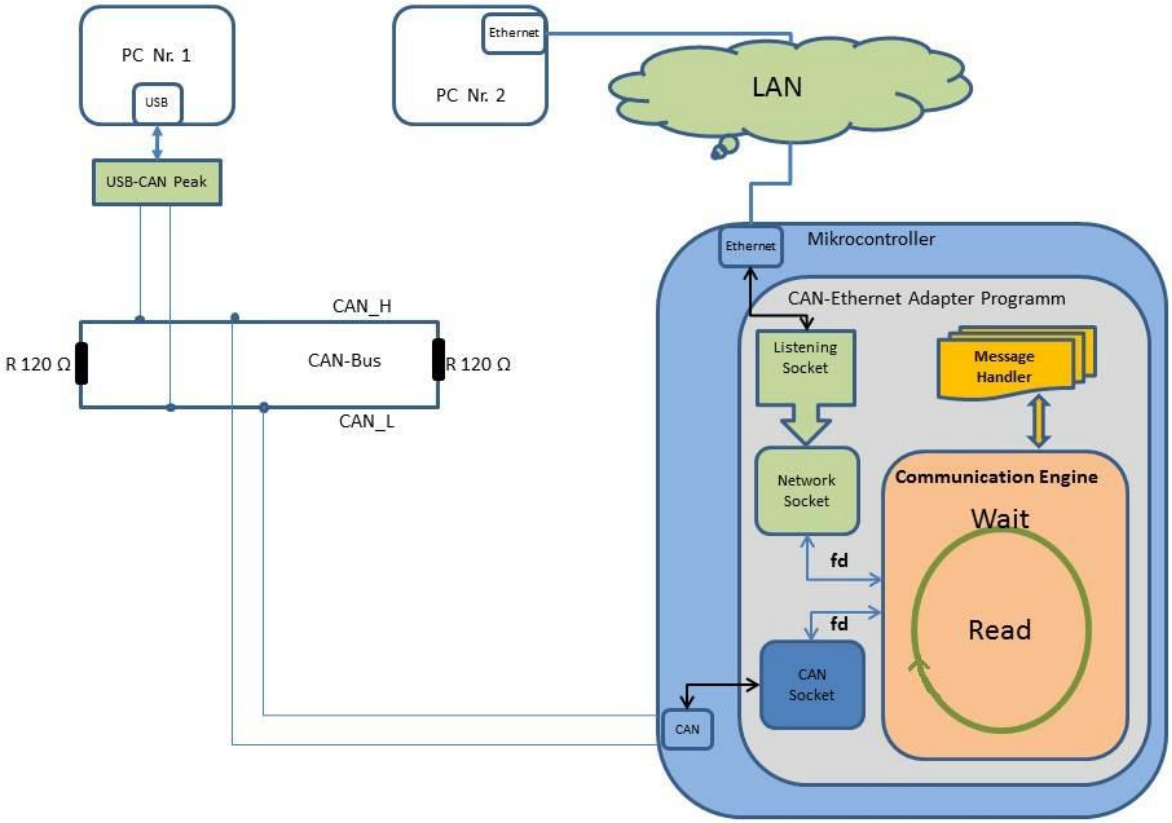

Abbildung 7-3: CAN-Ethernet Adapter Anwendungsprogramm

Wenn eine Datei durch den LAN oder den CAN Port empfangen wird, wird dies der „Communication-Engine“ durch den File Deskriptor signalisiert und damit das gesamte Programm aufgeweckt. Dann liest die „Communication Engine“ die empfangene Nachricht durch ihre Read Methode über den jeweiligen Socket ein.

Wenn eine CAN Nachricht gelesen wird, wird der zugehörige Message Handler (CAN-Message Handler) aufgerufen. Im CAN-Message Handler wird die empfangene CAN Nachricht zu einer Ethernet Nachricht konvertiert und durch die Write Funktion über den Netzwerk Socket an den PC Nr. 2 gesendet.

Wenn eine Ethernet Nachricht gelesen wird, wird hier ebenso der zugehörige Message Handler (Ethernet-Message Handler) aufgerufen. Im Ethernet-Message Handler wird die empfangene Ethernet Nachricht zu einer CAN Nachricht konvertiert und durch die zugehörige Write Funktion über den CAN Socket an den PC Nr. 1 gesendet.

7.6 Die Evaluierung des CAN-Ethernet Adapter Anwendungsprogramms

Für die Evaluierung werden erneut 3 PCs und ein Mikrocontroller verwendet. Wie in Abbildung 7-4 gezeigt ist, sind PC Nr. 1 und PC Nr. 3 über eine PCAN-USB Geräteschnittstelle an das CAN-Netzwerk angeschlossen. Der PC Nr. 2 ist über seine Ethernet Schnittstelle an das LAN angeschlossen. Der Mikrocontroller ist auf der einen Seite über Ethernet an das LAN und auf der anderen Seite über einen CAN Port mit dem CAN verbunden.

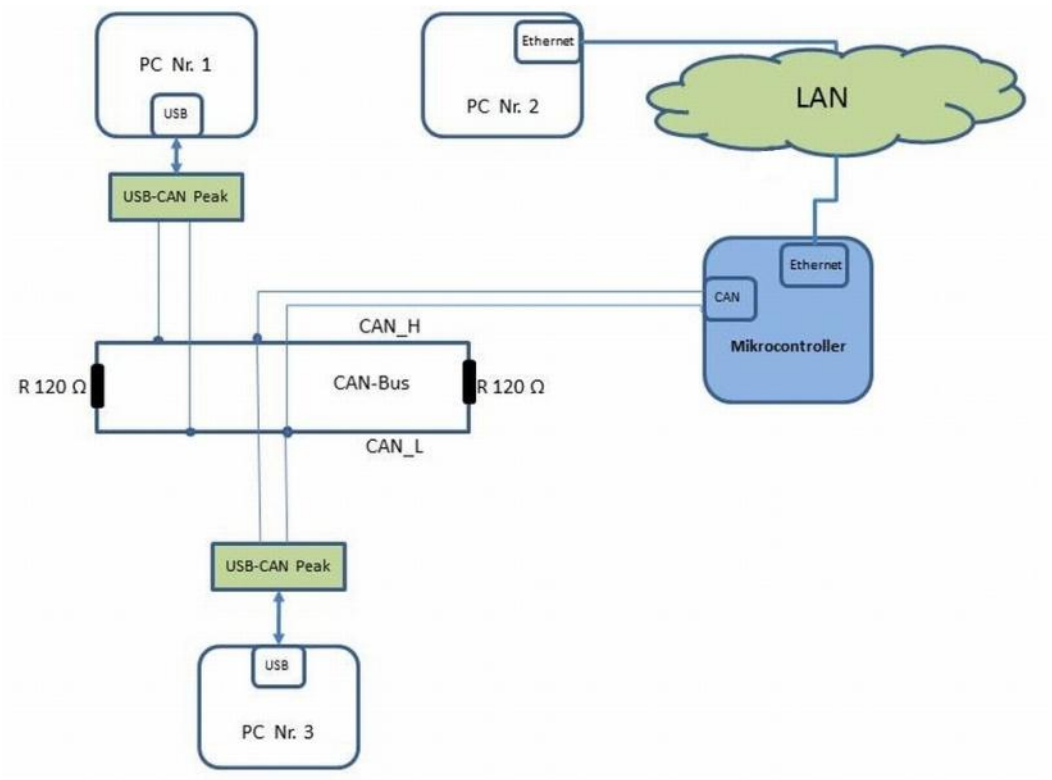

Abbildung 7-4: CAN-Ethernet Adapter Testaufbau

7.6.1 Die Implementierung des Testprogramms auf den PCs

- Testprogramm auf PC Nr. 1: die Implementierung von diesem Test Programm ist analog zu dem Testprogramm auf dem PC1 in Kapitel 6.5.2.1.

- Testprogramm auf PC Nr. 2: das Testprogramm auf PC Nr. 2 wird vermittels der Socket Programmierung implementiert. Der PC Nr. 2 ist über den Ethernet Port an das LAN-Netzwerk angeschlossen. In diesem Anwendungsprogramm wird zuerst ein Client Socket erstellt. Der Client Socket verbindet sich mit dem CAN-Ethernet Adapter (als Server) auf dem Mikrokontroller. Wenn die Verbindung hergestellt ist, wird zuerst eine Counter Variable mit dem Wert „0“ in das Data-Feld der Ethernet-Nachricht gepackt und dann

gesendet. Das Data-Feld der Nachricht hat eine maximale Länge von 8 Byte. Das Anwendungsprogramm wartet in einer Main-Schleife auf neue Nachrichten. In dieser Schleife wird mit einer „Wait“ Methode auf eine Nachricht auf dem Socket gewartet. Durch diese „Wait“ Methode geht das Testprogramm mit der angeforderten Wartezeit in den Sleep Mode. Wenn eine Nachricht empfangen wird, wird das Programm wieder aufgeweckt. Die empfangene Nachricht wird gelesen und das Data-Feld der Nachricht wird inkrementiert. Das inkrementierte Data Feld wird in eine neue Nachricht gepackt und an den Mikrocontroller gesendet. Diese Vorgang läuft bis das Data Feld den Wert 0xFF erreicht. Dann wird der Client Socket geschlossen und das Programm wird beendet.

- **PC Nr. 3**: auf dem PC Nr. 3 läuft erneut ein Windows Betriebssystem. Der Treiber für das PCAN-USB Gerät wird auf dem PC unter Windows installiert und schließlich wird erneut das Programm PCAN-View (siehe Kapitel Abbildung 6-2) installiert und zur Auswertung verwendet.

7.6.2 Der Evaluierungsverlauf

Für die Evaluierung werden analog zu Kapitel 6 die folgenden Schritte vollzogen:

1. Die PCs und der Mikrocontroller werden wie Abbildung 7-4 zu sehen verbunden.
2. Das USB-CAN Adapter Programm wird auf den Mikrocontroller kopiert und gestartet.
3. Das Anwendungsprogramm für PC Nr. 2 wird gestartet.
4. Das Programm PCAN-View wird auf PC Nr. 3 gestartet.
5. Das Anwendungsprogramm auf PC Nr. 1 wird gestartet.

Jedes Programm auf den PCs Nr.1 und Nr.2 zeigt erneut die empfangenen und gesendeten Nachrichten an. Zur Kontrolle wird das Programm PCAN-View verwendet um die laufenden Nachrichten auf dem CAN-Bus zu überprüfen. Die Anzahl der CAN Nachrichten mit der CAN-ID 0x666 und 0x66f zeigt die Korrektheit des Ethernet-CAN Adapters an.

Auch hier kann man während der Ausführung der Programme auf den 3 PCs, mit Hilfe von PCAN-View einige CAN Nachrichten mit unterschiedlichen CAN-IDs über den CAN-Bus senden, um die Korrektheit des Nachrichtenannahmeverfahrens (Maskierung) des USB-CAN Adapters zu prüfen.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Ziel der Diplomarbeit war die Entwickelung eines CAN-USB und eines CAN-Ethernet Adapterprogramms auf einer ARM Mikrocontroller (AT91SAM9263) basierten Platine. Für die Implementierung der beiden Adapterprogramme wurden unterschiedliche Programmier-methoden angewendet.

Für die Entwicklung des CAN-USB Adapterprogramms ist die Methode der hardwarenahen Programmierung (Low Level Programmierung) verwendet worden. Die angewandte Programmierungssprache war C. Um das CAN-USB Adapterprogramm zu entwickeln und auszuführen, müssen die voneinander abhängigen Peripheriegeräte wie USB Device Controller, Parallel Input/Output Controller, Advanced Interrupt Controller, Timer Counter, Power Managemet Controller und CAN Controller, auf dem Mikrocontroller verwendet. Für jedes verwendete Peripheriegerät wurden die entsprechenden Bibliotheken von Atmel optimiert und zusätzliche Bibliotheken entwickelt. Außerdem wurde die benötigte Bibliothek für die Konvertierung und Serialisierung von CAN zu USB-Nachrichten und umgekehrt geschrieben. Anschließend wurde das CAN-USB Adapterprogramm mit Hilfe der entwickelten Bibliotheken geschrieben und getestet.

Für die Implementierung des CAN-Ethernet Adapters wird die Methode der Socket Programmierung verwendet. Für die Verwendung dieser Methode wurde ein Embedded-Betriebssystem (Embedded-Linux) auf dem Mikrocontroller installiert. Dazu sind die entsprechenden Treiber für die verwendeten Geräte (CAN und Ethernet) auf dem Embedded-Betriebssystem installiert worden und das Anwendungsprogramm in C++ geschrieben. Dies ist im Application-Layer geschehen, also eine Abstraktionsebene höher als beim CAN-USB Adapter. Außerdem wurden die benötigten Bibliotheken für die Konvertierung und Serialisierung der CAN zu Ethernet-Nachrichten und umgekehrt geschrieben.

Um diese Adapter mit dem PC verbinden und testen zu können, sind zwei Test Szenarien für die jeweiligen Adapter geschrieben worden. Um diese Test Szenarien zu verwirklichen, wurden die benötigten Bibliotheken und entsprechenden Computertestprogramme für den PC entwickelt.

8.2 Ausblick

Für die weitere Entwicklung des CAN-USB und CAN-Ethernet Adapter und einfache Administrierung der Einstellungen, z.B. die entsprechenden Übergabeparameter für die Initialisierung des Adapters, wäre eine GUI sehr hilfreich. Dafür kann auf dem Mikrokontroller ein Web Server installiert und des Weiteren eine zugehörige Webapplikation die, die entsprechende GUI bereitstellt, implementiert werden.

Literaturverzeichnis

[Hach02] Mark Hachman, *ARM Cores Climb Into 3G Territory*, 14.10.2002, URL http://www.extremetech.com/extreme/52180-arm-cores-climb-into-3g-territory , Zugriff am 05.11.2011

[Turl02] Jim Turley, *The Two Percent Solution*, 2002, URL http://eetimes.com/discussion/other/4024488/The-Two-Percent-Solution, Zugriff am 06.11.2011

[ARM10] ARM, *ARM Architecture Referece Manual*, URL http://infocenter.arm.com/help/index.jsp, Zugriff am 08.11.2011

[Fur00] Steve Furber. ARM System-on –Chip Architecture. Addison-Wesley, 2nd edition 2000.

[ATM09] ATMEL, *AT91SAM9263EK Evaluation Board Rev.B User Guide*, Edition 6341D–ATARM–30-Sep-09. URL http://www.atmel.com/Images/doc6341.pdf, Zugriff am 30.11.2011

[ATM11] ATMEL, *AT91SAM9263 Datasheet* , Edition 6249I-ATARM-3-Oct-11, URL http://www.atmel.com/Images/doc6249.pdf, Zugriff am 30.11.2011

[GNU07] ATMEL, *GNU-Based Software Development on AT91SAM Microcontrollers*, Edition 6310A–ATARM–26-Mar-07, URL http://www.atmel.com/Images/doc6310.pdf, Zugriff am 01.12.2011

[SAM08] ATMEL, *AT91SAM-ICE User Guid*, Edition 6206B-ATARM-04-Mar-08, URL http://www.atmel.com/Images/doc6206.pdf,

[SAM06] ATMEL, *SAM Boot Assistant (SAM-BA) User Guide*, Edition 6132C-ATARM-09-Oct-06, http://www.google.de/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&sqi=2&ved=0CCsQFjAA&url=http%3A%2F%2Fwww.symbrion.eu%2Ftiki-download_file.php%3FfileId%3D19&ei=hltoT-T3HdDJswbAsriiCA&usg=AFQjCNFlVuATex03qZg8GKd0t52N0PVQFQ&sig2=t6OvWQyF9vcFnDzgrZdgqQ, Zugriff am 05.12.2011

[Bei04] Thomas Beierlein und Olaf Hagenbruch, *Taschenbuch Mikroprozessortechnik*, Edition 2004 3 Auflage

[Get07] ATMEL, *Getting Started with AT91SAM Microcontrollers*, edition 6297A–ATARM–27-Mar-07 http://www.atmel.com/Images/doc6297.pdf, Zugriff am 07.12.2011

[GNU11] GNU ARM-Toolchain, URL http://wiki.ubuntuusers.de/GNU_ARM-Toolchain, Zugriff am 10. 12. 2011

[Men] Mentor Graphics, *Sourcery CodeBench Lite Edition*, URL http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/, Zugriff am 10.12.2012

[Ecl] Eclipse, *Eclipse CDT (C/C++ Development Tooling)*, URL http://www.eclipse.org/cdt/, Zugriff am 10.12.2011

[Sch01] Prof. Dr. Volker Schubert and Dipl.Ing.Volker Rödig, University of Paderborn, Germany, *Grundlagen der USB-Schnittstelle*, Last Updated by Dr. Allwissend on 15.06.2001, URL http://groups.uni-paderborn.de/cc/arbeitsgebiete/messtech/elektro_grundlagen/usb/index.html, Zugriff am 04.01.2012

[Sch06] Sandro Schnegg, *USB Universal Serial Bus*, 23.Januar 2006, URL https://prof.hti.bfh.ch/uploads/media/USB.pdf, Zugriff am 04.01.2012

[Sch07] Stefan Schalomon, Prof Dr.Ing W.Rehm, TU Chemnitz, *Universal Serial Bus*

URL http://www.tu-chemnitz.de/informatik/RA/news/stack/kompendium/vortraege_97/usb/index.html , Zugriff am 04.01.2012

[ATM06] ATMEL, *AT91 USB Framework*, Edition 6263A–ATARM–10-Oct-06, URL http://www.atmel.com/Images/doc6263.pdf, Zugriff am 15.01.2012

[AT09] ATMEL, *AT91 USB CDC Driver Implementation*, Edition 6269B–ATARM–01-Jul-09, URL http://www.atmel.com/Images/doc6269.pdf, Zugriff am 15.01.2012

[Dav07] Robert I. Davis, Alan Burns, Reinder J. Bril, Johan J. Lukkien, *Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised*, 30 January 2007

[Har11] Oliver Hartkopp, *Programmierschnittstellen für eingebettete Netzwerke in Mehrbenutzerbetriebssystemen am Beispiel des Controller Area Network*, Dissertation,14.Januar.2011

[Nat98] NS Manju Nath, *CAN protocol eases automotive-electronics networking*, Edition 17.Agust.1998, URL http://www.edn.com/archives/1998/081798/17df2.htm, Zugriff am 16.02.2012

[Pah00] Stephan Pahlke und Sven Herzfeld, *CAN-Bus Fehlererzeugung und Analyse*, Diplomarbeit, 2000

[Mes] ME-Messsystem, *CAN Bus Grundlagen*, URL http://www.me-systeme.de/canbus.html, Zugriff am 16.02.2012

[Bos91] Robert Bosch GmbH, *CAN Specification Version 2.0*, September. 1991, URL http://esd.cs.ucr.edu/webres/can20.pdf, Zugriff am 16.02.2012

[PEA11] PEAK System Technik GmbH, *PCAN-USB User Manual*, Version 2.1.0, 5-10-2011, URL http://www.peak-system.com/produktcd/Pdf/English/PCAN-USB_UserMan_eng.pdf, Zugriff am 17.02.2012

[Pes11] PEAK System Technik GmbH, *PCAN Driver for Linux*, Version 7.1, 21-03-2011, URL http://www.peak-system.com/fileadmin/media/linux/files/PCAN%20Driver%20for%20Linux_eng_7.1.pdf, Zugriff am 17.02.2012

[Reh07] Stephan Rehfeld, *BitBake & OpenEmbedded unter Debian Sarge 3.1r0*, Version 0.0.3 , 10-01-2007, URL http://www.trusted-code.de/paper/OpenEmbedded.pdf, Zugriff am 01.03.2012

[Ber12] Berlios Developer, *BitBake build tool*, URL http://developer.berlios.de/projects/bitbake, Zugriff am 01.03.2012

[BerBi] Berlios Developer, *BitBake User Manual*, URL http://bitbake.berlios.de/manual/, Zugriff am 01.03.2012

[OPE01] OpenEmbedded, URLhttp://www.openembedded.org/wiki/Main_Page, Zugriff am 02.03.2012

[OPE02] OpenEmbedded, *OpenEmbedded User Manual*, URL http://docs.openembedded.org/usermanual/usermanual.html#chapter_introduction, Zugriff am 02.03.2012

[Ang] Ångström, *The Ångström Distribution*, URL http://www.angstrom-distribution.org/, Zugriff am 03.03.2012

[ATMLI] ATMEL, *AT91 Linux kernel sources*, URL http://www.at91.com/linux4sam/bin/view/Linux4SAM/LinuxKernel#Build, Zugriff am 01.03.2012

[ATMU] ATMEL, *U-Boot*, URL http://www.at91.com/linux4sam/bin/view/Linux4SAM/U-Boot, Zugriff am 03.03.2012

[COD] CODEPLANET, *TCP/IP Socket-Programmierung*, URL http://www.codeplanet.eu/tutorials/csharp/4-tcp-ip-socket-programmierung-in-csharp.html, Zugriff am 03.03.2012

[Rad] Mani Radhakrishnan and Jon Solworth , *Socket Programming in C/C++*, 24.09.2004, URL http://net.pku.edu.cn/~course/cs501/2011/code/BSD_Socket.t/sockets.pdf, Zugriff am 04.03.2012

[NET] NETWORK PROGRAMMING, URL http://www.tenouk.com/Module39a.html, Zugriff am 05.03.2012

[SOC] SocketCAN, URL http://en.wikipedia.org/wiki/SocketCAN, Zugriff am 04.03.2012

[Raj09] Hamid Reza Rajaie, *Distributed Architecture for Mobile Robots*, PhD thesis, 2009, ISBN 978-3-8322-8567-8

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Unterschrift:

Stuttgart, den 02.05.2012