Hochschule für Angewandte
Wissenschaften Hamburg
Hamburg University of Applied Sciences

BACHELORARBEIT

Software-Entwicklung für eine Datenerfassungsplattform zur Aufnahme von analogen, biomedizinischen Sensorsignalen auf der Basis eines Arduino Dues

vorgelegt von: Imke Haase
Matrikelnummer: 2047309
Fakultät: Life Sciences
Studiengang: Medizintechnik

Erstgutachter: Prof. Dr. Bernd Kellner
Zweitgutachter Prof. Dr. Philipp Rostalski

In Kooperation mit:

Dräger

Lübeck, den 14. Dezember 2015

Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle aus fremden Quellen wörtlich oder sinngemäß entnommenen Gedanken habe ich als solche kenntlich gemacht.

Die Arbeit war in gleicher oder ähnlicher Form bisher noch nicht Bestandteil einer Studien- oder Prüfungsleistung.

Lübeck, den 14. Dezember 2015

Imke Haase

Danksagung

Hiermit möchte ich mich bei all denjenigen herzlich bedanken, die mich während der Anfertigung meiner Bachelorarbeit unterstützt und motiviert haben.

Ganz besonders gilt mein Dank meinen Betreuern Dr. Ulf Pilz, Dr. Philipp Rostalski und Dr. Bernd Kellner, die mir jederzeit bei jeglichen Fragen und Problemen zur Seite standen, Alexander Prehn für die Einführung in LaTeX, Albert Jäger für die Vorarbeit, der Firma Drägerwerk AG & Co. KGaA dafür, dass ich in diesem Unternehmen forschen und arbeiten durfte, Herrn Schäfer von Mathworks für die geduldige Unterstützung und meinen Korrekturlesern.

Nicht zuletzt möchte ich mich bei meinem Freund Patrick bedanken, der mich immer wieder aufgebaut und motiviert hat und besonders auch bei meinen Eltern. Dank Eurer Unterstützung war mir dieser Weg überhaupt erst ermöglicht.

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der Software-Entwicklung für ein Board zur Aufnahme und Speicherung von analogen, biomedizinischen Sensorsignalen und zur Ansteuerung von Aktoren. Die Hardware wird durch ein Arduino Due Board auf Basis eines ARM Cortex M3 Mikrocontroller (µC) und eine von Dräger produzierte zusätzliche Aufsteckplatine realisiert. Durch dieses sogenannte 24-Bit-Shield wird das Arduino Due Board unter anderem um einen hochauflösenden 24-Bit-Analog-to-Digital-Converter des Typs ADS1256 und einen SD-Karten-Slot erweitert.

Ziel ist es mit der entwickelten Software ein mobiles, kostengünstiges System bereit zu stellen, das sowohl zur Aufnahme von analogen, biomedizinischen Signalen bei Langzeittests, als auch zur Realisierung kleiner Regelkreise verwendet werden kann.

Die Umsetzung der Softwareanforderungen für die Datenerfassungsplattform in Quellcode wurde durch die Entwicklungsumgebung „Atmel Studio 6.2“ in der Programmiersprache C realisiert. Es werden die Grundlagen und die Implementierung der Kommunikationsschnittstelle Serial Peripheral Interface, des Analog-to-Digital-Converter, der SD-Karte, des Dateisystems File-Allocation-Table-32 und der Ansteuerung von Aktoren über Pulsweitenmodulation beschrieben. Dabei wird auf die notwendigen Initialisierungen und die Anwendung der entwickelten Software eingegangen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADC	Analog-to-Digital-Converter
ASF	Atmel Studio Framework
CAN	Controller-Area-Network
CRC	Cyclic Redundancy-Code
CPU	Central Processing-Unit
CS	Chip-Select-Leitung
DAC	Digital-to-Analog-Converter
DMA	Direct Memory-Access
DRDY	Data Ready
FAT	File-Allocation-Table
GCC	GNU-Compiler-Collection
GNU	GNU's Not Unix
GPIO-Pin	General Purpose Input/Output Pin
GUI	Graphical User-Interface
IC	Integrated Controller
ICE	In-Circuit-Emulator
IDE	Integrated Development-Environment
I/O	Input/Output
JTAG	Joint Test Action Group
LED	Light-Emitting Diode
µC	Mikrocontroller
MISO	Master-In-Slave-Out-Leitung
MOSI	Master-Out-Slave-In-Leitung
MSB	Most Significant Bit
PC	Personal Computer
PGA	Programmable Gain-Amplifier
PWM	Pulsweitenmodulation
RDATA	Read-Data-Befehl
SCK	System-Clock
SD-Karte	Secure-Digital-Memory-Karte

SMB	Sub-Miniature-B
SNR	Signal-to-Noise-Ratio
SPI	Serial Peripheral Interface
TWI	Two-Wire Interface
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Diese Bachelorarbeit entstand in Zusammenarbeit mit der Drägerwerk AG & Co. KGaA. Das Unternehmen ist einer der weltweit führenden Hersteller sicherheitstechnischer und medizintechnischer Produkte. Dazu zählen vorwiegend Personenschutzausrüstung, Gasmesstechnik, Produkte zur Beatmung, für die Narkose und zur kontinuierlichen Überwachung von Vitalparametern. In der Entwicklung, Forschung oder Wartung dieser Produkten, müssen häufig die gemessenen Sensorwerte aufgenommen, digital weiterverarbeitet und Aktoren, wie Ventile oder Antriebe gesteuert werden.

1.1. Problemstellung

Bei der Firma Drägerwerk AG & Co. KGaA wird zur Softwareentwicklung bisher vorwiegend ein System der Firma dSPACE GmbH genutzt. Der Nachteil dieses Systems liegt in dem hohen Anschaffungspreis und der erschwerten Mobilität, die in der Größe begründet sind. Eine andere Methode der Softwareentwicklung ist die Entwicklung direkt auf der Ziel-Hardware, dies ist jedoch häufig nicht möglich. Zum einen kann das darin begründet liegen, dass die Ziel-Hardware zu leistungsschwach zur direkten Entwicklung ist, da die Optimierung des Quellcodes hinsichtlich Codegröße und Performance häufig erst zum Ende der Entwicklungs- und Testphase geschieht. Zum anderen wird die Hardware eines Produkts oft erst nach der Entwicklung der Software entwickelt, da die Hardware an die Anforderungen der Software angepasst wird.

Ziel der vorliegenden Arbeit ist es, eine Datenerfassungsplattform zu entwickeln, mit der es möglich ist, bis zu acht hochauflösende Sensorsignale parallel aufzunehmen und nach Bedarf auf einer SD-Karte zu speichern. Zudem sollen vom Anwender auf der Plattform Regelkreise implementiert werden können, um Aktoren mittels Pulsweitenmodulation anzusteuern. Durch diese Funktionalitäten soll eine kostengünstige und flexible Alternative für viele Anwendungsfälle des dSPACE Systems und der Entwicklung auf der Ziel-Hardware entstehen.

Der Schwerpunkt dieser Arbeit liegt dabei in der Entwicklung der Software für die Datenerfassungsplattform. Als Grundlage dienen die Ergebnisse der Vorarbeit von Herrn Albert Jäger während eines Praktikums bei der Drägerwerk AG & Co. KGaA. Im Rahmen seiner Arbeit fand die Wahl und die Entwicklung der eingesetzten Hardware statt[10].

Das Arduino Due Board wurde als Fundament der Hardware gewählt, da es aufgrund seiner Hardware-Architektur Echtzeitfähigkeit gewährleistet [9], die für die Umsetzung von Regelkreisen eine notwendige Voraussetzung darstellt. Der auf dem Arduino Due Board verbaute Analog-to-Digital-Converter hat eine Auflösung von 12 Bit. Viele Sensoren in medizintechnischen und sicherheitstechnischen Produkten, wie beispielsweise elektrochemische Sensoren, generieren jedoch Ströme im Bereich von wenigen µA bis zu einigen Hundert µA. Als Anforderung an den Analog-to-Digital-Converter wurde daher eine Auflösung von 24 Bit gestellt. Zudem soll auf der Plattform die Möglichkeit zur Speicherung großer Mengen Messdaten bestehen. Die Daten sollen außerdem auf einfachem Weg von einem beliebigen PC auslesbar sein. Dafür bietet sich eine SD-Karte mit einer File-Allocation-Table Struktur an. Um die Funktionalitäten des Arduino Due Board um die zusätzlichen Anforderungen zu erweitern, wurde daher in der Vorarbeit ein Board entwickelt [10], welches im Folgenden als „24-Bit-Shield“ bezeichnet wird. Dieses lässt sich auf das Arduino Due Board aufstecken und es stellt beispielsweise einen 24-Bit-Analog-to-Digital-Converter, einen Steckplatz für eine SD-Karte und vier MOSFET-Treiberbausteine bereit.

Die Realisierung der Software unterteilt sich in verschiedene Unteraufgaben. Dazu zählt die Ansteuerung und Initialisierung des Arduino Due Boards, die Herstellung einer Kommunikation zwischen dem Mikrocontroller (µC) und peripheren Bauteilen über das Bussystem Serial Peripheral Interface, die Entwicklung eines Treibers zur Ansteuerung des Analog-to-Digital-Converters, die Initialisierung des Gerätetreibers der SD-Karte, der Implementierung eines File-Allocation-Table-Dateisystems auf der SD-Karte und der zeitlichen Koordinierung des Programmablaufs.

1.2. Aufbau der Arbeit

Die Arbeit ist in sechs Kapitel gegliedert und beginnt mit diesem Kapitel 1 „Einleitung“, wodurch dem Leser ein Überblick über die Arbeit gegeben werden soll. Es werden die Voraussetzungen und die Anforderungen an die Datenerfassungsplattform

genannt und der zum Beginn der Arbeit vorliegende Stand betrachtet. Zudem wird die zugrundeliegende Vorarbeit, welche die verwendete Hardware betrifft, beschrieben.

Die folgenden zwei Kapitel bilden den fachlichen Teil, wobei zwischen allgemeingültigen Grundlagen und der hardwarespezifischen Implementierung differenziert wird. In dem Kapitel 2 „Grundlagen“ wird auf die Details der zu verwendenden Hardware eingegangen. Anschließend wird die Funktionsweise der Module der zu entwickelnden Software allgemein erläutert. Dazu zählen das Serial Peripheral Interface, der Analog-to-Digital-Converter, die Secure-Digital-Memory-Karte, die File-Allocation-Table und die Pulsweitenmodulation.

Im darauf folgenden Kapitel 3 „Entwurf und Implementierung“ wird beschrieben, wie diese Software-Module an die Hardware angebunden wurden und wie sie in C-Quellcode umgesetzt wurden.

Abschließend werden die Ergebnisse dieser Arbeit in dem Kapitel 4 „Ergebnisse“ zusammengefasst. Im Kapitel 5 „Diskussion“ wird ein Ausblick auf mögliche Erweiterungen gegeben.

2. Grundlagen

In diesem Kapitel soll auf die Details der zu verwendenden Hardware eingegangen werden. Anschließend werden die verschiedenen Elemente der zu entwickelnden Software und deren Verknüpfungspunkte mit der Hardware näher erläutert.

2.1. Hardware

In den folgenden Abschnitten wird zum einen die Hardware der Datenerfassungsplattform beschrieben. Dazu zählen das Arduino Due Board, das 24-Bit-Shield, die Serial Peripheral Interface-Schnittstelle und die Sub-Miniature-B-Steckverbindung. Zum anderen wird die Hardware vorgestellt, die für die Entwicklung verwendet wurde. Im Rahmen dieser Arbeit mussten keine weitere Änderungen an der Hardware vorgenommen werden.

2.1.1. Arduino Due Board

Das Mikrocontroller-System wird von einem Arduino Due Board Model „DUE R3“ bereitgestellt. Bei dem Arduino Due Board handelt es sich um ein Entwicklungsboard, welches von der Gründungsgruppe Arduino LLC und den Produzenten Arduino S.r.l. entworfen wurde und vertrieben wird. Das Arduino Due Board wurde ursprünglich für Einsteiger in der Programmierung von eingebetteter Software entwickelt. Inzwischen wird es auch von erfahrenen Anwendern als Rapid-Prototyping-Tool genutzt, um schnell neue Funktionalitäten testen zu können, ohne dafür zeitaufwendige Schaltungen aufbauen zu müssen. Das Board besteht aus einem µC und diversen analogen und digitalen Ein- und Ausgängen. Für die Arduino Boards wird vom Hersteller zudem eine eigene Integrated Development-Environment (IDE) zum freien Download angeboten [1]. Näheres dazu wird im Abschnitt 2.2 „Software“ im Kapitel „Grundlagen“ beschrieben. Hard- und Software des Arduino Due Boards sind quelloffen.

Tab. 2.1.: Kenndaten des Arduino Due Boards (angelehnt an [1])

Eigenschaft	**Kennzahl**
Mikrocontroller	ARM Cortex-M3 (AT91SAM3X8E)
Prozessor	32 Bit Prozessor mit ARM Architektur
CPU	85 MHz
SRAM	96 KByte
Flash Speicher	512 KByte
Digital I/O Pins	54 (davon 12 als PWM Ausgänge nutzbar)
ADC	2x 12 Bit
ADC-Eingänge	12 Analogeingänge (12 Bit)
Serielle Schnittstellen	4 UARTs, SPI Header, CAN Bus, TWI
Versorgungsspannung	Power Jack 6 bis 16 V (7 bis 12 V empfohlen)
Betriebsspannung	3.3 V
Programmierschnittstelle	JTAG Header, Native USB Port, Programming USB Port
LED	je eine RX, TX, ON und L (programmierbar)

Abb. 2.1.: Frontseite des Arduino Due Boards [1]

2.1.2. 24-Bit-Shield

Wie Abschnitt 1.1 „Problemstellung“ im Kapitel „Einleitung“ beschrieben, genügt das Arduino Due Board den Anforderungen für die Datenerfassungsplattform alleine nicht. Besonders die fehlende Möglichkeit zum Speichern der aufgenommen Daten auf einem mobilen Austauschmedium, die geringe Auflösung des internen Analog-to-Digital-

Converter des Arduino Due Boards und der begrenzte Kapazitätsdrift dessen Ausgänge machten die Entwicklung einer zusätzlichen Hardware erforderlich. Daher wurde während der Tätigkeit von Albert Jäger das sogenannte 24-Bit-Shield entwickelt, welches die gewünschten Erweiterungen in Form einer Leiterplatte mit zusätzlicher Peripherie für das Arduino Due Board mit sich bringt. Das 24-Bit-Shield wird mit seiner Stiftleiste auf die Buchsenleiste des Arduino Due Board aufgesteckt. Es bietet die Möglichkeit selbst noch ein weiteres Shield zu tragen und zu versorgen, da seine Stiftleiste nach oben hin wiederum eine Buchsenleiste bildet. Auf dem 24-Bit-Shield befinden sich unter anderem ein 24-Bit-Analog-to-Digital-Converter, ein Instrumentenverstärker, vier MOSFET-Treiberbausteine und ein SD-Karten-Steckplatz (Vgl. Tab. 2.2).

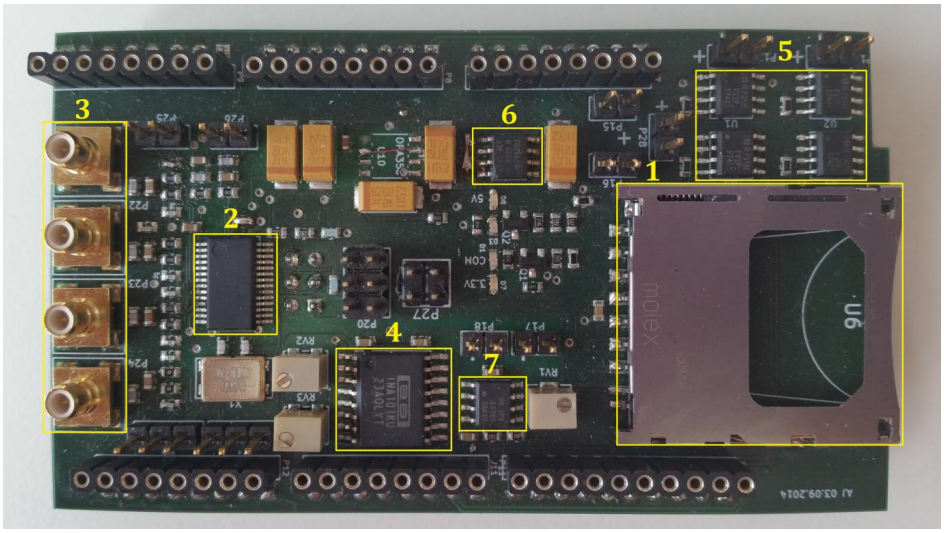

Abb. 2.2.: Frontseite des 24-Bit-Shields mit Referenz zu Tab. 2.2

Tab. 2.2.: Bauteile auf dem 24-Bit-Shield [10]

Nr.	Stk.	Bezeichnung	Hersteller	Herstellerbez.
1	1	SD-Karten-Konnektor	Molex	67600-8002[12]
2	1	24 Bit ADC	Texas Instruments	TI ADS1256[18]
3	4	SMB-Buchse	IMS Connector Systems	739287
4	1	Instrumentenverstärker	Texas Instruments	TI INA101KU[15]
5	4	Treiberbaustein	International Rectifier	IR F7401[8]
6	1	Referenzspannung	Texas Instruments	TI REF5025K[17]
7	1	Operationsverstärker	Texas Instruments	TI OPA177GS[16]

2.1.3. Zubehör

In-Circuit-Emulator:

Zur Programmierung des Arduino Due Boards mit der Integrated Development-Environment (IDE) „Atmel Studio“ von Atmel Corporation, welche bei der Erstellung dieser Arbeit verwendet wurde (siehe Abschnitt 2.2.1 „Integrated Development-Environment“ im Kapitel „Grundlagen“), ist es mit dem Plugin Visual Micro von Visual Micro Limited ebenfalls möglich, auf einen externen Programmierer zu verzichten. Zur Entwicklung und zum Test von embedded Software ist es jedoch durchaus sinnvoll einen In-Circuit-Emulator (ICE) zu verwenden. Damit ist es möglich, die aktuellen Werte der Register, der Ein- und Ausgänge und aller Variablen an jedem Teilschritt im Programmablauf zu beobachten. Zudem hat man Einblick in den Programmzähler und den Stapelzeiger des µC. Außerdem lassen sich mit dem ICE Hardware-Breakpoints setzen, ohne das Programm neu kompilieren zu müssen. In dieser Arbeit wurde zu diesem Zweck der ATMEL-ICE von Atmel Corporation verwendet. Der ICE wird auf der dem PC zugewandten Seite mit dem Micro-B-Stecker des Universal Serial Bus (USB)-Kabels verbunden, welches im obigen Fall zur Programmierung ohne ICE verwendet werden würde. Die Ausgangsseite des ICE wird mit der JTAG Schnittstelle des Arduino Due Boards verbunden (Vgl. Abb. 2.3). Bei richtiger Verbindung leuchtet am ICE die am Rande gelegene Light-Emitting Diode (LED) dauerhaft grün auf.

Schnittstselle:

Zur Programmierung des Arduino Due Boards mit der mitgelieferten Arduino IDE wird lediglich ein 2.0 USB-Kabel benötigt. Dies ist möglich, da auf dem Board bereits ein Bootloader integriert ist, der einen zusätzlichen Programmierer überflüssig macht.

Zur Aufnahme der teilweise hochempfindlichen, biomedizinischen, analogen Sensorsignale werden Hochfrequenzsteckverbinder verwendet. Diese sind durch ihren koaxialen Aufbau unempfindlicher gegenüber Störsignalen. Auf dem 24-Bit-Shield sind 4 Hochfrequenzsteckverbinder des Typs SMB verbaut.

Zur Kommunikation des µC über das Serial Peripheral Interface Bussystem wurde eine 6er Stiftleiste von dem 24-Bit-Shield nach unten auf die entsprechenden Pins am Arduino Due Board geführt. Dieselben Pins werden über eine Stiftleiste von dem 24-Bit-Shield außerdem nach oben geführt, sodass sich weitere Hardware-Peripherie über die Serial Peripheral Interface (SPI)-Schnittstelle anschließen lässt.

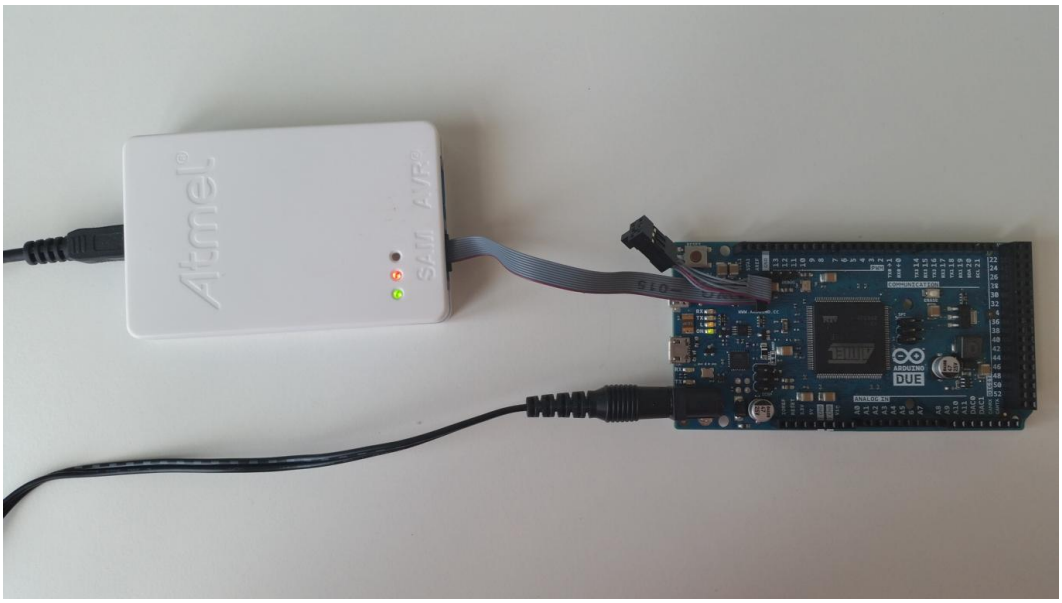

Abb. 2.3.: Anschluss des Atmel ICE an die JTAG-Schnittstelle des Arduino Due Board

2.2. Software

In den folgenden Unterabschnitten soll zuerst beschrieben werden, welche IDEs zur Programmierung in Frage kommen, welche Vor- und Nachteile sie jeweils mit sich bringen und welche IDE für die Programmierung verwendet wurde. Anschließend werden die Grundlagen und Kennzahlen der verschiedenen Software-Module Serial Peripheral Interface, Analog-to-Digital-Converter, Secure-Digital-Memory-Karte und File-Allocation-Table näher erklärt, um im Kapitel 3 „Entwurf und Implementierung“ dieses Wissen für das Design des Quellcodes anzuwenden.

2.2.1. Integrated Development-Environment

Eine integrierte Entwicklungsumgebung (engl. Integrated Development-Environment (IDE)) ist ein Softwarepaket, bestehend aus verschiedenen Anwendungsprogrammen, die gemeinsam die Aufgabe der Softwareentwicklung erfüllen. Zu den enthaltenen Komponenten zählen für gewöhnlich ein Texteditor, ein (oder mehrere) Compiler, ein Linker, ein Debugger und idealerweise auch eine Funktion zur Quelltextformatierung.

Wie im Abschnitt 2.1.1 „Arduino Due“ im Kapitel „Grundlagen“ angeschnitten, existiert für die Programmierung von Arduino Boards die IDE „Arduino IDE“. Diese ist im Internet frei erhältlich. Programmiert werden Arduino Boards in den höheren Programmiersprachen C bzw. C++. Die Arduino IDE ist eine Java Anwendung und basiert auf Processing. In der Arduino IDE sind einige Bibliotheken und fertige Programmbeispiele für diverse Arduino Boards bereits enthalten. Ein Projekt in der Arduino IDE wird als Sketch bezeichnet. Der Aufbau eines solchen Sketches unterscheidet sich etwas zu einem Standard Main-File. Es gibt keine Main-Funktion, sondern eine Setup- und eine Loop-Funktion, die für den Anwender versteckt von einer Main-Funktion aufgerufen werden.

Große Nachteile der Arduino IDE sind die fehlende Möglichkeit zur Strukturierung des Programms durch Headerfiles, die fehlende Option zum Springen in den Bibliotheksquelltext und eine fehlende Funktion der Quelltextformatierung. Zudem ist es nicht möglich über die Arduino IDE einen ICE zu verwenden, um die Analysemöglichkeiten zu erhalten, die in dem Abschnitt 2.1.3 „Zubehör“ im Kapitel „Grundlagen“ beschrieben wurden. Aus diesen Gründen wurde sich zur Bearbeitung dieser Arbeit gegen die Arduino IDE Entwicklungsumgebung entschieden.

Als alternative Entwicklungsumgebung wurde die ebenfalls frei erhältliche Software Atmel Studio 6.2 verwendet. Atmel Studio basiert auf Visual Studio von Microsoft und wird von Atmel Corporation vertrieben, welche ebenfalls Hersteller des auf dem Arduino Due Board verwendeten µC ist.

Atmel Studio bietet die Möglichkeit für das Arduino Due Board spezifische Projekte mit bereits vordefinierter Peripherie zu erzeugen. Da eingebettete Softwareprogrammierung sehr nah an der Hardware ist, bietet Atmel Studio für Board-Projekte nur die Möglichkeit der Programmierung in der Programmiersprache C. Als Toolchain wird in Atmel Studio standardmäßig GNU's Not Unix (GNU) GCC Compiler verwendet. Entwickelt wurde auf einem Dell Personal Computer (PC) mit einem Windows 7 (64 Bit) Betriebssystem.

Zur Erstellung eines neuen Projektes für das Arduino Due Board in Atmel Studio werden die Schritte aus Abb. 2.4 vorgenommen. Anschließend muss der ICE eingerichtet werden. Die Schritte dazu der Abb. 2.5 zu entnehmen.

Abb. 2.4.: Ablauf zur Erstellung eines Projekts für den Arduino Due in Atmel Studio

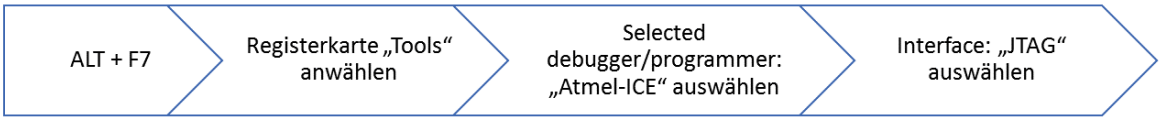

Abb. 2.5.: Einrichtung des Programmers/Debuggers in Atmel Studio

Zusätzlich lässt sich unter View → Toolbars → Atmel Debugger eine Leiste aktiveren, die die Buttons für die Debug-Funktionalitäten des Atmel-ICE, wie beispielsweise den Input/Output (I/O) View anbietet.

Atmel Studio stellt viele auf das Arduino Due Board zugeschnittene Bibliotheken über das ASF zur Verfügung. Beispielsweise wird eine Bibliothek zur Steuerung des Serial Peripheral Interface-Controllers angeboten. ASF Bibliotheken lassen sich über den Menüpunkt ASF → ASF Wizard importieren. Über ASF Explorer erreicht man die Dokumentation der Bibliotheken.

Um lediglich die Oberfläche und die Komfortabilität des Atmel Studio zu nutzen, aber weiterhin die Struktur der Arduino Sketches zu verwenden, bietet sich das kostenpflichtige Plugin „Arduino Micro“ an. Damit erhält man jedoch nicht die Möglichkeiten zum Testen, zur Fehlersuche und Fehlerbeseitigung die der Atmel-ICE mit sich bringt, da Arduino Micro den integrierten Bootloader zum Programmieren verwendet.

2.2.2. Serial Peripheral Interface

Die Kommunikation des µC zum Analog-to-Digital-Converter und zur SD-Karte erfolgt jeweils über das Bussystem SPI. SPI ist eine weit verbreitete Kommunikationsschnittstelle, die auf dem Master-Slave Prinzip beruht. In diesem Fall ist das Arduino Due Board der Master und der ADC und die SD-Karte sind jeweils Slaves. In der dieser Arbeit zugrundeliegenden Konstellation wird für die Datenübertragung wird ein Takt benötigt, der vom Mikrocontroller vorgegeben wird.

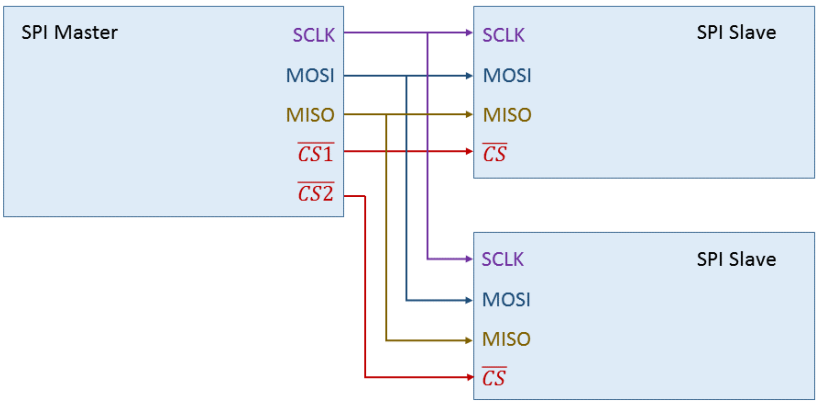

Abb. 2.6.: SPI-Verschaltung eines Masters mit zwei Slaves. Angelehnt an [2]

Eine SPI-Verbindung zwischen dem Master und einem Slave benötigt insgesamt vier Leitungen. Für jeden weiteren Slave ist eine zusätzliche Leitung erforderlich (Vgl. Abb. 2.6):

- **System-Clock (SCK)**: Hierüber wird der Takt vom Master an alle Slaves angegeben, in dem Datenpakete gesendet und empfangen werden. Diese Leitung besteht für alle Slaves jeweils zwischen dem Master und dem Slave.

- **Master-Out-Slave-In-Leitung (MOSI)**: Diese Leitung dient der Datenübertragung vom Master zum Slave.

- **Master-In-Slave-Out-Leitung (MISO)**: Diese Leitung dient der Datenübertragung vom Slave zum Master.

- **Chip-Select-Leitung (CS)**: Von dieser Verbindung muss es pro Slave jeweils eine geben. Hiermit wählt der Master denjenigen Slave an, mit dem er im Folgenden kommunizieren möchte. Dazu muss diese Leitung in der Regel „low“ geschaltet werden.

Das Schreiben vom Master zum Slave und das Lesen der vom Slave ausgehenden Daten erfolgt parallel, sodass der Master für eine Antwort auf seine aktuelle Anfrage erst einen neuen Befehl bzw. Dummy-Daten senden muss.

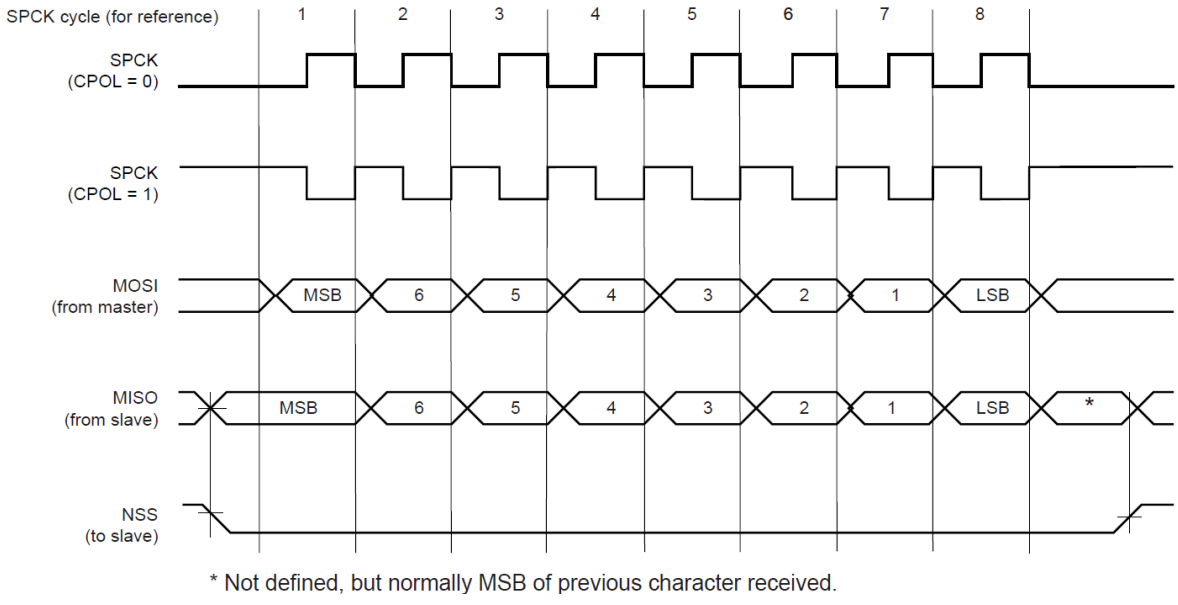

Abb. 2.7.: SPI-Timing der Datenübertragung mit Phase = 0 [2]

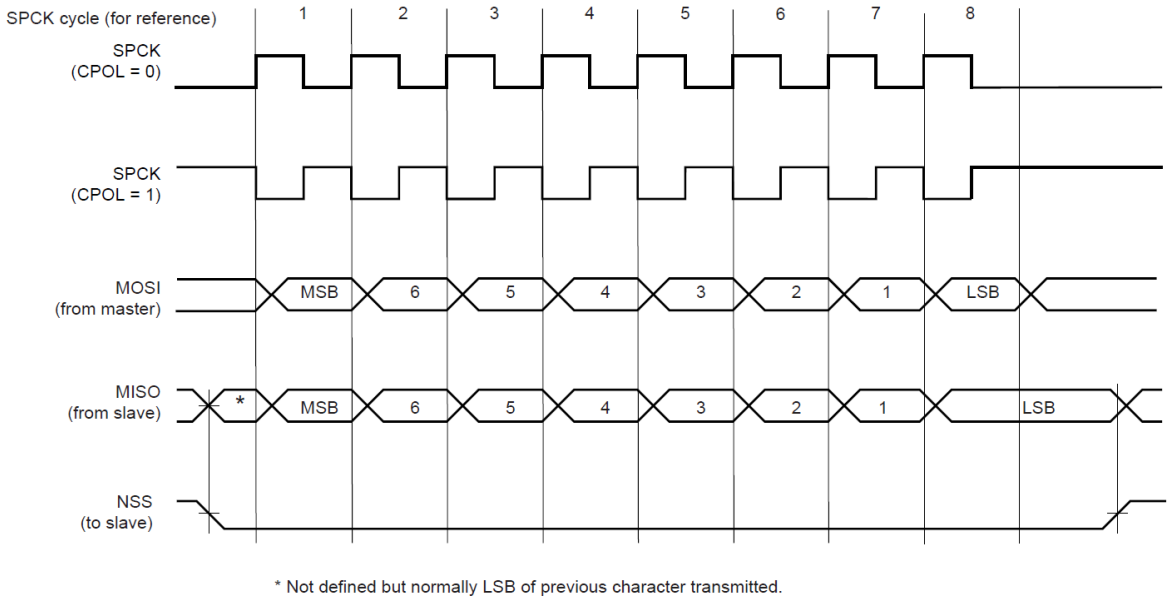

Abb. 2.8.: SPI-Timing der Datenübertragung mit Phase = 1 [2]

Für die Verwendung von SPI als Kommunikationsschnittstelle gibt es insgesamt vier Modi. Welcher Modus für einen Slave am Master jeweils eingerichtet werden muss, ist aus dem Datenblatt des Slaves zu entnehmen. Die Modi unterscheiden sich anhand zweier Eigenschaften: der Polarität und der Phase. Diese beiden Faktoren können jeweils den Wert 0 oder 1 annehmen. Die Phase gibt an, ob der Sende bzw. Lesevorgang für die MOSI- und für die MISO-Leitung jeweils zur ersten oder zweiten Flanke der Taktleitung erfolgen soll. Die Polarität gibt an, ob die Taktleitung im Ruhezustand high (1) oder low (0) annimmt. In Abb. 2.7 sind die beiden Modi mit der Phase = 0 und in Abb. 2.8 die beiden Modi mit der Phase = 1 grafisch dargestellt. Je Diagramm sind für die Polarität die beiden möglichen Zustände enthalten.

Für jeden Slave lassen sich in den Registern des µC individuell Einstellungen vornehmen und in einem Register abspeichern, sodass beim Wechsel zwischen den Slaves die Einstellungen nicht von Neuem erfolgen müssen [2].

Welche Einstellungen in dem konkreten Fall dieser Arbeit vorgenommen werden müssen, wird im Kapitel 3 „Entwurf und Implementierung“ im Abschnitt 3.1 im Kapitel „Entwurf und Implementierung“ Serial Peripheral Interface (SPI) beschrieben.

2.2.3. Analog-to-Digital-Converter

Zum Wandeln analoger in digitale Signale wird ein Analog-to-Digital-Converter (ADC) benötigt. Dabei gibt es verschiedene Verfahren, die sich vor allem durch die Wandlungszeit, Abtastrate, die Auflösung und durch den Schaltungsaufwand und damit auch im Preis unterscheiden. Bei dem auf dem 24-Bit-Shield verbauten ADC von Texas Instruments, Typ ADS1256, handelt es sich um einen Delta-Sigma Wandler vierter Ordnung. Delta-Sigma Wandler sind hochauflösende Wandlertypen bei gleichzeitig relativ hoher Abtastrate. Sie bestehen im Wesentlichen aus zwei Blöcken. Den ersten Block bildet der Delta-Sigma-Modulator, bestehend aus einem Komparator (Delta), einem Integrator (Sigma) und einem 1 Bit Digital-to-Analog-Converter (DAC). Am Komparator wird die Differenz aus dem analogen Eingangssignal und dem 1 Bit gewandelten Ausgangssignal gebildet. Anschließend wird das Signal integriert. Als zweiter Block folgt das Tiefpass Filter. Je höher die Ordnung des Wandlers, desto besser wird das Signal-Rausch-Verhältnis (engl. Signal-to-Noise-Ratio (SNR)) [11, 14].

Der ADC ADS1256 besitzt einen Eingangsmultiplexer mit acht analogen Eingängen (Vgl. Abb. 2.9). Dadurch ergeben sich je nach Verwendung entweder acht single-ended Eingänge, bei denen eine gemeinsame vom Board bereitgestellte Masse verwendet wird (AINCOM), oder aber vier differentielle Eingänge, bei denen jeder Eingang einen eigenen Bezugspunkt hat. Bei der Verwendung von single-ended Eingängen können Störsignale, wie beispielsweise Rauschen oder die Netzfrequenz und deren Oberschwingungen, nicht vom zu messenden Signal unterschieden werden. Daher ist dieser Eingangstyp nur bei kurzen Leitungen und relativ hohen Spannungspegeln zu empfehlen. Bei der Verwendung differentieller Eingänge werden die Gleichtaktstörsignale durch Differenzbildung gedämpft oder im Idealfall ausgelöscht. Die verschiedenen Eingangstypen lassen sich auch kombinieren, sodass man beispielsweise zwei differentielle Eingänge und gleichzeitig vier single-ended Eingänge nutzen kann.

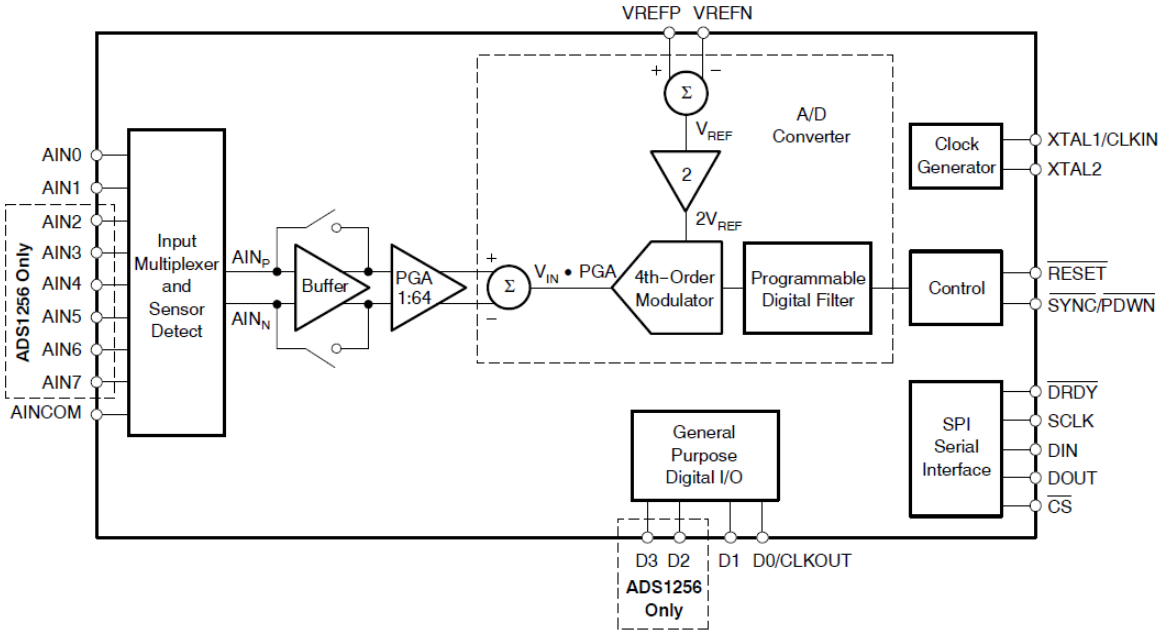

Abb. 2.9.: Blockschaltbild des ADS1256 [18]

Die Auflösung des ADS1256 liegt idealerweise bei 24 Bit und bietet damit $2^{24} = 16.777.216$ mögliche Zustände als Ergebnis der Wandlung.

Die Wandlungsrate gibt an, wie viele Wandlungen idealerweise maximal pro Sekunde durchgeführt werden können. Bei dem ADS1256 liegt die Wandlungsrate bei 30.000 Samples/Sekunde. Daraus ergibt sich eine Wandlungszeit von etwa $33,3\mu s$.

Der ADS1256 verfügt über einen Verstärker mit programmierbarem Verstärkungsfaktor (engl. Programmable Gain-Amplifier (PGA)). Den PGA bildet ein Operationsverstärker in Form eines invertierenden Verstärkers. Der integrierte PGA bietet als Verstärkungsfaktoren des Eingangssignals 1, 2, 4, 8, 16, 32 an.

Der Messbereich des ADS1256 liegt im Bereich der positiven und negativen zweifachen Referenzspannung (bei einem Verstärkungsfaktor von 1). Als Referenzspannung ist auf dem Board der Baustein REF5025 von Texas Instruments mit einer Spannung von 2,5 V verbaut. Daraus ergibt sich eine Spannung von -5 V bis +5 V als Messbereich. Der ADS 1256 hat einen Linearitätsfehler, das bedeutet eine maximale Abweichung zwischen der Soll-Kennlinie und der realen Kennlinie, von $\pm 0.0010\,\%$.

Da die Eingangswerte der Wandlung sowohl positiv als auch negativ sein können, die digitalen Ausgangswerte jedoch im Binärsystem ausgegeben werden, muss ein System

zur Transformation angewendet werden. Daher werden die Ausgangswerte am Zweierkomplement ausgerichtet. Der Vorteil der Darstellung im Zweierkomplement ist die Möglichkeit der Anwendung von Grundrechenoperationen. Das hochwertigste Bit (engl. Most Significant Bit (MSB)) wird im Zweierkomplement zur Kennzeichnung des Vorzeichens verwendet. Beträgt der Wert des MSB 1, so ist die Zahl negativ; bei einer 0 ist der Wert positiv.

Die Kommunikation zwischen dem µC und dem ADC wird in dieser Arbeit mittels SPI realisiert. Es ist auch möglich den ADC als serielle Übertragung zu realisieren, die manuell getaktet werden muss. Jedoch bedeutet das weitaus mehr Programmieraufwand und bietet keine Vorteile gegenüber dem SPI-Bus.

2.2.4. Secure-Digital-Memory-Karte

Die SD-Karte ist ein Speichermedium, dessen Flashspeicher beschreibbar, lesbar und löschbar ist. Der SD-Karten-Steckplatz befindet sich gut erreichbar auf der Oberseite des 24-Bit-Shields. Für die Entwicklung wurde eine SanDisk Ultra micro-SDHC Karte Class 10 mit 32 GB Speicherkapazität verwendet. SanDisk Corporation ist der Hersteller; „micro“ gibt die physikalische Größe der Karte an; „Class 10“ ist ein Indikator für die Geschwindigkeit. „SDHC“ steht für eine SD-Karte mit einer Speicherkapazität zwischen 4 und 32 GByte und wird auch als SD 2.0 bezeichnet [6]. Der Hauptunterschied zwischen SD-Karten mit Standardgröße und micro-SD-Karten ist die Gehäusegröße. Es gelten dieselben elektrischen Spezifikationen für beide Kartengrößen. Da der SD-Karten-Adapter auf dem Board jedoch für die Standardgröße von SD-Karten ausgelegt ist, wird ein Adapter von SanDisk verwendet um den Größenunterschied zu überbrücken. Die verwendete SD-Karte wirbt mit einer Lese- bzw. Schreibgeschwindigkeit von bis zu 80 Mbit/s bei sequentiellem Zugriff.

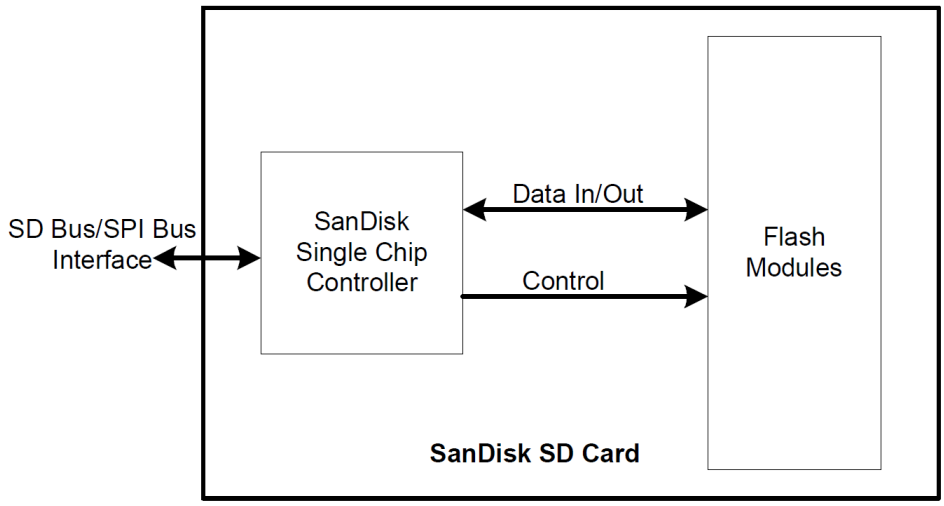

Abb. 2.10.: Aufbau der SD-Karte [13]

Wie in Abb. 2.10 ersichtlich, besteht die SD-Karte im Wesentlichen aus einem integrierten Controller (engl. Integrated Controller (IC)), dem eigentlichen Flashspeicher und dem SPI-/SD-Bussystem. Während der IC nur Befehle und Daten an den Speicher senden kann, kann der Speicher nur Daten an den IC schicken. Der IC kann zusätzlich über das Bussystem nach außen kommunizieren, um Befehle oder Daten entgegenzunehmen oder Daten nach außen zu übertragen. [13]

Grundsätzlich wird auf die bzw. von der SD-Karte jeweils nur ein Byte gesendet oder gelesen. In manchen Fällen müssen allerdings auch größere Daten übertragen werden. Dazu werden Blöcke verwendet, deren Größe immer ein Vielfaches eines Bytes darstellt. In Abb. 2.11 ist der Aufbau des Speichers nachzuvollziehen. Ein oder mehrere Blöcke werden als ein Sektor zusammengefasst [13]. Die Größe der Sektoren ist für eine SD-Karte einheitlich und beträgt in der Regel 512 KByte. Beim Löschen von Daten von der SD-Karte können nur vollständige Sektoren mit allen enthaltenden Blöcken gelöscht werden [13]. Bei Schreibvorgängen muss die Größe des Blocks immer exakt der Größe eines Sektors oder eines Vielfachen dessen entsprechen. Beim Lesevorgang beträgt die minimale Datenmenge ein Byte und die maximale gleicht der Größe eines Sektors. Weiterhin gibt es sogenannte WP (engl. für Write Protection) Groups, welche mehrere Sektoren zusammenfassen und für alle enthaltenden Sektoren einen Schreibschutz aktivieren können. Die Größe der WP Groups ist für jede SD-Karte festgelegt. Am Ende des Flash-Speichers der SD-Karte befindet sich ein geschützter Bereich, der ebenfalls in Blöcke und Sektoren aufgeteilt ist, der allerdings nicht zur Speicherung für Anwenderdaten zur Verfügung steht, da hier Inhaltsschutz-Operationen abgelegt werden können [13].

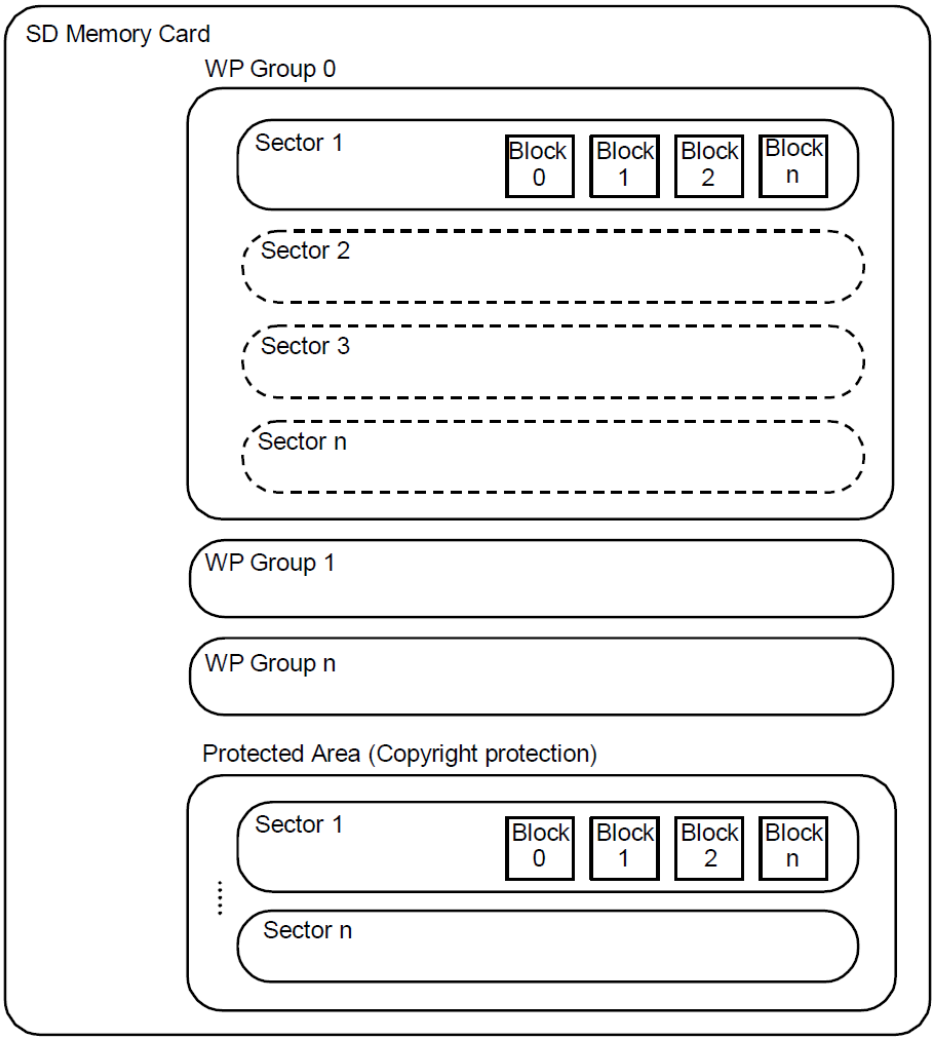

Abb. 2.11.: Aufbau des Speichers der SD-Karte [13]

Die Kommunikation mit der SD-Karte wird, wie auch bei dem ADC, durch einen SPI-Bus realisiert. Vor der ersten Verwendung der SD-Karte muss diese initialisiert werden. Die Initialisierungsroutine muss über eine serielle Datenübertragung über das SD-Bus-Protokoll bewerkstelligt werden. Hierzu sind eine Command-Leitung zur Übertragung von Kommandos an die SD-Karte und wahlweise eine oder vier Datenleitungen zur Übertragung von Datenpaketen an den µC nötig. Erst im Anschluss kann die Umstellung auf den SPI-Bus erfolgen. Während der seriellen Datenübertragung enthalten alle Kommandos und Datenpakete eine Cyclic Redundancy-Code (CRC)-Prüfsumme um die Korrektheit der Übertragung zu kontrollieren. Sobald die Übertragung auf den SPI-Standard umgestellt wurde, können die CRC-Prüfsummen ignoriert werden.

2.2.5. File-Allocation-Table

Die File-Allocation-Table (FAT) bietet eine Möglichkeit für den einfachen Austausch von Daten zwischen Systemen wie unter anderem PC, Smartphone, Digitalkamera, MP3-Player und Tablet [3]. Das FAT-Dateisystem gibt es in verschiedenen Ausführungen. Für Speicher ab einer Kapazität von 2 GB wird das FAT32-System verwendet. Die Bezeichnung FAT32 beruht darauf, dass jeder Eintrag im FAT-Abschnitt 32 Bit lang ist. Allerdings werden nur 28 Bit benutzt. Die restlichen 4 Bit sind reserviert. Entsprechend sind die Einträge beim FAT12- und FAT16-System 12 bzw. 16 Byte groß. Im Vergleich zu anderen Dateisystemen hat das FAT-System zwar eine schlechte Performance, doch es zeichnet sich durch seine Universalität gegenüber fast allen gängigen Betriebssystemen, wie beispielsweise Windows, OS X oder Linux und durch seinen einfachen Aufbau aus.

Aufgeteilt wird der Speicher auf dem Speichermedium in Sektoren, welche einer definierten festen Anzahl an Bytes entsprechen. In der Regel ist ein Sektor 512 Bytes groß. Eine definierte Anzahl an Sektoren wird wiederum zu sogenannten Clustern zusammengefasst.

Tab. 2.3.: Prinzipieller Aufbau des FAT-Systems

Abschnitt	**Bezeichnung**
1.	Reservierter Sektor Bereich
-1a	Volume Boot Record
-1b	Informationen zur FAT
-1c	weitere reservierte Sektoren
2.	File-Allocation-Table
3.	Datensektoren

Wie in Tab. 2.3 oben dargestellt, gliedert sich die Struktur des FAT32 Systems auf dem Datenträger in vier Bereiche. Der erste Bereich enthält das Volume Boot Record, allgemeine FAT-Informationen und eventuell noch weitere reservierte Sektoren. In dem Volume Boot Record wird der sogenannte Bootcode gespeichert. Dieser Code hat die Größe eines Sektors und ist zum Booten des Systems notwendig. Außerdem wird hier die Partitionstabelle hinterlegt, welche in der Regel bis zu vier Partitionen abbilden kann. Die Tabelle beinhaltet eine 16 Byte große Beschreibung pro Partition, worin grundlegende Informationen über das Dateisystem, wie beispielsweise der Type-Code des FAT-Systems (in unserem Fall FAT32), die Anzahl reservierter Sektoren pro Partition und der Beginn der jeweiligen Partition auf dem Speichermedium abgespei-

chert werden. Im Bereich 1b werden allgemeine Informationen über das Dateisystem abgespeichert. Darauf folgen optional weitere reservierte Sektoren.

Im darauf folgenden zweiten Bereich befindet sich die eigentliche File-Allocation-Table. Zur Gewährleistung einer höheren Datensicherheit, wird die FAT hier meist in zweifacher Ausführung abgelegt. Jede im Abschnitt 3. zu speichernde Datei nimmt ein oder mehrere Cluster des Speichers in Anspruch, abhängig von ihrer Größe. Dabei müssen die Cluster, die zu einer Datei gehören, nicht zwangsweise im Speicher hintereinander liegen, sondern liegen häufig verstreut. Jedes Cluster des Datensektor wird in der FAT durch einen Eintrag beschrieben. Diese 32 Byte großen Einträge beinhalten unter anderem den Dateinamen, die Dateierweiterung, die Erstellungszeit und das Datum des letzten Zugriffs. Daneben enthält ein Dateieintrag auch eine sogenannte Cluster-Nummer, woraus sich die Adresse des ersten Clusters der Datei ergibt. So ergibt die FAT eine verkettete Liste von Cluster-Nummern.

Der letzte Bereich enthält die tatsächlich zu speichernden Daten, welche sich jeweils aus einem oder mehreren Clustern zusammensetzen, wie oben bereits erwähnt. Dieser Bereich nimmt den größten Platz auf dem Speichermedium ein.

Die Umsetzung des FAT-Dateisystems in Software anhand einer Bibliothek, die Hierarchieebenen innerhalb der Bibliothek und die auf dieses Projekt angepassten Einstellungen, werden im Abschnitt 3.4 „File-Allocation-Table“ im Kapitel „Entwurf und Implementierung“ beschrieben.

2.2.6. Pulsweitenmodulation

Die Pulsweitenmodulation (PWM) stellt eine einfache Möglichkeit dar, um eine Ausgangsspannung variabel zu verändern. Angewendet wird dieses Verfahren beispielsweise um Aktoren anzusteuern oder Messwerte über lange Strecken zu übertragen. Der Vorteil der PWM liegt darin, dass keine Verfälschung der Signals durch eine abfallende Spannungshöhe entsteht.

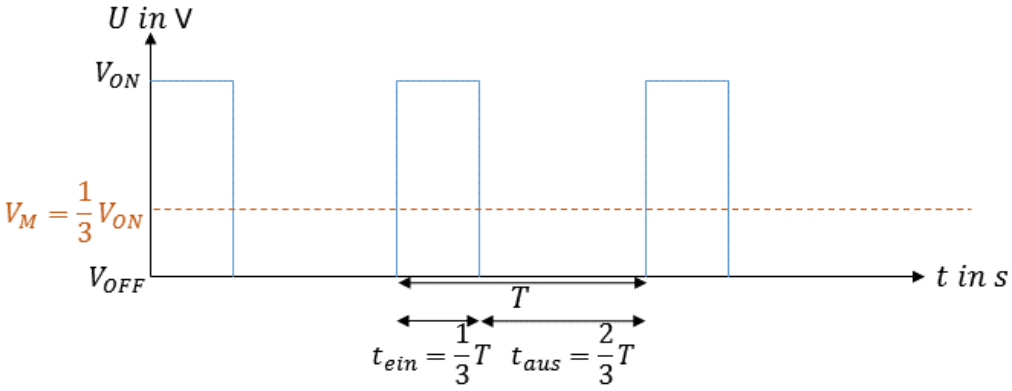

Abb. 2.12.: Beispiel eines PWM-Signals mit Tastverhältnis = 0,3

Als Grundlage für die PWM wird in unserem Fall ein digitales Rechtecksignal mit den Zuständen $V_{\mathrm{OFF}} = 0$ V und $V_{\mathrm{ON}} =$ bei 3,3 V verwendet. Die Frequenz des Signals ist konstant. Aufgrund des menschlichen Hörbereichs, der maximal etwa zwischen 16 Hz und 16 kHz liegt [7] wurde eine Frequenz für die Pulsweitenmodulation von 20 kHz festgelegt. Die Spannung wird durch die Änderung des Verhältnisses der Zeitdauer von V_{OFF} und V_{OFF} innerhalb einer Periode variiert. Dieses Verhältnis wird als Tastverhältnis bezeichnet und berechnet sich nach Formel (2.1).

$$\text{Tastverhältnis} = \frac{t_{\mathrm{ein}}}{t_{\mathrm{ein}} + t_{\mathrm{aus}}} = \frac{t_{\mathrm{ein}}}{T} \tag{2.1}$$

Aus dem Mittelwert über eine Periode T ergibt sich die resultierende Spannung V_M, die sich mit der Formel (2.2) [19] berechnen lässt. Durch die PWM lassen sich Ausgangsspannungen zwischen 0 V und 3,3 V realisieren.

$$V_{\mathrm{m}} = \frac{V_{\mathrm{ON}} \cdot t_{\mathbf{ein}}}{t_{\mathbf{ein}} + t_{\mathbf{aus}}} \tag{2.2}$$

3. Entwurf und Implementierung

In diesem Kapitel werden zunächst Themen behandelt, die die weiteren Abschnitte betreffen. Dazu zählen Konventionen der zugrundeliegenden Software, das Vorgehen beim Programmieren und die Struktur des Programms.

Konventionen

Zur Verbesserung der Lesbarkeit, Reduzierung der Fehleranfälligkeit, Erweiterbarkeit, Wiederverwendbarkeit und Wartbarkeit wurden für den Quellcode einige Konventionen angewendet. Um innerhalb der Modul-Quelltextdateien die Übersichtlichkeit zu gewährleisten, wurde folgende Reihenfolge eingehalten:

1. Modulkopf (siehe unten)
2. Include-Anweisungen
3. Define-Anweisungen für Konstanten
4. Define-Anweisungen für Makros
5. Definition von Datentypen (typedef)
6. Definition von globalen Konstanten (const)
7. Definition von globalen Variablen (extern)
8. Definition von lokalen Konstanten (static const)
9. Definition von lokalen Variablen (static)
10. Deklaration von lokalen Funktionen (static)
11. Definition von globalen Funktionen
12. Definition von lokalen Funktionen

Die Deklaration der globalen Variablen und Funktionen ist in eine Modul-Headerdatei auszulagern. Der Name der Modul-Headerdatei soll dem des Modul-Quelltextes entsprechen und soweit möglich soll die Modul-Headerdatei auch demselben Aufbau folgen.

Der Dateikopf einer Modul-Quelltextdatei soll dabei folgende Angaben beinhalten:

1. Modulname
2. Autor
3. Projektname
4. Kurzbeschreibung des Inhalts

Jedes Modul erhält ein Präfix, das aus drei Buchstaben besteht und das Modul identifiziert. Dieses Präfix wird den Namen aller Elemente, die innerhalb des Moduls deklarierten werden, vorangestellt. Dem Präfix folgt ein Unterstrich und der restliche Name. Je nach Typ des Elements, werden die Namen, wie in Tab. 3.1 aufgelistet, unterschiedlich dargestellt um die Lesbarkeit zu erhöhen.

Tab. 3.1.: Konventionen für Namensvergabe im C-Quellcode

Typ	**1. Buchstabe**	**Folgende Buchst.**	**Beispiel**
Defines	groß	groß	Adc_MAX_COUNTS
Konstanten	groß	groß	Adc_CHANNEL_ADDRESS
Makros	groß	groß	Adc_BYTES_TO_READ
Variablen	klein	klein	Adc_resolutionbits
Funktionen	groß	klein	Adc_WriteRegister(...)

Registerarchitektur für General Purpose Input/Output Pins

Ein Großteil der Peripherie ist über General Purpose Input/Output Pins (GPIO-Pins) verbunden. Auf dem µC sind mehrere Controller zur Verwaltung und Steuerung der GPIO-Pins und anderer Ein- und Ausgänge verbaut. Je ein Controller verwaltet 32 Leitungen. Für jeden GPIO-Pin lassen sich über den Controller viele Einstellungen vornehmen. Im Folgenden soll das Vorgehen zur Änderung von Einstellungen erklärt werden. Zudem werden die wichtigsten Einstellungen beschrieben, da diese in den anschließenden Kapiteln angewendet werden.

Wie in Tab. 3.1 zu erkennen, gibt es für die meisten Einstellungen jeweils drei Register. So beziehen sich beispielsweise die Register PIO_OER, PIO_ODR und PIO_OSR alle auf die Einstellung eines GPIO-Pins als Aus- bzw. Eingang. Ein Register ist jeweils immer für die Aktivierung und ein Register für die Inaktivierung der Funktion verantwortlich. Dabei betrifft diese Einstellung immer diejenigen Pins, die innerhalb dieses Registers mit einer Eins belegt werden. Bei einer Null wird die vorherige Einstellung beibehalten. Die Register zur Aktivierung und Deaktivierung bieten lediglich einen Schreibzugriff und keinen Lesezugriff. Nach dem Schreiben der genannten Re-

gister wird die Änderung in dem dritten Register abgespeichert, welches der besseren Lesbarkeit der aktuellen Konfigurierung dient. In diesem Statusregister werden alle Pins, die aktiviert wurden durch eine Eins und alle Pins, die deaktiviert wurden, durch eine Null dargestellt.

Offset	Register	Name	Access	Reset
0x0000	PIO Enable Register	PIO_PER	Write-only	–
0x0004	PIO Disable Register	PIO_PDR	Write-only	–
0x0008	PIO Status Register	PIO_PSR	Read-only	(1)
0x000C	Reserved			
0x0010	Output Enable Register	PIO_OER	Write-only	–
0x0014	Output Disable Register	PIO_ODR	Write-only	–
0x0018	Output Status Register	PIO_OSR	Read-only	0x0000 0000
0x001C	Reserved			
0x0020	Glitch Input Filter Enable Register	PIO_IFER	Write-only	–
0x0024	Glitch Input Filter Disable Register	PIO_IFDR	Write-only	–
0x0028	Glitch Input Filter Status Register	PIO_IFSR	Read-only	0x0000 0000
0x002C	Reserved			
0x0030	Set Output Data Register	PIO_SODR	Write-only	–
0x0034	Clear Output Data Register	PIO_CODR	Write-only	
0x0038	Output Data Status Register	PIO_ODSR	Read-only or(2) Read-write	–
0x003C	Pin Data Status Register	PIO_PDSR	Read-only	(3)
0x0040	Interrupt Enable Register	PIO_IER	Write-only	–
0x0044	Interrupt Disable Register	PIO_IDR	Write-only	–
0x0048	Interrupt Mask Register	PIO_IMR	Read-only	0x00000000
0x004C	Interrupt Status Register(4)	PIO_ISR	Read-only	0x00000000
0x0050	Multi-driver Enable Register	PIO_MDER	Write-only	–
0x0054	Multi-driver Disable Register	PIO_MDDR	Write-only	–
0x0058	Multi-driver Status Register	PIO_MDSR	Read-only	0x00000000
0x005C	Reserved			
0x0060	Pull-up Disable Register	PIO_PUDR	Write-only	–
0x0064	Pull-up Enable Register	PIO_PUER	Write-only	–
0x0068	Pad Pull-up Status Register	PIO_PUSR	Read-only	0x00000000
0x006C	Reserved			

Abb. 3.1.: Register des Parellel Input/Output Controllers [2]

Tab. 3.2.: Einstellungen für GPIO-Pins

Registernamen	**Funktion**
PIO_PER PIO_PDR PIO_PSR	Periphere Kontrolle über den GPIO-Pin inaktivieren (1) oder aktivieren (0)
PIO_OER PIO_ODR PIO_OSR	GPIO-Pin als Ausgang (1) oder als Eingang (0) schalten
PIO_SODR PIO_CODR PIO_ODSR	Setzt den Ausgangswert der GPIO-Pins fest, die als Ausgang konfiguriert wurden
PIO_IER PIO_IDR PIO_ISR	Aktiviert (1) oder Deaktiviert (0) den Interrupt, der bei einer Änderung des GPIO-Pins ausgelöst wird

Programmstruktur

Der Quelltext für die Datenerfassungsplattform ist in mehrere Unterdateien gegliedert. Die Unterdateien gliedern sich in Module bestehend aus einer .c-Datei und einer .h-Datei. Um Verwechslungen mit anderen .c- und .h-Dateien zu vermeiden werden die Unterdateien, die ein Modul bilden im Folgenden als Modul-Quelltextdatei und Modul-Headerdatei bezeichnet. Einige Dateien haben jedoch keine Zugehörigkeit zu einem Modul. Eine Datei ist die Main-Datei, welche die Main-Funktion beinhaltet, die wiederum nur aus Funktionsaufrufen besteht und den Programmablauf vorgibt. Die Datei „configurations.h“ beinhaltet keine Funktionen und fasst globale Variablen zusammen, die vom Anwender anpassbare Einstellungen betreffen. Darüber hinaus inkludiert eine Datei „asf.h“ alle Bibliotheken, die von den Modulen verwendet werden. Diese Bibliotheken werden von Atmel Studio über das Atmel Studio Framework (ASF) zur Verfügung gestellt. Die Bibliotheken des ASFs die für dieses Projekt verwendet wurden, werden in Tab. 3.3 aufgelistet.

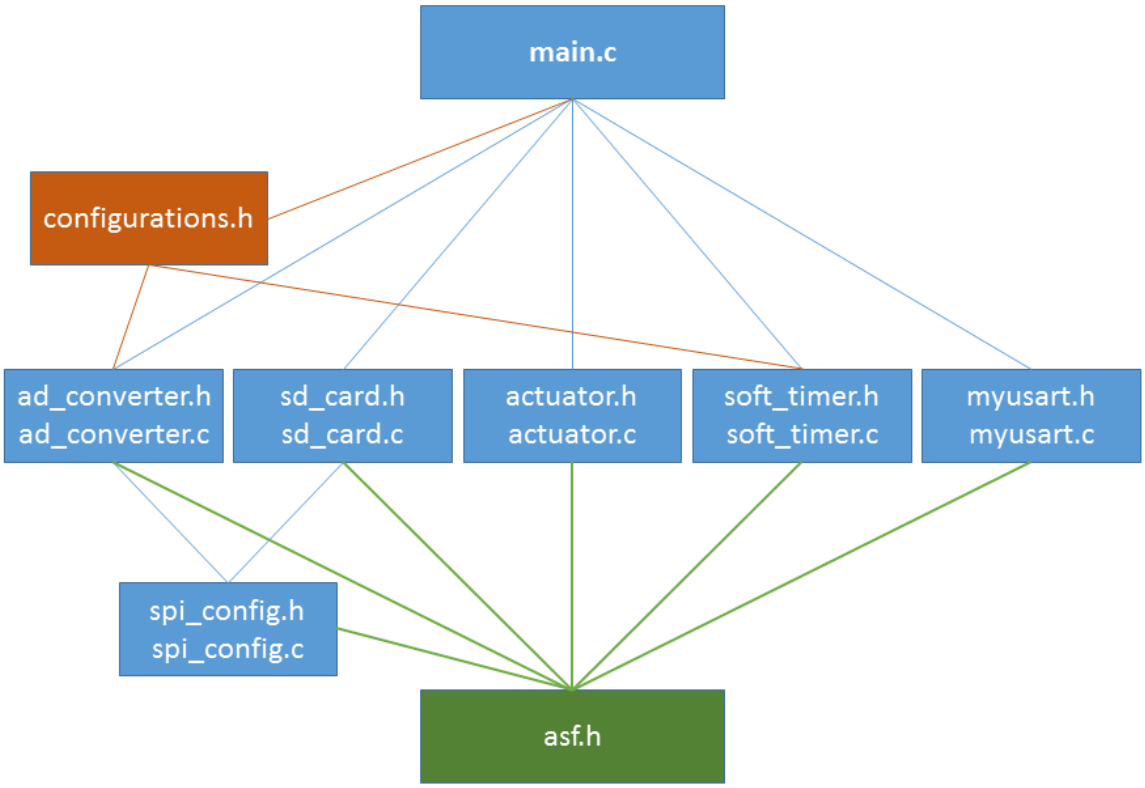

Abb. 3.2.: Unterteilung des Quellcodes in Unterprogramme

Die Module wurden anhand ihrer Funktionalitäten benannt als actuator, ad_converter, myusart, sd_card, soft_timer und spi_config. Wie in Abb. 3.2 zu sehen, ist in jedem Modul die asf.h- Datei inkludiert um die Bibliotheken innerhalb des Moduls zur Verfügung zu stellen. Das Modul spi_config beinhaltet Einstellungen die die SPI-Schnittstelle für den ADC und die SD-Karte konfiguriert. Daher wird dieses Modul lediglich in den Modulen ad_converter und sd_card eingebunden. Alle Module, bis auf die spi_config, werden wiederum in die main.c inkludiert. Die Datei configurations.h wird lediglich von den Modulen ad_converter und soft_timer verwendet.

Tab. 3.3.: Von dem ASF verwendete Bibliotheken

Bezeichnung
Delay routines
DMAC – DMAC Controller
FatFS file system
Generic board support
GPIO – General purpose Input/Output
Interrupt management – SAM implementation
IOPORT – General purpose I/O service
PIO – Parallel Input/Output Controller
SD/MMC stack on SPI interface
SPI – Common Standard SPI
Standard serial I/O (stdio) – SAM implementation
System Clock Control – SAM3X/A implementation
TC – Timer Counter
UART – Univ. Async Rec/Trans
USART – Serial interface

Hauptprogramm

Wie in Abb. 3.3 zu erkennen, werden zu Beginn der Main-Funktion 3.0.1, welche sich in der Datei main.c befindet, alle Initialisierungen vorgenommen. Anschließend werden in einer Endlosschleife die Hauptfunktionen der Module ad_converter, sd_card und actuator nacheinander aufgerufen. Dabei ist jede dieser Hauptfunktionen in einen eigenen Timer eingebettet und wird erst ausgeführt, nachdem dieser Timer erreicht wurde. Da der Timer des ADC kürzer ist, als der des Schreib-/Lesevorgangs für die SD-Karte, können mehrere Ergebnisse der Wandlung als ein Block auf die SD-Karte geschrieben werden. Dadurch wird das Programm effizienter gestaltet.

C-Funktion 3.0.1: `void main (void);`

Die folgenden Abschnitte sind nach Softwaremodulen gegliedert. Für jedes Softwaremodul wird, sofern vorhanden, jeweils die Pinbelegung, die Realisierung der Funktion zur Initialisierung und der Anwendungsfunktion erläutert. Mit Anwendungsfunktion sind dabei die Modulfunktionen gemeint, welche von der Endlosschleife der Main-Funktion 3.0.1 aus aufgerufen werden.

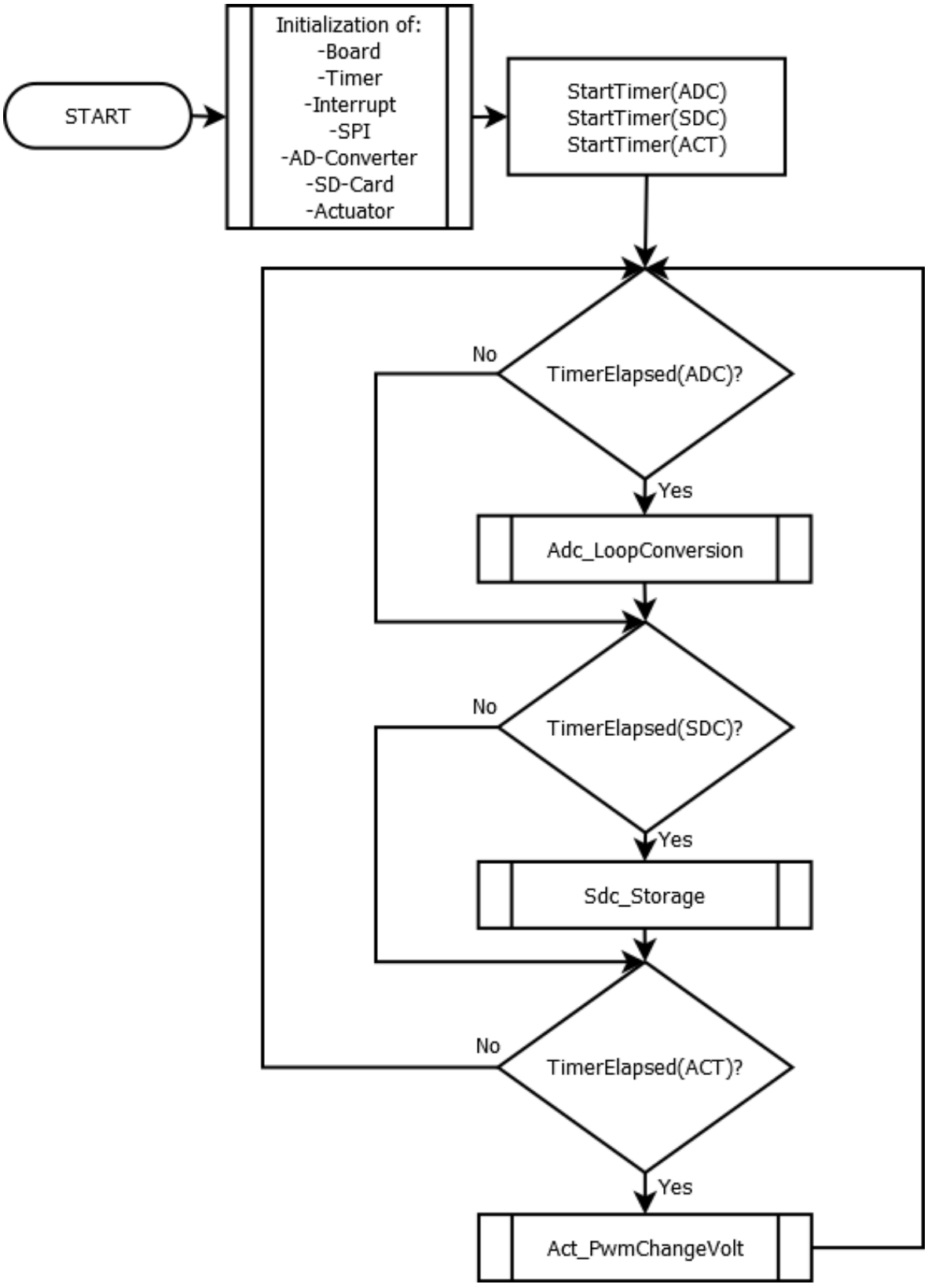

Abb. 3.3.: Flussdiagramm des Main-Programms

3.1. Serial Peripheral Interface

Die folgenden Erklärungen beziehen sich auf das Modul spi_config, welches sich aus den Dateien spi_config.h und spi_config.c zusammen setzt. Die Elemente des Moduls besitzen den Modulindex „Spi“. Als Grundlage dieses Moduls diente die ASF-Bibliothek „SPI - Common Standard SPI“.

Wie in Abschnitt 2.2.2 „Serial Peripheral Interface“ im Kapitel „Entwurf und Implementierung“ angeschnitten, können bei dem verwendeten µC SAM3X8E die Einstellungen für bis zu 4 Slaves in Registern abgespeichert werden. So müssen bei einem Wechsel zwischen Slaves die Konfigurationen nicht von Neuem vorgenommen werden. Die Einstellungen für die Kommunikation werden in der Initialisierungs-Funktion des SPI-Moduls zu Beginn der Main-Funktion aufgerufen (Vgl. Abb. 3.3).

Pinbelegung:

Zur korrekten Funktionsweise des SPI-Controllers, müssen dem µC zuerst die verwendeten Pins mitgeteilt werden. Zudem müssen für die Pins Konfigurationen vorgenommen werden, wie im Kapitel 3 im Absatz „Registerarchitektur für General Purpose Input/Output Pins“ beschrieben. In Tab. 3.4 sind alle für die Kommunikation relevanten Pins aus Sicht des Arduino Due Boards aufgelistet. Zudem sind die Ergebnisse der initialen Registereinstellungen anhand des jeweiligen Statusregisters abgebildet. Tab. 3.7 und Tab. 3.9 zeigen die Verbindung der Pins am ADC bzw. der SD-Karte und den entsprechenden Pins am Arduino Due Board.

Tab. 3.4.: Pinbelegung und initiale Konfiguration der SPI-Peripherie am Arduino Due Board

Arduino Pin	Arduino Name	PIO_PSR	PIO_OSR
10	CS-SDC	0	1
4	CS-ADC	0	1
57	$\overline{\text{DRDY}}$ ADC	1	0
74	MISO	0	0
75	MOSI	0	1
76	SCK	0	1

Initialisierung:

Die Initialisierung wird über die C-Funktion 3.1.1 aufgerufen. Im ersten Teil der Funktion zur Initialisierung im SPI-Modul werden Einstellungen vorgenommen, die alle Slaves betreffen. Diese generellen Registereinstellungen sind in Tab. 3.5 aufgeführt. Der zweite Teil der Initialisierung betrifft die spezifischen Einstellungen des ADC und der SD-Karte, welche in Tab. 3.6 zusammengefasst werden.

C-Funktion 3.1.1: void Spi_Init (void);

Tab. 3.5.: Registereinstellungen für die SPI-Kommunikation

Beschreibung	**Register**	**Wert**
SPI_reset(SPI0)	SPI_MR → SWRST	1
MC im Master-Mode	SPI_MR → MSTR	1
Fixed peripheral select	SPI_MR → PS	0
Peripheral select decode (deaktiviert)	SPI_MR → PCSDEC	0
Mode Fault Detect (deaktiviert)	SPI_MR → MODFDIS	1
Don't Wait Data Read before Transfer	SPI_MR → WDRBT	0
Local Loopback (deaktiviert)	SPI_MR → LLB	0
Delay between Chip Select	SPI_MR → DLYBCS	0
SPI Enable Status (einmalig) set with	SPI_CR → SPIEN	1
SPI Enable Status read with	SPI_SR → SPIENS	1

Tab. 3.6.: SPI Registereinstellungen für die SD-Karte und den ADC

Beschreibung	**Register**	**SD-Karte**	**ADC**
Peripheral Chip Select	SPI_ MR → PCS	0	1
MODE_ 0	SPI_ CSR0 → CPOL	0	0
MODE_ 0	SPI_ CSR0 → NCPHA	0	1
keep CS low after Transfer	SPI_ CSR0 → CSAAT	1	1
8 Bits per Transfer	SPI_ CSR0 → BITS	0	0
Serial ClockBaud Rate - for Init	SPI_ CSR0 → SCBR	210	12
Serial ClockBaud Rate - later	SPI_ CSR0 → SCBR	4	-
Delay before SPCK	SPI_ CSR0 → DLYBS	0	0
Delay before Consecutive Transfers	SPI_ CSR0 → DLYBCT	0	0

Anwendung:

C-Funktion 3.1.2: `void Spi_SelectDevice (uint8_t ChannelId);`

In der Hauptschleife der Main-Funktion wird lediglich die C-Funktion 3.1.2 benötigt. Diese Funktion wird zu Beginn jeder Kommunikation verwendet, sobald der Slave gewechselt wird. Durch Aufruf der Funktion wird die CS-Leitung für den gewählten Slave auf Low gesetzt, wodurch zudem automatisch die in den Registern hinterlegten Einstellungen für diesen Slave verwendet werden.

3.2. Analog-to-Digital-Converter

Die Beschreibungen innerhalb dieses Abschnittes beziehen sich auf die Dateien ad_converter.h und ad_converter.c, welche gemeinsam das Modul ad_converter bilden und dessen Elemente den Modulindex „Adc“ besitzen.

Pinbelegung:

Neben den SPI-Pins, die im vorherigen Abschnitt vorgestellt wurden, sind beim ADC noch weitere Pins für die Umsetzung der Software von Relevanz. Alle notwendigen Pins sind in Tab. 3.7 aufgeführt. Die $\overline{\text{DRDY}}$-Leitung nimmt den Wert Eins an, sobald valide Werte einer Wandlung vorliegen. Die Leitungen AIN0 bis AIN7 sind die Eingangskanäle des ADCs. Dabei bilden AIN0 & AIN1, AIN2 & AIN3 und AIN4 & AIN5 die Differenzeingänge 1 bis 3. Sie sind über die Sub-Miniature-B (SMB)-Buchsen P21, P22 und P23 nach außen geführt. AIN6 und AIN7 sind als Single-ended Eingang an Konnektor P25 und P26 verfügbar. Zudem ist es möglich durch Setzen eines Jumpers an K1 bzw. K2 auf dem Board die Eingänge AIN6 und AIN7 als weiteren Differenzeingang an der SMB-Buchse P24 zu verwenden. In diesem Fall werden die Eingänge über einen Instrumentenverstärker INA101 mit Offset-Abgleich verstärkt und anschließend an den Pin AIN6 am ADC weitergeleitet. Als Referenzspannung dient dem ADC eine 2,5 Volt Referenzspannungsquelle.

Tab. 3.7.: Für die Software relevante Pins des ADC

Pin	Name	Arduino Pin	Arduino Name	Shield Konnektor
23	DIN	75	MOSI	P20
22	DOUT	74	MISO	P20
21	$\overline{\text{DRDY}}$	57	$\overline{\text{DRDY}}$ ADC	/
24	SCLK	76	SCK	P20
20	$\overline{\text{CS}}$	4	CS-ADC	/
6	AIN0	/	/	P21
7	AIN1	/	/	P21
8	AIN2	/	/	P22
9	AIN3	/	/	P22
10	AIN4	/	/	P23
11	AIN5	/	/	P23
12	AIN6	/	/	P24 bzw. P25
13	AIN7	/	/	P24 bzw. P26

Initialisierung:

Der ADC ADS1256 benötigt keine allgemeine Initialisierung. Zu Beginn jeder Wandlung müssen jedoch Einstellungen für jeden zu wandelnden Kanal vorgenommen werden. Diese Einstellungen müssen innerhalb der main-Funktion, außerhalb der while-Schleife einmal bekannt gegeben werden. Dies geschieht über die C-Funktion 3.2.1.

C-Funktion 3.2.1: void Adc_SetChannelParams (uint8_t channel_id, bool loopactive, Adc_gain_t gain, Adc_drate_t datarate);

Mit dem ersten Übergabeparameter der Funktion „channel_id“ wird der Kanal ausgewählt, für den die folgenden Einstellungen gelten sollen. Die Variable loopactive schaltet für den gewählten Kanal die AD-Wandlung mit einer Eins an und mit einer Null aus. Über gain lässt sich der Verstärkungsfaktor einstellen. Hierfür sind nur die Zweierpotenzen von 0 bis 7 zulässig. Die Wahl des Verstärkungsfaktors muss dem µC als hexadezimaler Befehl mitgeteilt werden. Zur besseren Anwendung wurden diese Befehle in dem Dateityp Adc_gain_t durch Ersetzungen in lesbare Hochsprache zusammengefasst. Dasselbe wurde für die Variable datarate vorgenommen. Mit diesem Parameter wird die Anzahl der Wandlungen pro Sekunde festgelegt.

Anwendung:

Der ADS1256 bietet dem Anwender zwei Modi. Im Modus „Fast Channel Cycling“ wird nach dem entsprechenden Befehl ein Kanal mit maximal möglicher Geschwindigkeit zyklisch gewandelt, bis ein entsprechender Stop-Befehl übertragen wird. Der Modus „One Shot Conversion“ wandelt einen Kanal einmalig. Für die Datenerfassungsplattform wurde angenommen, dass in der Regel mehr als ein analoges Signal gemessen wird. Häufig sollen von verschiedenen analogen Eingängen die Werte zum selben Zeitpunkt miteinander verglichen werden. Der ADS1256 ist jedoch nicht fähig Kanäle parallel, sondern lediglich nacheinander zu wandeln. Daher sollten die Kanäle möglichst schnell hintereinander gewandelt werden, damit der Zeitpunkt der Wandlung möglichst dicht beieinander liegt. Das Wechseln der zu wandelnden Kanäle ist mit dem Modus „One Shot Conversion“ am schnellsten möglich. Infolgedessen wurde dieser Modus für die Umsetzung des ADC-Moduls verwendet. Die Kanal-spezifischen Einstellungen, die in der C-Funktion 3.2.1 getroffen werden, werden im Ablauf zwischen dem Schreiben ins MUX-Register und dem Senden des SYNC Befehls vorgenommen.

C-Funktion 3.2.2: void Adc_LoopConversion (void);

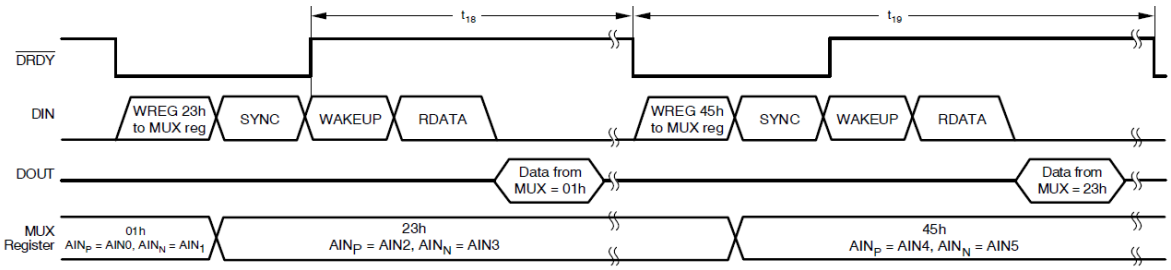

Abb. 3.4.: Befehlssequenz für das Wandeln verschiedener Kanäle nacheinander [18]

Zum Starten einer Wandlung müssen einige Befehle an den ADC gesendet werden. Die Befehle des ADS1256 sind in Tab. 3.8 aufgeführt. In Abb. 3.4 ist exemplarisch eine Befehlssequenz zum Starten und Lesen einer Wandlung dargestellt. Sobald die Leitung $\overline{\text{DRDY}}$ den Wert Null angenommen hat, können Befehle an den ADC gesendet werden. Zunächst muss der zu wandelnde Kanal gewählt werden. In dem Beispiel 3.4 wird der Befehl WREG verwendet, um den hexadezimalen Wert 23 in das Multiplexer-Register MUX zu schreiben. Dadurch wird der Differenzeingang mit dem positiven Pin AIN2 und dem negativen Pin AIN3 für die nächste Wandlung ausgewählt. Anschließend wird der Befehl SYNC gesendet, gefolgt von dem Befehl WAKEUP, um die Wandlung zu starten. Zwischen den Befehlen muss jeweils eine definierte Wartezeit eingehalten werden. Solange gewandelt wird, nimmt der Pin $\overline{\text{DRDY}}$ den Wert Eins an. Während die Wandlung stattfindet, können durch den Befehl RDATA die Ergebnisse der vorherigen Wandlung auf der MISO-Leitung eingelesen werden. Im Beispiel handelte es sich dabei um die Wandlung des Differenzeingangs AIN0 und AIN1. Nach dem Senden des Befehls RDATA stehen die Ergebnisse erst nach einer definierten Verzögerung zur Verfügung (Vgl. Abb. 3.5). Sobald $\overline{\text{DRDY}}$ wieder den Null angenommen hat, kann die nächste Wandlung nach dem selben Schema gestartet werden.

Im Quellcode wird dieser Ablauf durch den Aufruf der C-Funktion 3.2.2 ausgeführt. Hier wird für jeden Kanal abgefragt, ob dieser durch das Flag loopactive in der C-Funktion 3.2.1 aktiviert wurde. Nur aktivierte Kanäle werden gewandelt. Das Ergebnis einer Wandlung setzt sich aus drei Bytes zusammen. Nach jeder Wandlung werden die drei Bytes gelesen, sortiert und in das Array Adc_Counts[ChannelId] zusammengefügt und zwischengespeichert. Der Wert entspricht der Einheit Counts im Binary Format. Unter Berücksichtigung der Referenzspannung, des Verstärkungsfaktors und der Darstellung im Zweierkomplement, wird aus dem zwischengespeicherten Wert in den entsprechenden Spannungswert umgerechnet. Anschließend wird dieser Wert zusammen mit einem Zeitstempel in das Array aus Zeichenketten Adc_StringResult[ChannelId]

geschrieben. Dieses Array wird im Abschnitt 3.3 „Secure-Digital-Memory-Karte“ im Kapitel „Entwurf und Implementierung“ weiter verwendet.

Tab. 3.8.: Definitionen der Befehle des ADS1256 [18]

Befehl	**Beschreibung**	**1. Befehlsbyte**	**2. Befehlsbyte**
WAKEUP	Completes SYNC and exits Standby Mode	0000 0000	
RDATA	Read Data	0000 0001	
RDATAC	Read Data Continuously	0000 0011	
SDATAC	Stop Read Data Continuously	0000 0011	
RREG	Read from REG rrr	0001 rrrr[1]	0000 nnnn[2]
WREG	Write to REG rrr	0101 rrrr	0000 nnnn
SELFCAL	Offset and Gain Self-Calibration	1111 0000	
SELFOCAL	Offset Self-Calibration	1111 0001	
SElFGCAL	Gain Self-Calibration	1111 0010	
SYSOCAL	System Offset Calibration	1111 0011	
SYSGCAL	System Gain Calibration	1111 0100	
SYNC	Synchronize the A/D Conversion	1111 1100	
STANDBY	Begin Standby Mode	1111 1101	
RESET	Reset to Power-Up Values	1111 1110	
WAKEUP	Completes SYNC and exits Standby Mode	1111 1111	

[1] Startregister für Lese-/Schreibbefehle.
[2] Anzahl der zu lesenden/schreibenden Register -1. Z. B. um drei Register zu lesen, setze nnnn=0010.

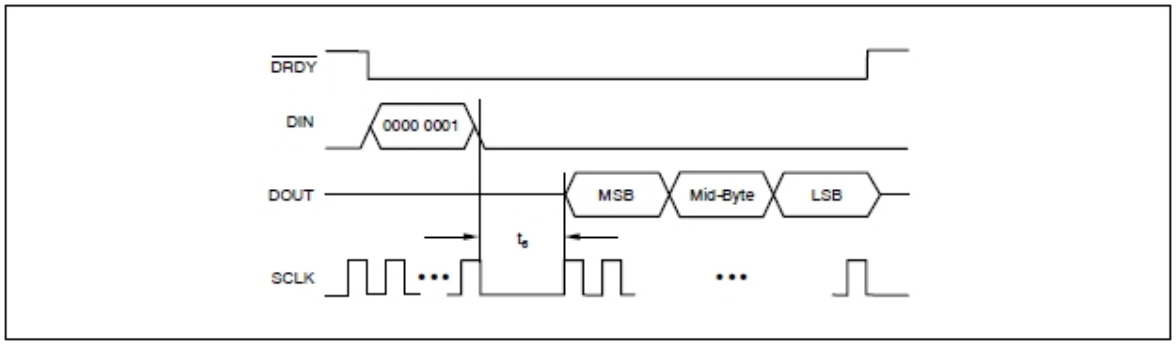

Abb. 3.5.: Befehlssequenz für RDATA [18]

3.3. Secure-Digital-Memory-Karte

Die Beschreibungen innerhalb dieses Abschnittes beziehen sich auf die Dateien sd_card.h und sd_card.c, welche gemeinsam das Modul ad_card bilden und dessen Elemente den Modulindex "´Sdc“ besitzen. Als Grundlage des Moduls dienen die ASF-Bibliotheken „SD/MMC stack on SPI interface“ und „FatFS file system“. Letztere wird im folgenden Abschnitt 3.4 „File-Allocation-Table“ im Kapitel „Entwurf und Implementierung“ näher erläutert.

Pinbelegung:

Über den SD-Karten-Konnektor werden die Pins des Arduino mit den Pins an der SD-Karte verbunden. Abgesehen von den notwendigen vier SPI-Pins wird die Karte nur mit der Versorgungsspannung von 3,3 V und Masse verbunden. In Tab. 3.9 sind alle Pins der SD-Karte und die verbundenen Pins am Arduino aufgelistet.

Tab. 3.9.: Verwendete Pins der SD-Karte

Pin	Name	Beschreibung	Arduino Pin	Arduino Name
1	CS	Chip Select (Active low)	10	CS-SDC
2	DataIn	Master zu Slave Befehle und Daten	75	MOSI
3	VSS2	Masse der Versorgungsspannung	DGND	DGND
4	VDD	Versorgungsspannung	3.3V	3.3V
5	CLK	Clock	76	SCK
6	VSS2	Masse der Versorgungsspannung	DGND	DGND
7	DataOut	Slave zu Master Daten und Status	74	MISO
8	RSV	Reserviert	/	/
9	RSV	Reserviert	/	/

Initialisierung:

Die Funktion 3.3.1 setzt sich aus einigen Funktionen zusammen, die von der ASF-Bibliothek „SD/MMC stack on SPI interface“ bereitgestellt wird. Zuerst wird die Funktion void sd_mmc_init (void) aufgerufen. Hiermit wird der Status der SD-Karte zurückgesetzt und der Gerätetreiber initialisiert.

C-Funktion 3.3.1:

```
void Sdc_Init (void)
{
    sd_mmc_init();
    Sdc_Handler();
}
```

Anschließend wird die Funktion void Sdc_Handler (void) ausgeführt. Innerhalb dieser wird die Funktion sd_mmc_check(uint8_t slot) verwendet. Mit dieser Funktion wird die Verbindung mit der SD-Karte überprüft und sofern diese sichergestellt ist, wird die eigentliche Initialisierung durchgeführt. Dabei wird unter anderem SPI als zu verwendendes Kommunikationsverfahren festgelegt, die spezifischen Daten der SD-Karte werden ausgelesen, die maximale Lese- und Schreibgeschwindigkeit wird erfragt und angewendet und es findet ein Kommunikationstest statt. Nach erfolgreicher Initialisierung wird der Status der SD-Karte auf „SD_MMC_CARD_STATE_READY“ gesetzt.

C-Funktion 3.3.2:

```
void Sdc_Setup (void)
{
    Sdc_MountDisk();
    Sdc_CreateAdcTable(Sdc_filename);
}
```

Die Funktion 3.3.2 ist speziell für die Speicherung der ADC-Werte in einer Tabelle ausgelegt. Mit Sdc_MountDisk(void) wird ein FAT-Laufwerk auf der SD-Karte eingebunden. Mehr dazu im Abschnitt 3.4 „File-Allocation-Table“ im Kapitel „Entwurf und Implementierung“.

Anschließend wird die Funktion Sdc_CreateAdcTable(char table_name[15]) ausgeführt. Hierdurch wird eine Tabelle mit Spaltenüberschriften erstellt, denen später die gewandelten Werte zugeordnet werden. Im Detail werden dazu folgende Schritte ausgeführt: Zuerst wird die zuvor erstellte FAT-Datei geöffnet und es wird mit der Funktion 3.4.4 an das Ende der Datei gesprungen. Anschließend wird eine Zeichenkette gebildet, welche die Spaltenüberschriften enthält. Es soll eine Spalte mit dem Zeitstempel und für jeden aktivierten Kanal eine weitere Spalte erstellt werden. Dazu werden mit Hilfe des Flags Adc_loop_active, welches mit der Funktion 3.2.1 gesetzt wurde, die Kanäle ermittelt, welche tatsächlich gewandelt werden sollen. Die Zeichenkette mit den Spaltenüberschriften wird nun in das File geschrieben. Zuletzt wird die FAT-Datei wieder geschlossen.

Anwendung:

Die Funktion 3.3.3 wird nach Ablauf des Timers für die SD-Karte innerhalb der Endlosschleife aufgerufen. Hierüber werden alle gemessen ADC-Werte in die zuvor erstellte Tabelle in die Spalte des jeweiligen ADC-Kanals geschrieben.

C-Funktion 3.3.3: `void Sdc_Storage (char table_name[15]);`

3.4. File-Allocation-Table

Die Beschreibungen innerhalb dieses Abschnittes beziehen sich auf die ASF-Bibliothek „FatFS file system“. Die Funktionen der Bibliothek werden von den Funktionen des oben beschriebenen Moduls sd_card verwendet. Geschrieben wurde die Bibliothek von dem Japaner Elm Chan als quelloffene Software für Bildung, Forschung und Entwicklung [4].

Bibliothek:

C-Funktion 3.4.1:

```
DSTATUS disk_initialize (BYTE drv);
DSTATUS disk_status (BYTE drv);
DSTATUS disk_read (BYTE drv, BYTE *buff, DWORD sector, BYTE count);
DSTATUS disk_write (BYTE drv, BYTE const *buff, DWORD *sector, BYTE count);
DSTATUS disk_ioctl (BYTE drv, BYTE ctrl, void *buff);
```

Wie in Abb. 3.6 in türkis dargestellt, setzt sich die Bibliothek aus fünf Dateien zusammen. Vom Hauptprogramm aus (Application) werden ausschließlich Funktionen aus der Datei ff.c verwendet. Die Datei bietet unter anderem Funktionalitäten um ein FAT-Laufwerk zu erstellen, zu öffnen, zu schließen, in ihm zu schreiben und zu lesen. Die Dateien ffconf.h, ff.h und integer.h sind für den Anwender nicht von Bedeutung. Die Bibliothek wurde als allgemeine Dateisystem-Ebene entwickelt und ist somit nicht für eine bestimmte Hardware spezialisiert. Um die FAT-Bibliothek für die genutzte Hardware einzurichten, muss zuerst die Datei diskio.h angepasst werden. Hier müssen die Funktionen aus 3.4.1 mit den Treiber-Funktionen für die SD-Karte gefüllt werden. Diese Funktionen aus der Datei diskio.h dienen als Grundlage für die Funktionen in der Datei ff.c, wozu unter anderem die unten beschriebenen Funktionen 3.4.2 bis 3.4.6 zählen.

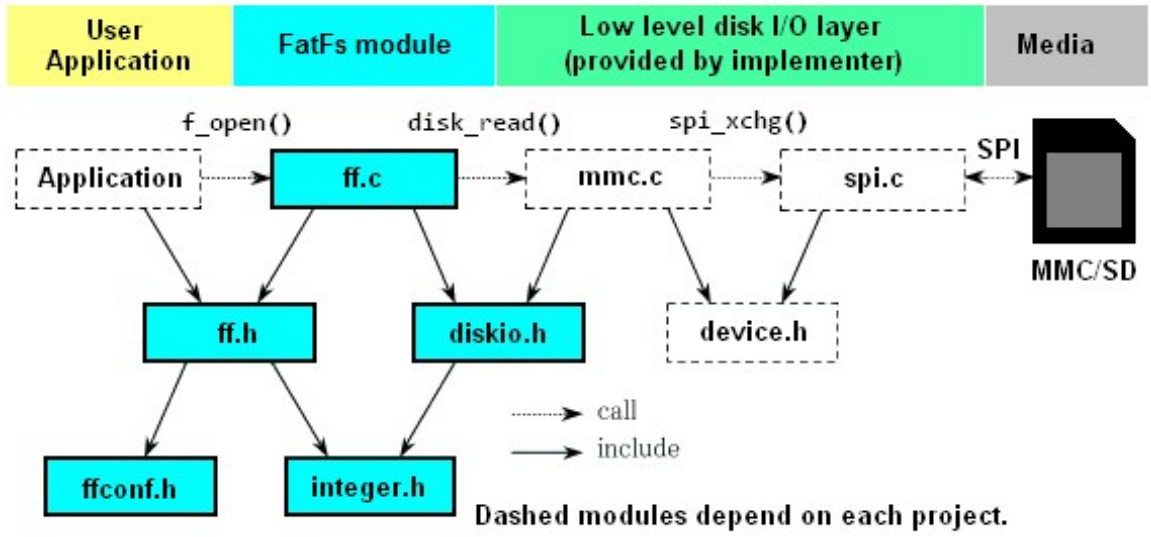

Abb. 3.6.: Aufbau der FAT-Bibliothek [5]

Initialisierung:

Wie im Abschnitt 3.3 „Secure-Digital-Memory-Karte“ im Kapitel „Entwurf und Implementierung“ beschrieben, wird die Funktion 3.4.2 von der Funktion 3.3.2 Sdc_Setup (void) verwendet. Hiermit wird geprüft, ob die SD-Karte als Laufwerk bereits existiert. Andernfalls wird sie als neues Laufwerk eingebunden und ein File-System auf ihr implementiert.

C-Funktion 3.4.2: FRESULT f_mount (BYTE vol, FATFS *fs);

Anwendung:

Beim Aufruf der Funktion 3.4.3 wird eine FAT-Datei erstellt oder geöffnet. Mit dem Zeiger fp wird das Dateiobjekt ausgewählt. Der Zeiger path zeigt auf die Zeichenkette, die den Namen der Datei enthält. Um den Modus der Datei festzulegen, kann für den letzte Parameter mode zwischen den folgenden Konstanten gewählt werden:
FA_READ
FA_OPEN_EXISTING
FA_WRITE
FA_CREATE_NEW
FA_CREATE_ALWAYS
FA_OPEN_ALWAYS
Dabei ist auch eine Kombination aus mehreren Konstanten möglich. Für dieses Projekt wurden die Modi FA_OPEN_ALWAYS und FA_WRITE verwendet. Durch die

Konstante FA_OPEN_ALWAYS wird eine Datei mit dem übergebenen Namen immer dann erstellt, wenn noch keine Datei mit demselben Namen existiert. Andernfalls wird die bereits bestehende Datei geöffnet. Mit FA_WRITE wird Schreibzugriff auf die Datei gewährleistet. Um die Funktionen 3.4.4, 3.4.5 und 3.4.6 nutzen zu können, müssen zuvor die Funktionen 3.4.2 und 3.4.3 für dasselbe FAT-System und dieselbe Datei mindestens einmal erfolgreich ausgeführt worden sein.

C-Funktion 3.4.3: FRESULT f_open (FIL *fp, const TCHAR *path, BYTE) mode);

Die Funktion 3.4.4 ermöglicht das Verschieben des Schreib- und Lese-Zeigers innerhalb der FAT-Datei. Diese Funktion wird bei jedem Aufruf von 3.3.3 verwendet, um den Zeiger in die nächste Zeile der Tabelle zu setzen. Um die Funktion nutzen zu können, muss zuvor einmal die Funktion 3.4.2 erfolgreich ausgeführt worden sein.

C-Funktion 3.4.4: FRESULT f_lseek (FIL *fp, DWORD ofs);

Mit der Funktion 3.4.5 wird eine Zeichenkette in eine zuvor geöffnete Datei geschrieben. Der Zeiger str zeigt dabei auf die Zeichenkette, die in die Datei geschrieben werden soll. Das Schreiben geschieht immer im Anschluss an das zuletzt geschriebene Bit, es sei denn der Schreib- und Lesezeiger wurde mit der Funktion 3.4.4 verschoben.

C-Funktion 3.4.5: FRESULT f_puts (const TCHAR* str, FIL* fil);

Mit der Funktion 3.4.6 wird ein File wieder geschlossen, nachdem es geöffnet wurde. Dies sollte möglichst immer geschehen, sobald nicht mehr geschrieben oder gelesen wird, um Dateiverluste zu vermeiden.

C-Funktion 3.4.6: FRESULT f_close (FIL *fp);

3.5. Pulsweitenmodulation

Pinbelegung:

Auf dem Mikrocontroller SAM3X8E befindet sich ein Controller zur Steuerung von insgesamt 8 PWM-Ausgängen. Vier der acht Pins werden zur Ansteuerung von Aktoren auf dem 24-Bit-Shield nach außen geführt (Vgl. Tab. 3.10). Die Ausgänge können einen maximalen Strom von 15 mA bereitstellen. Um auch größere Lasten betreiben zu können, wurden vor die Ausgangspins auf dem 24-Bit-Shield Treiberstufen des Typs IRF7401 verbaut. Hierdurch ist es möglich, einen Dauerstrom von bis zu 3 A zu liefern [9].

Tab. 3.10.: Für die Software relevante Pins der PWM

Arduino Pin	Arduino Name	µC Pin	Shield Konnektor
6	PWM6	PC24	P13
7	PWM	PC23	P14
8	PWM	PC22	P15
9	PWM	PC21	P16

Initialisierung:

Die Initialisierung des PWM-Controllers wurde mit der Funktion 3.5.1 realisiert. Dabei müssen zunächst die PWM-Clock und alle zu benutzenden PWM-Kanäle deaktiviert werden um Einstellungen vornehmen zu können. Für die Pulsweitenmodulation stehen zwei Uhren zur Verfügung: clka und clkb. Der Controller wird mit dem Wert initialisiert, der über die Variable frequency eingelesen wird. Es wird clka genutzt und wie im Kapitel 2.2.6 „Pulsweitenmodulation“ erläutert, wird hierfür eine Frequenz von 20 kHz eingestellt. Als nächstes werden die vier genutzten PWM-Kanäle konfiguriert. Für alle Kanäle werden die folgenden Einstellungen getroffen:

alignment = PWM_ALIGN_LEFT
: Das Rechtecksignal soll am linken Rand der Periode ausgerichtet sein.

polarity = PWM_LOW
: Der Grundzustand des Signals ist low.

prescaler = PWM_CMR_CPRE_CLKA

Der Vorteiler teilt die Frequenz vom Quarz soweit runter, bis sie der eingestellten Frequenz entsprcht.

period = Act_PERIOD_VALUE

Die Periode wird auf 100

duty = Act_INIT_DUTY_VALUE

Als Tastverhältnis wird initial 50

Zum Schluss der Funktion werden alle Kanäle mit den neuen Einstellungen wieder aktiviert.

C-Funktion 3.5.1: void Act_Init (uint32_t frequency);

Anwendung:

Mit der Funktion 3.5.2 lässt sich die Ausgangsspannung der PWM-Pins einzeln steuern. Über die Variable channel lässt sich der anzupassende Kanal aus einer Liste des Datentyps Act_channel_t auswählen. Mit dem Parameter volt lassen sich Spannungen in der Einheit Volt übergeben. Aus der Spannung wird mit der Formel 2.2 das Tastverhältnis berechnet, welches für den gewählten Kanal eingestellt wird. Diese Funktion wird in der Endlosschleife der Main-Funktion 3.0.1 aufgerufen. Der Aufruf geschieht, sobald der Timer für das Modul Actor abgelaufen ist (Vgl. Abb. 3.3).

C-Funktion 3.5.2: void Act_PwmChangeVolt (Act_channel_t channel, float volt);

4. Ergebnisse

Ziel dieser Arbeit war es, die Software für eine Datenerfassungsplattform zu entwickeln, mit der analoge, biomedizinische Sensorsignale aufgenommen und portabel auf einer SD-Karte gespeichert werden können. Außerdem sollten sich mit dem Gerät Ausgänge ansteuern lassen, um optional einen Regelkreis zu implementieren. Damit soll eine kostengünstige und portable Alternative für einige Anwendungsbereiche des dSPACE Systems geschaffen werden.

Um abwägen zu können, ob das Ziel dieser Arbeit erreicht wurde, sollten die auf dem Arduino Due Board und dessen Peripherie implementierten Funktionalitäten verifiziert werden. Dazu wurde mit dem Signalgenerator eine definierte Spannung von 20 mV auf den Eingang CH1 des, ADCs gegeben. Die Kanäle CH4 und CH5 wurden ebenfalls aktiviert, sie wurden jedoch mit keiner Spannung verbunden. Damit sollte die korrekte Formatierung der Tabelle überprüft werden. Die gewandelten Werte über eine festgesetzte Zeitdauer wurden auf der SD-Karte gespeichert. In Abb. 4.1 ist die resultierende Tabelle auf der SD-Karte abgebildet. Zeitgleich wurde die Eingangsspannung mit Zehn multipliziert und über einen PWM-Ausgang ausgeben. Am Ausgang wurde das Ergebnis mit einem Multimeter gemessen.

Timestamp CH1	Volts CH1	Timestamp CH4	Volts CH4	Timestamp CH5	Volts CH5
00:00:02	0.019530	00:00:02	-0.000017	00:00:02	-0.000005
00:00:03	0.019532	00:00:03	-0.000004	00:00:03	-0.000003
00:00:03	0.019531	00:00:03	-0.000007	00:00:03	-0.000005
00:00:04	0.019531	00:00:04	-0.000009	00:00:04	-0.000003
00:00:04	0.019297	00:00:04	0.000001	00:00:04	-0.000004
00:00:04	0.019297	00:00:04	0.000002	00:00:04	-0.000000
00:00:05	0.019592	00:00:05	0.000009	00:00:05	-0.000003
00:00:05	0.019586	00:00:05	0.000022	00:00:05	-0.000002
00:00:06	0.019531	00:00:06	0.000041	00:00:06	-0.000004
00:00:06	0.020297	00:00:06	0.000010	00:00:06	-0.000000
00:00:06	0.020424	00:00:06	0.000018	00:00:06	-0.000000
00:00:07	0.020297	00:00:07	0.000011	00:00:07	-0.000000
00:00:07	0.019694	00:00:07	0.000014	00:00:07	-0.000006
00:00:08	0.019674	00:00:08	0.000009	00:00:08	-0.000005
00:00:08	0.019531	00:00:08	0.000006	00:00:08	-0.000003
00:00:09	0.019063	00:00:09	-0.000000	00:00:09	-0.000001
00:00:09	0.019297	00:00:09	-0.000018	00:00:09	-0.000002
00:00:09	0.019531	00:00:09	-0.000010	00:00:09	-0.000006
00:00:10	0.019531	00:00:10	-0.000010	00:00:10	-0.000004

Abb. 4.1.: Beispiel einer Tabelle auf der SD-Karte

Die Abb. 4.1 zeigt, dass die aufgenommenen Werte für den Kanal CH1 sehr nahe der zu augebrachten Spannung liegt. Die Abweichungen von maximal 0,939 mV liegen vermutlich in der Ungenauigkeit des Spannungsgenerators begründet. Das Format der erzeugten Tabelle entspricht den Vorgaben. Es werden nur Spalten für die aktivierten Kanäle erzeugt.

Am PWM-Ausgang wird eine mittlere Spannung von ca. 200 mV gemessen. Mit einem Oszilloskop konnte die Frequenz des PWM-Signals auf 20,016 kHz bestimmt werden. Damit liegt die Frequenz um 16 Hz höher als angenommen. Wie dem Abschnitt 2.2.6 „Pulsweitenmodulation“ im Kapitel „Grundlagen“ beschrieben, hat eine Abweichung der PWM-Frequenz in dieser Größenordnung jedoch keinen Einfluss auf die zu erzeugende mittlere Spannung.

Damit sind die Anforderungen der Messung von analogen Signalen, der Speicherung der Daten auf der SD-Karte und die Erzeugung einer Ausgangsspannung erfüllt. Zudem wurde durch die Messungen gezeigt, dass das Board unabhängig von einem Computer eingesetzt werden kann, wodurch die Anforderung nach Portabilität und Flexibilität erreicht wird.

5. Diskussion und Ausblick

Alle im Abschnitt 1.1 „Problemstellung“ im Kapitel „Einleitung“ genannten Anforderungen für die Verwendung der Datenerfassungsplattform in forschungsnahen Evaluationen und in der Softwareentwicklung wurden im Rahmen dieser Arbeit erfüllt. Über den 24-Bit-ADC können Signale hochauflösend aufgenommen und verarbeitet werden. Die Werte werden in einer Tabelle auf der SD-Karte gespeichert. Zudem lassen sich Aktoren über die vier PWM-Ausgänge ansteuern. Aus der Kombination der gemessen Eingangssignale und der PWM-Ausgänge, lassen sich Regelkreise mit dem Board realisieren. Durch seine geringen Maße ist das Board flexibel einsetzbar.

Soll jedoch das Programm auf dem Arduino Due Board geändert werden, wie beispielsweise um andere ADC-Kanäle auszuwählen, den Verstärkungsfaktor zu ändern oder einen Regelkreis zu implementieren, muss stets der Quellcode geändert werden. Die Änderung des Quellcodes setzt Programmierkenntnisse und Zeit zum Einlesen voraus. Dieses Vorgehen ist nicht praktikabel, da der Hauptnutzen der Datenerfassungsplattform die schnelle und unkomplizierte Erhebung von Messwerten sein und das Board universell von jedem Anwender eingesetzt werden soll.

Um das Board benutzerfreundlicher zu gestalten, kann eine grafische Benutzeroberfläche eine mögliche sinnvolle Erweiterung darstellen, um das Boards je nach Anwendungsfall schnell und unkompliziert programmieren zu können. Hierzu würde sich beispielsweise die Software MATLAB mit dem Zusatzprodukt Simulink von dem Hersteller The Mathworks eignen. Simulink ist ein Programm zur Modellierung von Systemen mittels grafischer Benutzeroberfläche. Der Vorteil der Software liegt darin, dass sich die Funktionalität der Erstellung komplexer Regelkreise und mathematischer Algorithmen einbinden lässt, um daraus einen Programmcode zu erstellen und auf den Mikrocontroller zu übertragen. Für Simulink wird vom Hersteller The Mathworks ein Arduino Support Package bereitgestellt. Hiermit soll es möglich sein, in einem Schritt Modelle aus Simulink in C-Code zu wandeln, zu kompilieren und auf das Arduino Due Board zu übertragen. Zudem werden durch das Zusatzpaket einige Blöcke zur allgemeinen Ansteuerung des Arduino bereitgestellt. Dazu zählen Blöcke zur Kommunikation und

analoge und digitale Ein- und Ausgängsblöcke. Um die Hardware des 24-Bit-Shields anzusteuern, müssen jedoch eigene Blöcke erstellt werden. Mit dem „Legacy Code Tool“ oder alternativ mit dem „S-Function Builder“ lässt sich bereits bestehender C-Code (Legacy Code) einbinden, um daraus sogenannte S-Functions zu erstellen. Eine S-Function wird in Simulink als ein Block dargestellt, der nach Belieben parametriert und mit anderen Blöcken verbunden werden kann. So ließen sich beispielsweise der ADC, die SD-Karte und die PWM-Ausgänge jeweils durch einen Simulink-Block darstellen.

Bisherige Untersuchungen haben jedoch ergeben, dass es zu Komplikationen kommt, wenn bei der Erstellung der S-Functions Legacy Code verwendet wurde, der nicht die Bibliotheken der Arduino IDE, sondern wie in unserem Fall die Atmel Bibliotheken verwendet. Nach Angaben von The Mathworks lässt sich dieser Legacy Code nur verwenden, indem ein sogenanntes Custom-Target entwickelt und verwendet würde. Dieses würde das Arduino Support Package ersetzen. Die Entwicklung solch eines Targets ist anspruchsvoll und zeitaufwendig, weswegen diese Tätigkeit nicht mehr Teil der vorliegenden Arbeit ist.

Eine weitere mögliche Ergänzung kann die Erweiterung der Hardware durch Komponenten darstellen. Zum einen könnte ein Rotary Encoder zur Einstellung von Sollwerten für einen Regelkreis genutzt werden. Zum anderen wäre es möglich durch einen zusätzlichen Taster die Messung oder das gesamte Programm zu starten und zu stoppen. Dies kann nützlich sein, wenn die Plattform unabhängig von einem PC betrieben wird, da andernfalls die Messung nur durch Unterbrechung der Betriebsspannung gestoppt werden könnte. Über die auf dem Board bereits integrierte LED ließe sich zusätzlich der aktuelle Betriebsmodus durch verschiedene Leuchtfrequenzen oder Helligkeiten darstellen.

Literaturverzeichnis

[1] Arduino LLC: *Arduino Due*. URL:www.arduino.cc/en/pmwiki.php?n=Main/ArduinoBoardDue, 2015. – eingesehen am 07.12.2015

[2] Atmel Corp.: *SAM3X/SAM3A Series Complete*. Rev. C, March 2015. http://www.atmel.com/Images/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf

[3] Bhat, W. A. und Quadri S.: Review of FAT Data Structure of FAT32 file system. In: *Oriental Journal of Computer Science & Technology* 3 (2010), Nr. 1

[4] Chan, E.: *FatFs - Generic FAT File System Module*. http://elm-chan.org/fsw/ff/00index_e.html, November 2015. – eingesehen am 11.12.2015

[5] Chan. E.: *Modules*. http://elm-chan.org/fsw/ff/img/modules.png, 2015. – eingesehen am 25.11.2015

[6] Dumont, A.: Alles über SD-Karten. In: *com-magazin* (2012), November, S. 84–87

[7] Frings, S. und Müller, F.: *Biologie der Sinne: Vom Molekül zur Wahrnehmung*. Springer-Verlag, 2013

[8] International Rectifier: *IRF7401PbF HEXFET Power MOSFET*. 8/10/04, Oktober 2004

[9] Jäger, A.: *Entwurf einer kompakten Steuereinheit zur Volumenstromregelung unter Verwendung von Proportionalventilen*, Fachhochschule Lübeck, Bachelorthesis, 2014

[10] Jäger, A.: *KiCad E.D.A.: Alcohol Test*. 0.1, September 2014

[11] Kiencke, U. und Eger, R.: *Messtechnik*. Springer, 2007

[12] Molex Inc.: *SD MEMORY CARD CONN. ASS'Y (TOP MOUNT TYPE) LEAD-FREE*. S2012-0634, März 2012

[13] SanDisk Corp.: *SanDisk Secure Digital Card*. Version 1.9, Dezember 2003

[14] Schreier, R. und Temes, G.: *Understanding delta-sigma data converters*. Bd. 74. IEEE press Piscataway, NJ, 2005

[15] Texas Instruments Inc.: *High Accuracy INSTRUMENTATION AMPLIFIER INA101*, 1998

[16] Texas Instruments Inc.: *High-Speed, Single-Supply, Rail-to-Rail OPERATIONAL AMPLIFIERS MicroAmplifier Series*. SBOS099A, Juli 2001

[17] Texas Instruments Inc.: *Low-Noise, Very Low Drift, Precision VOLTAGE REFERENCE*. SBOS410, Juni 2007

[18] Texas Instruments Inc.: *Very Low Noise, 24-Bit Analog-to-Digital Converter*. Rev. K, September 2013. `http://www.ti.com/lit/ds/symlink/ads1256.pdf`

[19] Tietze, U. und Schenk, C.: *Halbleiterschaltungstechnik*. Bd. 12. Springer, 1993

A. Anhang

Abb. A.1.: Schaltplan des 24-Bit-Shields [10]

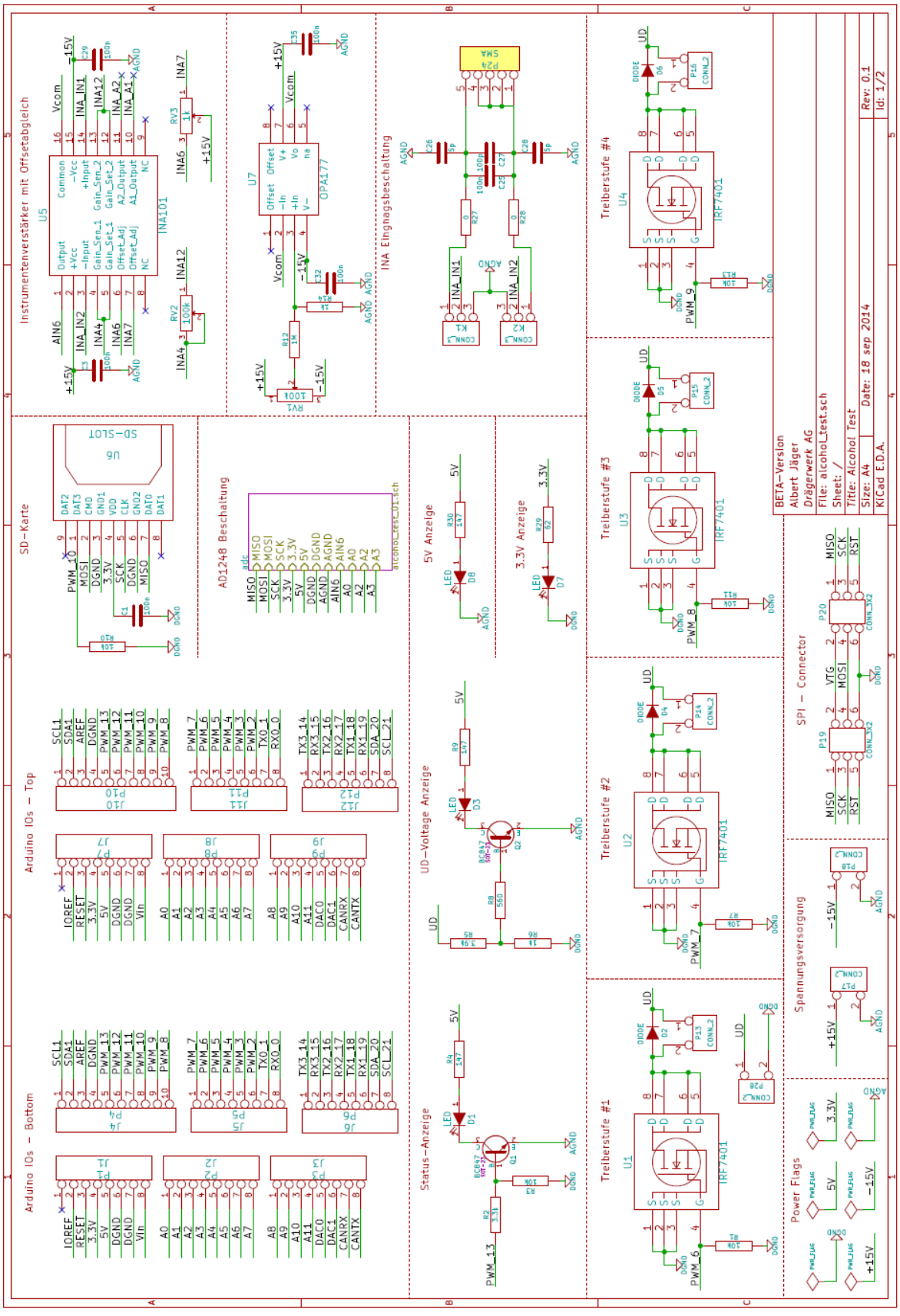

Abb. A.2.: Schaltplan des 24-Bit-Shields [10]