

Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Bachelorarbeit Nr. 280

Entwicklung einer Schrittmotorsteuerung zur Regelung eines inversen Pendels

Benjamin Prölb

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. rer. nat. Dr. h.c. Kurt Rothermel
Betreuer/in:	Dipl.-Ing. Ben Carabelli
Beginn am:	November 16, 2015
Beendet am:	Mai 17, 2016
CR-Nummer:	C.2.4, C.3

Kurzfassung

Bei dem inversen Pendel handelt es sich um ein Problem aus der Regelungstechnik. Dabei müssen die Steuerdaten dem Regler zeitnah zur Regulierung des Regelkreises vorliegen.

Wenn ein System die Anforderungen zur Regulierung des Prozesses nicht erfüllt, muss die Schwachstelle im System ermittelt werden. Bei einem Regelkreis müssen die einzelnen Komponenten auf die Erfüllung der gestellten Anforderungen geprüft werden.

In diesem Sinne wird in der vorliegenden Arbeit ein bereits existierender Aufbau eines inversen Pendels genauer analysiert. Dabei wird der benutzte Aufbau überprüft und auf seine Tauglichkeit geprüft. Es werden verschiedene Ansätze betrachtet, um dabei die Geschwindigkeit der Implementierung zu erhöhen. Eine direkte Programmierung der Hardware ist dabei von besonderer Bedeutung.

Abstract

The inverted pendulum is a classic problem in control theory. It requires the control feedback to be instantly available to the controller in order to regulate the control loop.

If a system does not meet the constraints for the regulation of a process, the system's bottleneck has to be found. In a control loop each component has to be checked for meeting the constraints.

Therefore, in this thesis an already existing setup of an inverted pendulum is analysed in detail. Different approaches to improve the performance of the implementation are discussed. In that context, bare-metal programming is especially important.

Inhaltsverzeichnis

Liste genutzter Abkürzungen	5
Liste aller Abbildungen	6
1. Einführung	7
1.1. Motivation	7
1.2. Ziel dieser Arbeit	7
1.3. Überblick	8
2. Grundlagen: Regelungstechnik	9
2.1. PID-Regler	10
3. Vorgegangene Arbeiten	11
3.1. Versuchsaufbau	11
3.1.1. Überblick	11
3.1.2. Funktionsweise	12
3.2. Hardware	12
3.2.1. Arduino Due	12
3.2.2. Kamera: Playstation EYE	13
3.2.3. A4988 Schrittmotortreiber	13
3.3. Software	14
3.3.1. Anforderungen an die Kamera	14
3.3.2. Erkennung des inverses Pendel	14
4. Programmierung	15
4.1. Arduino IDE	15
4.1.1. Grundlegende Bedienung	15
4.1.2. Eigenschaften der Arduino IDE	16
4.1.3. Probleme mit der Arduino IDE	16
4.2. Atmel Software Framework	17
4.2.1. Grundlagen	17
4.2.2. Makefile für Atmel Software Framework	18
4.2.3. Chip beschreiben	20
4.2.4. LED-Blinker Implementierung	20
4.2.5. UART Implementierung	22
4.2.6. Timer Counter Implementierung	23
4.2.7. Ansteuerung eines Schrittmotors	25
5. Schlussfolgerung	27
5.1. Zeitersparnis	27
5.2. Implementierungsprobleme	27
5.3. Weiterführende Arbeit	28
5.3.1. JTAG Hardware	28
5.3.2. Raspberry Pi Kamera Modul	28
5.3.3. Steuerung über einen Raspberry Pi 2/3	29

6. Zusammenfassung	30
Literatur	31
Anhang	33
A. Makefile	33
B. LED Blinker Test Source Code	36
C. UART Test Source Code	38

Liste genutzter Abkürzungen

ASF	Atmel Software Framework
CAN	Controller Area Network
CPU	Central processing unit
DAC	Digital-to-analog converter
DMAC	Direct Memory Access Controller
FPS	Frames per Second
GCC	GNU Compiler Collection
GUI	Graphical User Interface
I2C	Inter-Integrated Circuit
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LED	Light Emitting Diode
MCK	Master Clock
NVIC	Nested Vector Interrupt Controller
PDC	Peripheral Direct Memory Controller
PIO	Parallel Input/Output Controller
PMC	Power Management Controller
PS EYE	Playstation Eye
SRAM	Static random-access memory
TC	Timer Counter
TWI	Two Wire Interface
UART	Universal Asynchronous Receiver Transceiver
USB	Universal Serial Bus
V4L	Video4Linux
WDT	Watchdog Timer

Abbildungsverzeichnis

1.	Regelungskreis	9
2.	Versuchaufbau inverses Pendel	11
3.	Arduino Due. Bildquelle: [LLC15a]	12
4.	Ausschnitt: Datenblatt Seite 31	23

1. Einführung

Mikrocontroller sind heutzutage in fast allen Geräten vorhanden, ob im Auto, Handy, Uhren, Wasserkocher oder sogar im Kugelschreiber mit eingebauter HD Kamera [CM3]. Wir benutzen ständig so viele Mikrocontroller in unserem Alltag, dass sie in unserem Leben nicht mehr wegzudenken sind. Dank dieser Technik wird unser Leben in den meisten Bereichen enorm vereinfacht.

1.1. Motivation

Jeder der sich für Mikrocontroller interessiert und diese selbst programmiert, muss kaum noch Datenblätter oder Definitionen lesen. Die Tools zum Programmieren von Mikrocontroller haben sich in den letzten Jahren sehr benutzerfreundlich gestaltet. Heutzutage gibt es meistens ein vordefiniertes Framework, in dem Funktionen hinterlegt sind, mit denen der Programmcode im Baukastenprinzip zusammen gestellt werden kann. Die Software wird in den meisten Fällen auf der Internetseite des Herstellers zum Download angeboten. Nach Installation der Software wird mithilfe eines USB-Kabels als Schnittstelle eine Verbindung zum Programmieren und Kommunizieren mit der Hardware hergestellt. Mit grundlegenden Programmierkenntnissen sind Testprogramme schnell entwickelt und können auf ihre Funktion hin geprüft werden. Für einfache Programme, die keine zeitkritischen Anforderungen haben, ist diese Art der Programmierung meist ausreichen. Wird ein Demonstrator¹, wie das inverse Pendel, eingesetzt, dann genügt die Implementierung durch ein gegebenes Framework eventuell nicht mehr, beispielsweise falls die Schnittstellen für benötigte Komponenten nicht bereitgestellt werden. Um exakt zu wissen, welche Anweisungen der Mikrocontroller gerade ausführt, ist eine fundierte Kenntnis vom Hardwareaufbau nötig. Es sollten genaue Vorstellungen von der Kausalität des Programmablaufs vorhanden sein, sowie ein gutes Wissen über die benötigte Zeit zur Ausführung von Befehlen auf dem Prozessor.

1.2. Ziel dieser Arbeit

In der vorangegangenen Arbeit [Nä15] wurde ein Demonstrator, in diesem Fall ein inverses Pendel, erstellt. Das inverse Pendel wurde gewählt, da es ein sehr instabiles System darstellt. Das bedeutet, dass der Vorgang zum Regulieren des Pendels einen zeitkritischen Faktor hat. Deshalb benötigt der Prozess zum Regeln des inversen Pendels immer aktuelle Daten über die Position in der sich das Pendel momentan befindet. Die Position beinhaltet in diesem Fall die Stelle des Schlittens auf einer beschränkten Strecke, sowie den Winkel vom Drehpunkt des Pendels relativ zu einer Grundstellung. Installiert ist das inverse Pendel auf einem ausgemusterten Druckerlaufband. Das hat den Vorteil, dass es durch einen Schrittmotor gesteuert wird und somit exakt die Position des Schlittens ermittelt werden kann. Durch eine Kamera wird der Neigungswinkel des inversen Pendels bestimmt. Dabei wird der Winkel des inversen Pendels zu der gewünschten rechtwinkligen Stellung bezüglich des Druckerlaufbands ermittelt.

¹Ein Demonstrator ist ein Vorführer bzw. Beweisführer. In diesem Fall ist es ein Vorführgerät um eine Funktionalität zu demonstrieren.

Das inverse Pendel ist nicht regelbar, da es kurze Zeit nach dem Start aus seiner Gundstellung umfällt und als normales Pendel agiert. Dies ist aber nicht gewünscht. Das Ziel der aktuellen Arbeit ist es, die möglichen Schwachstellen des aktuellen Systems zu finden und, falls möglich, mit vorhandener Hardware zu beheben. Der Fokus liegt dabei auf der Ansteuerung des Schrittmotors, um den Schlitten auf dem das Pendel justiert ist, kontinuierlich steuern zu können. Eine hardwarenahe (bare-metal) Programmierung des Chips soll dabei verwendet werden. Als Ergebnis soll ein lauffähiger Demonstrator erstellt werden, der für weitere Forschungsarbeiten im Bereich der Regelungstechnik und Vernetzung dienen soll. Um dieses Ziel zu erreichen wurden verschiedene Ansätze zur Programmierung der Hardware untersucht.

1.3. Überblick

In Abschnitt 2 werden die grundlegende Steuerung eines inversen Pendels und gängige Regelungstechniken näher erläutert. Da diese Arbeit auf einer vorangegangenen Arbeit aufbaut, wird in Abschnitt 3 eine Übersicht über das bereits vorhandene Equipment und die Software gegeben. Abschnitt 4 behandelt die Programmierung eines Arduino Due. Anschließend werden in Kapitel Abschnitt 5 die erarbeiteten Ergebnisse erfasst. Eine Zusammenfassung erfolgt in Abschnitt 6.

2. Grundlagen: Regelungstechnik

In der Regelungstechnik gibt es verschiedene Arten, einen dynamischen Prozess zu steuern. Es kommt dabei immer auf das System an, welches man regeln möchte. Im Folgenden werde ich das grundlegende System eines Regelkreises vorstellen, sowie auf verschiedene Methoden eingehen, diesen zu regulieren.

Eine Regelung findet immer Anwendung in Bereichen bei denen eine Abweichung zwischen Soll- und Ist-Wert besteht (siehe Abbildung 1). Es wird durch eine Eingangsgröße ein Soll-Wert vorgegeben, gefolgt von einem Prozess wird eine Ausgangsgröße ausgegeben. Die Ausgangsgröße soll den angestrebten Soll-Wert erfüllen. Auf den Prozess wirken nicht bestimmbar Störgrößen ein, die eine Abweichung der Ausgangsgröße zur Folge haben. Die Korrektur erfolgt durch einen Regelkreis, in dem von der Ausgangsgröße eine Rückführung zur Eingangsgröße erfolgt. Die Rückführung enthält den Ist-Wert, der über das Messglied bzw. die Kamera bestimmt wird. Der Ist-Wert wird mit der Eingangsgröße zur Regelabweichung bestimmt. Mit einem bestimmten Regler wird eine Korrektur bestimmt, die die Regelabweichung ausgleicht. Der Regelungsparameter wird an den Stellmotor weitergegeben und somit die Regelstrecke bzw. der Prozess korrigiert. Es wird durch die Regelung ein geschlossener Kreislauf gebildet in dem eine ständige Übertragung vom Ausgang zum Eingang stattfindet. Regelkreise sind veränderbar durch Eingriffsparameter wie deren Proportionalanteil, ihr Integral oder den Differenzial, sowie Verstärkungsfaktoren oder die Abtastrate des Ausgangssignals. Allerdings ist eine Regelung immer ein Kompromiss aus Dynamik und Stabilität [Lun14].

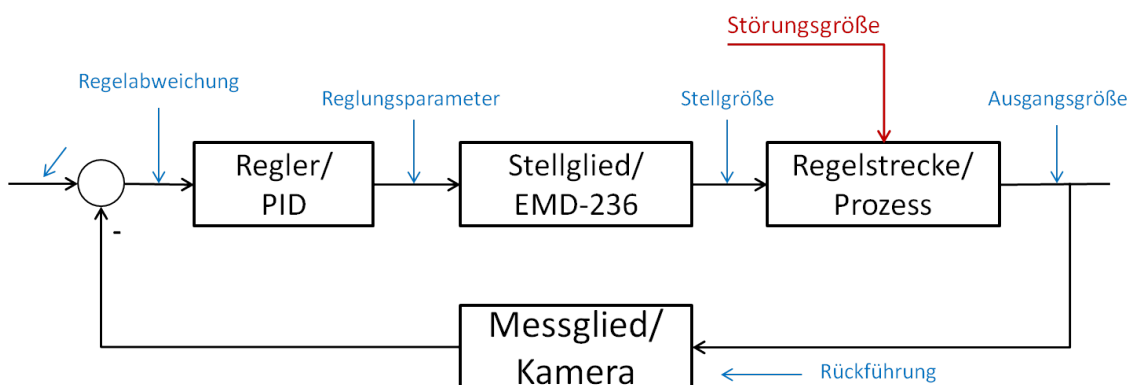


Abbildung 1: Regelungskreis

2.1. PID-Regler

Bei einem PID-Regler (Proportional-Integral-Derivative Controller) handelt es sich um einen aus der Regelungstechnik stammenden Begriff für einen speziellen sich selbst regelnden Steuerkreis.

$$K_{PID}(x) = k_p + \frac{k_i(x+1)}{x-1} + k_d(x-1)$$

In dem vorliegenden Fall des inversen Pendels wurde als Regelsystem ein PID-Regler gewählt. Die Regelstecke besteht aus einem schwingenden System das eine hohe Dynamik bzw. Bewegung des Schlittens erfordert. Es eignet sich hierbei eine Kombination aus dem P-,I- und D-Glied. Das P-Glied ist für die proportionale Stabilisierung verantwortlich. Mit dem I-Glied wird die Genauigkeit des Systems gewährleistet. Das D-Glied ist das Differenzial und hat einen hohen Einfluss auf die Dynamik des Systems, allerdings ist es zusätzlich auch ein Faktor für das Über- oder Unterschwingungsverhalten. Dabei sollte darauf geachtet werden das richtige Dämpfungseinstellungen vorgenommen werden, damit keine kontinuierliche Schwingung bleibt oder im schlimmsten Fall ein aufschwingendes Verhalten generiert wird. Die K-Faktoren sind die Verstärkungsfaktoren der einzelnen Glieder [PDIWS] [Nä15].

3. Vorangegangene Arbeiten

In diesem Abschnitt wird der Aufbau und die grundlegende Steuerung des inversen Pendels behandelt. Da diese Arbeit eine weiterführende Arbeit ist, wird hier kurz auf die bereits vorhandenen Komponenten eingegangen.

3.1. Versuchsaufbau

In diesem Kapitel wird der Aufbau der einzelnen Komponenten und ihre Verbindungen untereinander dargestellt. Hierbei soll auch auf die unterschiedlichen Kommunikationsmöglichkeiten zwischen den Komponenten eingegangen werden.

3.1.1. Überblick

In Abbildung 2 ist der grundlegende Aufbau des inversen Pendels dargestellt, um dem Leser ein besseres Verständnis des Zusammenhangs der hier benutzen Komponenten zu geben.

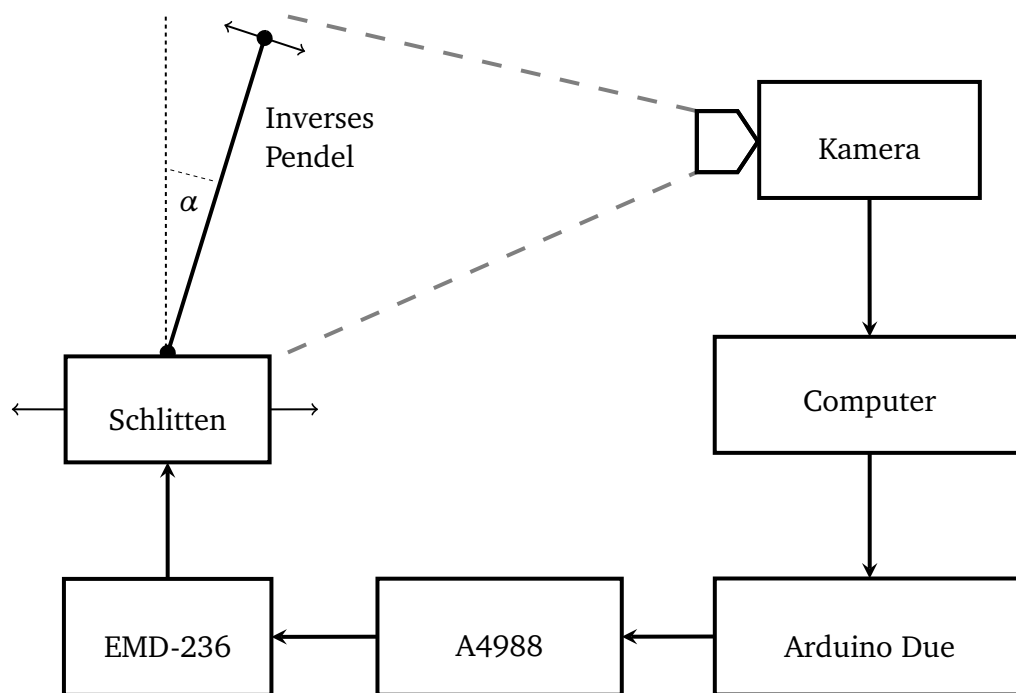


Abbildung 2: Versuchsaufbau des inversen Pendels, Vgl. [Nä15]

Aus einem ausgemusterten Drucker wurde das Schrittlaufband mit Schrittmotor (EMD-236) ausgebaut. Die Druckeinheit wurde entfernt und stattdessen ein inverses Pendel installiert. Durch den DMOS Microstepper Driver A4988 [All09] wird der Schrittmotor angesteuert. Die Steuerung und Regulierung des inversen Pendels übernimmt hierbei ein Arduino Due. Dabei ist es wichtig, dass man zu jedem Zeitpunkt weiß, an welcher Position sich der Schlitten befindet.

3.1.2. Funktionsweise

Zunächst ist die Bestimmung der Position des Pendels erforderlich, um danach über die Regelungstechnik das inverse Pendel ausbalancieren zu können. Das Pendel ist auf dem Schlitten so gelagert, dass es nur einen Freiheitsgrad hat und somit nur in einer Ebene kippen kann. Diese Position wird mit Hilfe einer Kamera ermittelt, die auf das Pendel ausgerichtet ist. Durch Anbringen zweier Markierungen am unteren Drehpunkt des Pendels und am oberen Ende des Pendelstabes, ist es möglich mit einem Bilderverarbeitungsalgorithmus den Winkel des Pendels zu bestimmen. Mit dem Winkel α wird auf dem Arduino Due mittels eines PID Reglers berechnet, mit welcher Geschwindigkeit der Schrittmotor sich bewegen soll und in welche Richtung der Motor bewegt werden muss. So können mögliche Fehlstellungen des inversen Pendels ausgeglichen werden [Nä15].

3.2. Hardware

Um ein inverses Pendel steuern zu können, muss eine diverse Hardware vorhanden sein, auf die in den folgenden Abschnitten im Detail eingegangen wird. Die unterschiedliche Hardware muss aufeinander abgestimmt sein. Dabei muss man darauf achten, dass bei den verwendeten Komponenten die Aus- und Eingangspins mit derselben Spannung betrieben werden.

3.2.1. Arduino Due

Mit dem Arduino Due Board [LLC15a] hat man ein mächtiges Entwicklerboard zur Verfügung, welches das erste Board der Firma Arduino ist, das mit einem Arm-Prozessor AT91SAM3X8E des Herstellers Atmel bestückt ist. Dabei handelt es sich um einen ARM Cortex-M3 [Atm15], der mit einer Taktfrequenz von 84 MHz arbeitet. Durch die zahlreich verfügbare Peripherie eignet sich das Board für die unterschiedlichsten Projekte. Programmiert wird der Arduino Due über eine serielle Schnittstelle und den zusätzlich auf dem Board aufgetragenen ATmega16U2.

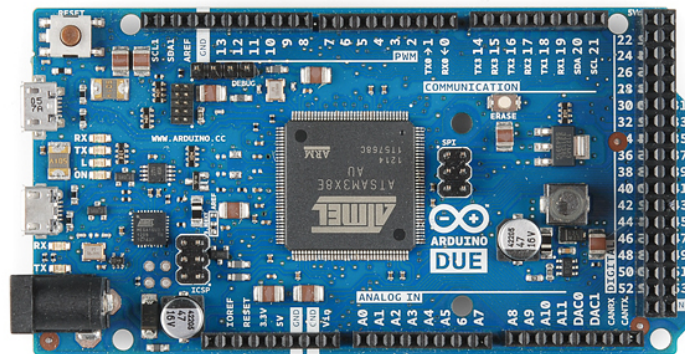


Abbildung 3: Arduino Due. Bildquelle: [LLC15a]

ARM Core Leistungsüberblick

- A 32-Bit Cortex-M3
- CPU Takt mit 84 Mhz.
- 96 KBytes SRAM.
- 512 KBytes Flashspeicher, welcher für Programmcode benutzt werden kann.
- 4 Serielle Schnittstellen.
- 12 Analoge Pins, eine Abtastung von 4096 Werten des Eingangssignals möglich.
- 2 Pins Digital-to-analog converter (DAC).
- Controller Area Network (CAN).
- Inter-Integrated Circuit (I2C), von Atmel Two Wire Interface (TWI) genannt.

3.2.2. Kamera: Playstation EYE

Die verwendete Kamera ist eine Playstation EYE. Sie wurde ausgewählt, da diese Kamera im Preis-Leistungs-Verhältnis gute Ergebnisse liefert. Dabei schafft diese bei einer Auflösung von 640x480 Pixel eine Framerate von 60 Frames per Second (FPS). Bei einer Auflösung von nur 320x240 Pixel sogar 120 FPS [Pla][Hac15b].

Gerade bei der Arbeit mit Bilderverarbeitungsalgorithmen ist die Playstation Eye (PS EYE) eine hervorragende Kamera. So ist es zum Beispiel möglich, Eye-Tracking zu betreiben um auf einer virtuellen Tastatur Eingaben zu machen. Dies wird erreicht, indem mit Infrarotlicht eine Reflexion auf dem Auge erzeugt wird, die die PS EYE erkennen kann und somit die Position des Auges bestimmen kann [Hac15a].

3.2.3. A4988 Schrittmotortreiber

Der DMOS Microstepper Driver A4988 [All09] ist eine Platine, welche zur Ansteuerung von Schrittmotoren benutzt wird. Der A4988 bietet die verschiedenen Schrittgrößen von einem vollen Schritt (Full Step) bis ein sechszentel Schritt (Sixteenth Step). Ein Schritt ist ein periodisches Signal, welches minimal $1\mu s$ an und $1\mu s$ aus sein muss, damit ein Schritt vom A4988 erkannt wird. Die Richtung, in die der Motor sich drehen soll, wird über einen Eingangspin (DIR) gesteuert. Nachdem ein Richtungswechsel vorgenommen wurde, muss am Pin DIR für mindestens 200 ns das Signal angelegt bleiben, bis ein neuer Schritt erkannt wird. Diese schnellen Steuerzeiten sind mit dem A4988 Treiber ausreichend für den Schrittmotor EMD-236, welcher in Abschnitt 4.2.7 und Abschnitt 5.2 erörtert wird.

3.3. Software

In diesem Abschnitt wird auf die verwendete Software und Voraussetzungen auf der Seite des Regulators eingegangen. Im Grunde reicht hierbei ein System aus, welches Python [Ros15] und OpenCV [Gar15] bereit stellt. Desweiteren muss auch der Treiber für die benutzte Kamera vom System bereitgestellt werden, der über eine USB-Schnittstelle eine serielle Verbindung zur Hardware ermöglicht.

3.3.1. Anforderungen an die Kamera

Um die Position des inversen Pendels erkennen zu können, muss eine Kamera gewählt werden, die unter dem verwendeten Betriebssystem unterstützt wird. Unter Linux werden viele diese Treiber durch das Kernelmodul Video4Linux (V4L) unterstützt. Unter anderem werden auch die Treiber für die Playstation Eye von Video4Linux bereitgestellt.

3.3.2. Erkennung des inverses Pendel

Wie schon in Abschnitt 3.1.2 beschrieben, wird der Winkel des Pendels über zwei Markierungspunkte an den Enden des Pendels bestimmt. Um die Auswahl der Farbe eines Markers zu erleichtern wurde eine grafische Oberfläche implementiert. Die Farbe wird über Regler des HSV Farbraumes eingestellt. Zur Erkennung wurde ein Algorithmus gewählt, welcher ein binäres Bild analysiert. In diesem Bild wird nach einer größten zusammenhängenden Fläche gesucht. Daraufhin wird der Mittelpunkt dieser Fläche berechnet [Nä15] [Hou62].

4. Programmierung

In diesem Abschnitt werde ich auf die Programmierung des Arduino Due mittels der vom Hersteller ausgelieferten Arduino Entwicklungsumgebung Arduino IDE eingehen, und welche Einschränkungen es dabei gibt. Eine Alternative bietet das hardwarenahe Atmel Software Framework (ASF), welches im weiteren Verlauf dieses Kapitels erläutert wird.

4.1. Arduino IDE

Die Arduino IDE [LLC15c] ist ein Programm mit einer grafischen Oberfläche (GUI) und Komandozeile, das eine Entwicklungsumgebung für die von Arduino produzierten Produkte bereitstellt. Das Programm ist als Open Source Software erhältlich. Die Benutzeroberfläche ist in Java geschrieben, wobei im Hintergrund die GNU Compiler Collection (GCC) der GNU Compiler Foundation [GCC87] arbeitet. Die geschriebenen Programme werden mit dem GCC zu Binärdateien übersetzt.

4.1.1. Grundlegende Bedienung

Die Arduino Entwicklungsumgebung ist sehr einfach aufgebaut, so dass eine möglichst große Nutzergruppe erreicht werden kann. Im Wesentlichen besteht ein Projekt aus einer Hauptdatei `<filename>.ino`. Der Ordner, in dem die Hauptdatei sich befindet, muss den gleichen Basisnamen haben (Linux Beispiel: `./<fn>/<fn>.ino`). Wie bereits erwähnt, ist dieses Programm auch per Konsolenbefehl bedienbar. Dabei werden die entsprechenden Einstellungen mittels Argumenten an dieses Programm übergeben. Mit einer Makefile kann der gesamte Vorgang automatisiert durchgeführt werden.

Programmcode 1: Arduino IDE main.cpp

```
1  int main(void)
2  {
3      init();
4
5      initVariant();
6
7      #if defined(USBCON)
8          USBDevice.attach();
9      #endif
10
11     setup();
12
13     for (;;) {
14         loop();
15         if (serialEventRun) serialEventRun();
16     }
17
18     return 0;
19 }
```

In der `<filename>.ino` muss eine `loop`-Funktion implementiert werden. Diese muss wiederum von der Arduino IDE in der Datei `main.cpp` angegeben und ersetzt werden. Im Programmcode 1 kann man sehen, dass es noch weitere Funktionen wie `initVariant()` und `setup()` gibt, welche man ebenfalls überladen kann. Meistens wird aber nur `setup()` überschrieben, da in dieser Funktion die Pin-Konfiguration mittels `pinMode(<PinNr>, <Input|Output>)` vorgenommen wird. Zum Kompilieren muss man nur den „Verify“ Button betätigen. Daraufhin fängt die Arduino IDE an die nötigen Dateien zu übersetzen. Sobald der Vorgang vollständig durchgelaufen ist und keine Fehler aufgetreten sind, kann man den übersetzten Code einfach auf das entsprechende Gerät übertragen, welches man zuvor über ein Drop-Down-Menü ausgewählt hat.

4.1.2. Eigenschaften der Arduino IDE

Vorteile:

- Einfache Programmierung, da die Funktionen gut geschrieben sind.
- Leichtes und schnelles testen.
- Für Anfänger sehr gut geeignet, sofern man sich nicht mit der Funktionsweise des Chips auseinandersetzen will.

Nachteile:

- Schwierig die Funktionen nachzuvollziehen, da diese doch sehr komplex geschrieben sind.
- Schwer speziell auf die Hardware (z.B. SAM3X8E) einen angepassten Code zu schreiben. Zum Beispiel einen Timer Counter.
- Langsam beim Übersetzen und Übertragen des Codes, da unnötige Funktionen übertragen werden.
- Ordnerstruktur wird von der Arduino IDE vorgegeben.
- Die Dauer eines Übersetzungsvorgangs liegt bei etwa 30 Sekunden

4.1.3. Probleme mit der Arduino IDE

Anfangs habe ich versucht, den Timer Counter selbst mit den einzelnen Speicherbereichen für nur einen der neun verfügbaren Timer Counter zu programmieren. Dieses habe ich zunächst mit einer Initialisierungsfunktion versucht. Nachdem die Tests in Programmcode 7 erfolgreich waren, habe ich begonnen, den geschriebenen Code in eine C++-Klasse zu überführen. Dabei bin ich auf das Problem gestoßen, dass der GCC der Arduino IDE beim Übersetzen keinen Fehler gemeldet hat und eine fehlerhafte binäre Datei erzeugt hat, die auf dem Arduino Due nach dem Übertragen nicht lauffähig ist. Das Klassenobjekt konnte nicht auf dem Chip allokiert werden. Dies habe ich wiederum nur durch einen weiteren minimalen Test meiner eigens dafür geschriebenen Puffy-Klasse herausgefunden. Diese Puffy-Klasse bestand nur aus einem privaten

Attribut und zwei Funktionen `set ()` und `get ()`. Eine alternative zur Programmierung über die Arduino IDE ist, den Chip direkt zu programmieren. Diese direkte Art der Programmierung nennt man auch bare-metal Programmierung. Diesen Vorgang beschreibe ich im nächsten Kapitel.

4.2. Atmel Software Framework

Die Firma Atmel, welche der Hersteller für den Prozessor auf dem Arduino Due ist, stellt für ihre Produkte ebenfalls ein Framework bereit, das frei als Download zur Verfügung steht. Dieses Framework nennt sich ASF. Es wird in zwei verschiedenen Versionen angeboten: einmal als Atmel Studio, welche nur unter Windows lauffähig ist, und als ASF Standalone Version, welche mittels einer Makefile unter Linux benutzt werden kann. Das ASF unterstützt so viele Peripheriegeräte, dass es den Rahmen dieser Arbeit sprengen würde. Ich beschränkte mich im Folgenden nur auf die Peripherie, die ich in dieser Arbeit auf dem Chip benutze.

4.2.1. Grundlagen

In Folgenden gehe ich näher auf die Peripheriegeräte aus dem Datasheet für den Atmel SAM3X8E [Atm15] ein, die ich genauer untersucht habe und auch in meiner Arbeit konfiguriert und benutzt habe:

- **Power Management Controller (PMC):**
Der PMC stellt die zentrale Taktverwaltung auf dem Chip bereit. Mit dem PMC konfiguriert man den Master Clock (MCK) welcher als Hauptreferenztakt dient. Des Weiteren werden mit dem PMC einzelne Komponenten und Peripheriegeräte des Cortex-M3 gesteuert. Indem man ungenutzte Komponenten deaktiviert, kann man den Energieverbrauch optimieren. Es ist sogar möglich den Cortex-M3 in einen Schlafmodus zu versetzen. Dabei wird der Master Clock auf einen geringeren Takt gesetzt, wobei die Umschaltzeiten im Datenblatt zu beachten sind.
- **Parallel Input/Output Controller (PIO):**
Ein PIO kann bis zu 32 Eingangs-/Ausgangsleitungen steuern. Dabei kann es sich um eine interne Peripherie oder einen normalen Ein- oder Ausgang handeln. Es ist sogar möglich einen Ausgang direkt wieder als Eingang zu benutzen. Der Arduino Due besitzt 4 PIOs. PIOA steuert nur 30 Leitungen, PIOB eine maximale Anzahl von 32, PIOC wiederum nur 31 und PIOD sogar nur 10. Damit kommt man auf 144 konfigurierbare Leitungen/Signale auf einem ganzen Board.
- **Peripheral Direct Memory Controller (PDC):**
Der PDC ist nichts anderes als ein ganz normaler Direct Memory Access Controller (DMAC). Um dem PDC die Daten zu übergeben, die gesendet werden sollen, muss man diesem mit einem 32-Bit Pointer mitteilen, an welcher Stelle die Daten anfangen und zudem angeben, wieviele Daten übermittelt werden sollen.

- **Universal Asynchronous Receiver Transceiver (UART):**

Die UART Komponente bietet dem Entwickler die Möglichkeit über das, nach dem RS-232 Standard [Str] implementierte, Protokoll mit dem Chip Daten auszutauschen. Um hierbei den Prozessor nicht unnötig mit Wartezyklen zu belasten, benutzt man hierzu – wie zuvor dargestellt – einen PDC.

- **Timer Counter (TC):**

Der Timer Counter ist ein komplexes Zählwerk, welches parallel zum Cortex-M3 Prozessor arbeitet. Der Zähler besteht aus einem 32-Bit Register, welches pro Takt um eins inkrementiert wird. Ein solches Zählwerk hat drei Vergleichsregister RA, RB und RC. Die Vergleichsregister können mit unterschiedlichen Werten initialisiert werden. Sobald der Zählerstand einen gesetzten Wert erreicht hat, wird ein Ereignis ausgeführt. Dabei kann man den Ausgang des Timer Counter entweder auf HIGH/LOW setzen, toggeln oder nicht verändern. Ein Timer Counter ist so aufgebaut, dass dieser 3 individuell steuerbare Timerkanäle besitzt. Diese können pro Timer Counter Modul einzeln gesteuert werden. Mit einem Timer Counter ist man auch in der Lage ein Signal an einem Ausgang zu verzögern.

4.2.2. Makefile für Atmel Software Framework

In diesem Abschnitt werde ich auf die grundlegenden Befehle und Anweisungen in der Makefile eingehen, die man bei der Entwicklung eines Projektes für den Arduino Due mit dem ASF zu beachten hat. Der Arduino Due wird mittels eines Atmel ATmega16U2 [Atm12] programmiert. Mit diesem zusätzlichen Chip ist es jederzeit möglich eine neue binäre Programmdatei zu übertragen, falls zuvor ein fehlerhaftes Programm übertragen wurde. Eine spezielle Eigenschaft, die dabei zu beachten ist, ist, dass man den ATmega16U2 Chip mit einer Baudrate von 1200 über den seriellen Port ansprechen muss, damit dieser in den Programmiermodus schaltet, um dann mittels des BOSSA Flashers [Shu] den SAM3X8E zu programmieren.

Programmcode 2: Makefile

```
1 SRC += main.c
2 OUTPUT = main
3 TTY_DEVICE_PORT = $(shell ./detect_serial_port.sh)
4
5 ASF_ROOT = /opt/asf-standalone-archive-3.29.0.41/xdk-asf
   ↪ -3.29.0
6
7 INCLUDES += -I.
8 INCLUDES += -I$(ASF_ROOT)/common/boards
9 [...]
10 ASF_SRC += $(ASF_ROOT)/sam/utils/cmsis/sam3x/source/
   ↪ templates/system_sam3x.c
11 [...]
12
13 upload: $(OUTPUT).bin
14     stty -F $(TTY_DEVICE_PORT) 1200
15     bossac -e -w -v -b $(OUTPUT).bin -R
```

Im Programmcode 2 sind ein paar der wichtigsten Befehle als Ausschnitt aus der, im Anhang A abgebildeten, Makefile aufgeführt. Mit der Variable SRC gibt man die Quelldateien an, die übersetzt werden sollen. Durch OUTPUT gibt man den Namen seiner Ausgabedatei an. In Zeile 3 der Makefile kann man direkt das zu programmierende Gerät angeben oder das Skript `detect_serial_port.sh` benutzen, welches in Listing 3 angegeben ist.

Auf einem Linuxsystem prüft das Skript alle `/dev/ttyACM*` Geräte. Mit dem Befehl `udevadm info -q property -n i` werden die Produkteigenschaften zu einem Gerät angezeigt. Anhand dieser Informationen wird nun geprüft, ob das aktuell ausgewählte Gerät mit der zu Beginn des Skript gegebenen `ID_Model` und `ID_VENDOR` Kombination übereinstimmt. Falls dies der Fall ist, wird das entsprechende Gerät ausgegeben.

Programmcode 3: `detect_serial_port.sh`

```
1 #!/bin/bash
2 ID_MODEL=003d
3 ID_VENDOR=2341
4
5 for i in `ls /dev/ttyACM*`; do
6     property=$(udevadm info -q property -n $i)
7
8     idproduct_found=false
9     idvendor_found=false
10
11     if [[ $property =~ "$ID_MODEL" ]]; then
12         idproduct_found=true
13     fi
14
15     if [[ $property =~ "$ID_VENDOR_ID" ]]; then
16         idvendor_found=true
17     fi
18
19     if [[ "$idvendor_found" == "true" ]] && [[ "
    ↳ $idproduct_found" == "true" ]]; then
20         echo $i
21     fi
22 done
23
24 exit 0
```

In der Makefile wird das gefundene Gerät dann der Variable `TTY_DEVICE_PORT` zugewiesen. Da der Arduino Due zwei USB-Anschlüsse hat, muss bei der Entwicklung darauf geachtet werden, welches Kabel als erstes eingesteckt werden muss. Da der Linuxkernel die `ttyACM`-Nummerierung über den `udev`-Dienst automatisch vornimmt, kommt man dabei leicht durcheinander. Deshalb ist dieses kleine Skript sehr hilfreich. Durch die Angabe des `ASF_ROOT` teilt man der Makefile den Wurzelfad zum ASF Standalone Verzeichnis mit. Somit ist die Angabe der `INCLUDES` und `ASF_SCR` wesentlich übersichtlicher.

Die Angabe der `CFLAGS` im Anhang A muss für die jeweilige Architektur einmalig angepasst werden. Die `CFLAGS` konfigurieren welche Warnungen ausgegeben werden und welche Optimierungen GCC vornehmen soll. Die Variable `LDFLAGS` beinhaltet die Angabe des Linkerskripts, wobei darin definiert ist, wie die einzelnen, erzeugten Objektdateien in der Binärdatei angeordnet werden.

4.2.3. Chip beschreiben

Ein wichtiger Abschnitt in der Makefile, auf den ich noch explizit eingehen möchte, ist der Abschnitt zum Übertragen des übersetzten Quellcodes. In Zeile 13 ist die Direktive `upload` definiert. Diese prüft, ob die Datei `main.bin` bereits existiert. Liegt diese vor, wird über eine serielle Schnittstelle zu dem ermittelten seriellen Gerät eine Verbindung mit der Baudrate 1200 in Zeile 14 aufgebaut. In der letzten Zeile wird der Befehl `bossac` dann mit dem Parameter `-e`, der dem Flashspeicher zugeordnet ist, gelöscht und mittels `-w` wieder neu beschrieben. Der Parameter `-v` gibt dabei an, dass der Flashspeicher nochmals mit der angegebenen Binärdatei geprüft werden soll. Der Parameter `-b` gibt an, dass der Prozessor aus dem FLASH-Speicher booten soll und die Option `-R` sorgt dafür, dass die CPU nach erfolgreicher Übertragung neu gestartet wird.

4.2.4. LED-Blinker Implementierung

Den ersten Test, ob die Programmierwerkzeuge richtig konfiguriert sind und miteinander arbeiten, nennt man in den meisten Fällen ein „Hallo Welt!“-Programm. Auf einer Hardware wie dem Arduino Due, ist der erste Funktionstest eine blinkende LED zu programmieren. Bei Mikrocontrollern nennt man dieses Programm auch „Herzklopfen“. Dabei wird eine LED mit einem passenden Widerstand an einem Pin angeschlossen. Der erste Funktionstest besteht nun darin dem Mikrocontroller Leben einzuhauchen, wobei die LED wie ein Herzschlag oder Lebenszeichen des Chips aufleuchtet.

Im Programmcode 4 ist eine minimalistische Implementierung eines solchen Tests für den Arduino Due dargestellt. Bei einem Neustart des Systems wird die `main()` in Zeile 13 aufgerufen. Anschließend wird die `SystemInit()` aufgerufen und der MCK auf dem Board initialisiert. Dieser wird per Standardwert bei einem Arduino Due auf 84 MHz eingestellt. Die Funktion ist in der ASF Standalone Version in dem Pfad `sam/utils/cmsis/sam3x/source/templates` und der Quelldatei `system_sam3x.c` zu finden. Damit der SAM3X8E sich nicht selbst neu startet, weil der Watchdog Timer WDT² übergelaufen ist, deaktiviert man diesen mit der Zeile 16. Hier schreiben wir in das Watchdog Timer Register `WDR_MR` in die 15te Stelle des Registers eine 1 hinein. Damit ist der Watchdog Timer deaktiviert und unser Chip startet nicht mehr neu.

In der Funktion `setup_led()` wird nun die Konfiguration des Parallel Input/Output (PIO) Controller vorgenommen. Hierbei wird der PIN `PIO_PB25`, welcher nach der Arduino Due Pinbelegung [LLC15b] der PIN 13 bzw. PIN L für die auf dem Board

²Bei einem Watchdog Timer handelt es sich um einen Timer, welcher den Prozessor neu starten kann, falls sich dieser in einem Deadlock befindet. Auf dem AT91SAM3X8E ist ein 12-Bit Timer, der nach spätestens 16 Sekunden neustartet [Atm15].

integrierte LED ist, als Ausgang konfiguriert. Dafür muss man die Signalpfade korrekt auf den PIN ansteuern. Mit dem Register `PIO_PER` konfiguriert man parallel die zwei Multiplexer, die dafür zuständig sind, ob das Signal von einer Peripherie kommt oder ob man es über das Register `PIO_CODR` setzen kann. Durch das Setzen des Registers `PIO_OER`, welches das PIO Output Enable Register ist, wird der Ausgang an den PIN endgültig durchgeschaltet. Bei dem Arduino Due muss man noch beachten, dass es sogenannte Pull-up Widerstände gibt, die den Ausgang auf eine logische Eins hochziehen können. Dieser Widerstand wird mit der Zeile 9 `PIO_PUDR` für den PIN `PIO_PB27` deaktiviert. In Zeile 10 wird das PIO Output Data Register `PIO_CODR` gesetzt und damit die LED ausgeschaltet. Nun Betrachten wir noch die `while`-Schleife im Programm. Diese ist eine Endlosschleife. Die LED wird durch die Anweisung in Zeile 20 eingeschaltet. Die darauf folgende `for`-Schleife erzeugt eine Pause, bevor die LED für dieselbe Zeit wieder ausgeschaltet wird. Als Ergebnis haben wir nun eine blinkende LED, die wir „Herzklopfen“ nennen.

Programmcode 4: LED-Blinker Source Code

```
1 #include <system_sam3x.h>
2 #include <sam3x8e.h>
3 #include <pio.h>
4
5 static void setup_led()
6 {
7     PIOB->PIO_PER = PIO_PB27;
8     PIOB->PIO_OER = PIO_PB27;
9     PIOB->PIO_PUDR = PIO_PB27;
10    PIOB->PIO_CODR = PIO_PB27;
11 }
12
13 int main()
14 {
15     SystemInit();
16     WDT->WDT_MR = WDT_MR_WDDIS;
17     setup_led();
18
19     while (1) {
20         PIOB->PIO_SODR = PIO_PB27;
21         for (int i = 0; i <= 8400000; i++) {}
22         PIOB->PIO_CODR = PIO_PB27;
23         for (int i = 0; i <= 8400000; i++) {}
24     }
25     return 0;
26 }
```

4.2.5. UART Implementierung

Über die UART-Schnittstelle wird der Winkel, aktuell eine halbgenaue Gleitkommazahl mit 2 Byte, an den Arduino Due übertragen. Der gesamte Quelltext dazu findet sich im Anhang C.

Zu Beginn muss man Speicher zum Empfangen (`uart_receive_buffer`) bzw. Senden (`uart_transmit_buffer`) allokiert. Wenn man gleichzeitig senden und empfangen möchte, braucht man zwei getrennte Speicher. In der `setup_pmc()` wird die benötigte Peripherie PMC, PIO, DMAC und UART aktiviert. Daraufhin setzt man in der `setup_pdc_uart()` die Speicheradressen des UART Transmit Pointer und UART Receive Pointer. Zu jedem Pointer gehört jeweils auch eine Speichergröße, welche über die UART Transmit/Receive Counter Register `UART_TCR`, `UART_RCR` gesetzt wird.

Programmcode 5: UART Ausschnitt 1

```
1  UART->UART_IDR = 0xFFFFFFFF;
2  NVIC_EnableIRQ(UART_IRQn);
3  UART->UART_IER = UART_IER_RXBUFF;
```

Der UART Controller selbst muss nun noch konfiguriert werden. Dies wird in der `setup_uart()` gemacht. Zu Beginn wird die Pin-Konfiguration vorgenommen, indem der PIO-Kontroller für die RX (`PIO_PA8A_URXD`) und TX (`PIO_PA9A_UTXD`) Pins so konfiguriert wird, dass die Signale an den UART-Kontroller weitergereicht werden. Um die Übertragungsrate zu setzen, muss das Register `UART_BRGR` des UART gesetzt werden. Um eine Baudrate von 115200 zu setzen, muss eine 45 in das Register geschrieben werden, da für den Baudratengenerator die folgende Formel die Übertragungsrate generiert:

$$\frac{\text{Master Clock}}{16 * x} = \text{Baudrate}$$

Damit der Prozessor durch einen Interrupt des UART Controller unterbrochen werden kann, muss dieser bei dem Nested Vector Interrupt Controller (NVIC) für diese Peripherie aktiviert werden. Dies wird in Zeile 2 in Programmcode 5 gemacht. In der Zeile davor sorgt man dafür, dass alle Interrupts deaktiviert werden, damit nicht unerwartet doch ein Interrupt ausgeführt wird. Nun wird in Zeile 3 der Interrupt `RXBUFF` aktiviert, wobei dieser ausgelöst wird, sobald der Speicher zum Empfangen von Daten voll ist. Dies wird so umgesetzt, dass bei jedem empfangenen Byte der `UART_RCR` um eins dekrementiert wird. Sobald dieses Register 0 erreicht hat wird der Interrupt ausgelöst.

Im Fall, dass ein Interrupt ausgelöst wurde, wird die Interrupt Service Routine (ISR) (`UART_Handler`) aufgerufen (Programmcode 6). Um das Interrupt Flag wieder zu löschen, muss das Register `UART_IMR` ausgelesen werden. In Zeile 5 vergleicht man, ob genau der Interrupt `RXBUFF` aktiviert war. Falls dies der Fall ist, werden die Daten mittels der `for`-Schleife in den Sendespeicher kopiert. Nun müssen noch die Pointer und Speichergrößen gesetzt werden. Somit können wieder neue Daten empfangen

werden und die zuvor gesendeten Daten werden durch den DMAC parallel übertragen. Dies stellt eine einfache `echo`-Funktion der gesendeten Daten dar.

Programmcode 6: UART Interruptroutine

```

1 void UART_Handler(void)
2 {
3     uint32_t uart_imr = UART->UART_IMR;
4
5     if (uart_imr & (UART_IMR_RXBUFF))
6     {
7         for (int i = 0; i < BUFFERSIZE; i++){
8             uart_transmit_buffer[i] = uart_receive_buffer[i];
9         }
10
11         UART->UART_RPR = (uint32_t) uart_receive_buffer;
12         UART->UART_RCR = BUFFERSIZE;
13         UART->UART_TPR = (uint32_t) uart_transmit_buffer;
14         UART->UART_TCR = BUFFERSIZE;
15     }
16 }

```

4.2.6. Timer Counter Implementierung

Die Timer Counter Implementierung folgt demselben Schema wie die zuvor behandelten Implementierungen. Zunächst wird in Zeile 5-8 des Programmcode 7 der PIN für den zugehörigen Timer so konfiguriert, dass dieser auf die Peripherie weitergeleitet wird.

0x40080000	SPI1	25
	TC0	TC0
+0x40		27
	TC0	TC1
+0x80		28
	TC0	TC2
0x40084000		29
	TC1	TC3
+0x40		30
	TC1	TC4
+0x80		31
	TC1	TC5
0x40088000		32
	TC2	TC6
+0x40		33
	TC2	TC7
+0x80		34
	TC2	TC8
0x4008C000		35
	TWI0	

Abbildung 4: Inkonsistente Bezeichnung der Timer Counter im Datenblatt. Bildquelle: [Atm15]

Die Konfiguration des Timers Counter Moduls ist etwas komplizierter. Hierbei ist das Datenblatt etwas inkonsistent, wie man in Abbildung 4 sehen kann. Ein Timer Counter besteht aus einem Modul wie in Abschnitt 4.2.1 beschrieben. Beim Programmieren wird ein solches Modul mit TC0, TC1 oder TC2 bezeichnet. Dabei handelt es sich um die Basisadressen, die wie in Abbildung 4 zu sehen, an den Adressen 0x40080000, 0x40084000 und 0x40088000 beginnen. Hierbei sieht man auch, dass für jeden Kanal zur Konfiguration ein Adressbereich von 0x40 Bit zur Verfügung steht. Wobei auch noch Reserve-Speicherstellen vorhanden sind für Weiterentwicklung seitens des Herstellers Atmel. Die Timer werden aber auch der Reihe nach von TC0 bis TC8 gekennzeichnet. Diese Kennzeichnung spiegelt sich dann bei der zugehörigen ISR wieder. Diese werden nach dem Schema `TC[0-8]_Handler()` bezeichnet.

In den Zeilen 10,11 des Programmcode 7 wird zuerst der Takt für den TC0 Kanal 0 und alle Interrupts vorerst deaktiviert. Um die gewünschte Funktionalität des TC0 zu bekommen, müssen die richtigen Bits gesetzt werden (Zeile 12-16). Im Grunde wird in den Speicher an der Adresse des TC_CMR der folgende Binärwert 0b00000000010001101100000000000000 geschrieben, was für den Menschen schwer zu lesen ist. Dabei bedeutet TC_CMR_WAVESEL_UP_RC, dass das Ausgangssignal zu Beginn auf HIGH liegt und bei einem Vergleich des Registers C der Timer Counter neu gestartet wird. TC_CMR_WAVE gibt an, dass der Timer im Wavemodus und nicht im Capture Modus arbeitet. Die weiteren zwei Parameter sorgen dafür, dass der Ausgang an den richtigen Zeitpunkten auf den dafür vorgesehenen Wert gesetzt werden. TC_CMR_ASWTRG_SET sagt, dass der Timer nach einem erfolgreichen Vergleich mit dem Register C neu gestartet wird.

Nun werden in Zeile 17 und 18 die Vergleichswerte in die Register A und C des Zählers geschrieben. Somit wird der Ausgang zu Beginn auf HIGH gesetzt. Sobald der Zählerstand TC_CV den Wert des Registers A TC_RA erreicht hat, wird der Ausgang auf LOW gesetzt. Genau dann wenn der Zählerwert dem Wert des Registers C entspricht, wird der Ausgang wieder auf HIGH geschaltet.

Programmcode 7: Timer Counter Initialisierung

```

1  static void setup_timer_TC0 ()
2  {
3      uint32_t ul_sr;
4
5      PIOB->PIO_IDR = PIO_PB25B_TIOA0;
6      PIOB->PIO_PDR = PIO_PB25B_TIOA0;
7      PIOB->PIO_ABSR = PIO_PB25B_TIOA0 | ul_sr;
8      PIOB->PIO_PUER = PIO_PB25B_TIOA0;
9
10     TC0->TC_CHANNEL[0].TC_CCR = TC_CCR_CLKDIS;
11     TC0->TC_CHANNEL[0].TC_IDR = 0xFFFFFFFF;
12     TC0->TC_CHANNEL[0].TC_CMR = TC_CMR_WAVSEL_UP_RC |
13                                 TC_CMR_WAVE |
14                                 TC_CMR_ACPA_CLEAR |
15                                 TC_CMR_ACPC_SET |
16                                 TC_CMR_ASWTRG_SET;
17     TC0->TC_CHANNEL[0].TC_RA = (uint32_t) (0x29040);
18     TC0->TC_CHANNEL[0].TC_RC = (uint32_t) (0x52080);
19
20     NVIC_EnableIRQ(TC0_IRQn);
21     TC0->TC_CHANNEL[0].TC_IER = TC_IER_CPCS;
22
23     TC0->TC_CHANNEL[0].TC_CCR = TC_CCR_SWTRG |
24                                 TC_CCR_CLKEN;
25 }

```


Die Zeilen 20 und 21 aktivieren den Timer TC0 für Interrupts und ein Interrupt wird ausgelöst, sobald der Zähler den Wert des Registers C erreicht hat. Zum Schluss wird der Timer neugestartet und aktiviert.

Bei der Auswahl des Timers muss man sich überlegen, für welchen Zweck man diesen im Projekt einsetzen möchte. Auf dem AT91SAM3X8E haben die Timer Counter TC3, TC4 und TC5 keine Verbindungsleitung, die von extern zugänglich sind. Diese 3 Timer wären geeignet um Programm Verzögerungen, interne Wartezeiten oder wiederkehrende Aufgaben zu erledigen.

4.2.7. Ansteuerung eines Schrittmotors

Mit den oben genannten Implementierungen kann man nun die Implementierung der Ansteuerung des Schrittmotors zur Regulierung des inversen Pendels beginnen. Die vorangegangene Abschnitte waren wichtig um zu prüfen, dass die Komponenten im Einzelnen einwandfrei funktionieren. Die Fehlersuche wird somit um ein Vielfaches eingeschränkt. Durch das einzelne Verständnis der Funktionalität der einzelnen Komponenten kann man diese leichter miteinander verbinden und zu einem komplexen Programm zusammenführen.

Der benutzte Timer Counter ist wie in Abschnitt 4.2.6 initialisiert. Die dazugehörige ISR des TC0_Handler wird aufgerufen, wenn der Zählerstand des TC0 den Vergleichswert im Register TC_RC erreicht hat. Diese sorgt dafür, dass der Schrittmotor sich in einem Bereich von 200 Schritten hin- und herfährt.

Als erster Befehl wird das Statusregister TC_SR des TC in Zeile 3 ausgelesen. Damit wird das Interrupt Flag gelöscht. Da wir bei der aktuellen Konfiguration mit keinem anderen Interrupt rechnen müssten, da die restlichen Interrupts deaktiviert sind, braucht man hier nicht extra prüfen, welcher Interrupt ausgelöst wurde. In Zeile 5 wird geprüft in welche Richtung sich der Schrittmotor bewegt und dementsprechend muss die Position `pos` inkrementiert oder dekrementiert werden. Dieser Ablauf wird solange fortgeführt, bis entweder `pos` 200 oder 0 beträgt. Sobald dies der Fall ist, wird der Timer Counter mit `TC_CCR_CLKDIS` im Register `TC_CCR` deaktiviert. Die richtige `direction` wird entsprechend gesetzt. Die in Zeile 11 und 22 definierten `for`-Schleifen generieren eine Pause, welche die in Abschnitt 3.2.3 definierte Schaltzeit einzuhält. Der Ausgangspin muss entsprechend der Richtung noch in Zeile 12 bzw. 23 gesetzt oder gelöscht werden. Als letzter Befehl in dem TC0_Handler wird der Timer Counter mit `TC_CCR_SWTRG` neu gestartet und mit `TC_CCR_CLKEN` aktiviert.

Programmcode 8: ISR zur Steuerung des Schrittmotors

```
1 void TC0_Handler(void)
2 {
3     TC0->TC_CHANNEL[0].TC_SR;
4
5     if(direction)
6     {
7         if( pos == 200 )
8         {
9             TC0->TC_CHANNEL[0].TC_CCR = TC_CCR_CLKDIS;
10            direction = 0;
11            for(int i = 0; i <= 84000; i++){ }
12            PIOC->PIO_SODR = PIN_DIR;
13        }
14        pos = pos + 1;
15    }
16    else
17    {
18        if( pos == 0 )
19        {
20            TC0->TC_CHANNEL[0].TC_CCR = TC_CCR_CLKDIS;
21            direction = 1;
22            for(int i = 0; i <= 84000; i++){ }
23            PIOC->PIO_CODR = PIN_DIR;
24        }
25        pos = pos - 1;
26    }
27
28    TC0->TC_CHANNEL[0].TC_CCR = TC_CCR_SWTRG |
29                                TC_CCR_CLKEN;
30 }
```

5. Schlussfolgerung

Im Folgenden werden die Verbesserungen durch die direkte Programmierung erläutert. Des Weiteren wird auf Probleme der einzelnen Komponenten eingegangen. Durch die Einschränkung und Grenzen der aktuell genutzten Hardware wird in diesem Abschnitt auch noch auf neu veröffentlichte Hardware eingegangen, die zur weiteren Entwicklung genutzt werden könnte.

5.1. Zeitersparnis

Bei Anwendung einer hardwarenahen Programmierung mit dem Atmel Software Framework muss man sich nicht selbst um einen eigenen Treiber für einzelne Komponenten auf dem Chip kümmern, sondern man kann hier auf gegebenen Funktionen des Frameworks zurückgreifen. Den Überblick über die exakte Ausführung auf dem Chip kann man dabei schnell verlieren, da man wieder die Definitionen der Funktionen suchen und analysieren muss. Dies ist teilweise nicht ganz einfach, da das ASF ein sehr komplexes Framework bereitstellt.

Die Arduino IDE stellt eine einfache Entwicklungsumgebung bereit, bei der sogar Anfänger schnell und einfach ein lauffähiges Programm schreiben können, da die vorhandenen Funktionen sehr gut auf der eigenen Homepage [LLC] dokumentiert sind. Für eine einfache Ausgabe über eine seriellen Schnittstelle muss man lediglich die `Serial.begin()` mit der gewünschten Baudrate aufrufen. Damit kann man bereits Daten über die serielle Schnittstelle komfortabel senden und empfangen. Ein Nachteil dieses Komforts ist, dass man beim Schreiben von zeitkritischen Programmen nicht sofort sagen kann, wieviele Unterfunktionen aufgerufen werden und wieviel Zeit diese in Anspruch nehmen.

Einen Zeitfaktor den man bei der Entwicklung nicht unterschätzen darf, ist das Übertragen des Programmcodes auf die Hardware. Über die Arduino IDE dauert es ca. 17,428 Sekunden bis der Programmcode der Implementierung eines einfachen LED-Blinklichts übersetzt, geprüft und übertragen ist. Bei einer Implementierung mit dem ASF dauert dieser Vorgang lediglich ca. 4,410 Sekunden. Wohingegen der Programmcode für die Steuerung eines inversen Pendels mittels Arduino IDE ca. 26,694 Sekunden und mit dem ASF nur ca. 6,310 Sekunden. Das sind etwa 20 Sekunden Zeitersparnis pro Übersetzung und Übertragung des Programmcodes auf die Zielhardware. Bei einer Fehlersuche in einem Programmcode ist dies ein enormer Zeitfaktor.

5.2. Implementierungsprobleme

Das Datenblatt [Atm15] beinhaltet alle Spezifikationen, die nötig sind um den SAM3X8E zu programmieren. Aber ohne das nötige Grundlagenwissen, wie die Funktionsweise einer Komponente sich verhält, kann man damit wenig anfangen. Desweiteren wie man aus dem Abschnitt 4.2.6 entnehmen kann, führt die Inkonsistenz des Datenblatt zu weiteren Problemen. Copy&Paste Fehler, welche ich in dem Datenblatt gefunden haben, führen auch dazu, dass man sich zusätzlich überlegen muss, ob das geschriebene auch wirklich so stimmen kann.

Weitere Probleme sind mit dem A4988 Treiber aufgetreten. Hierbei war die Ansteuer-

rung des Schrittmotors nicht ohne weiteres möglich, da die maximale Frequenz, die der A4988 am Ausgang zur Ansteuerung eines Schrittmotors liefern kann, viel zu hoch ist. Deshalb mussten die Ansteuerzeiten empirisch für den Schrittmotor EMD-236 ermittelt werden. Zusätzlich kommt noch hinzu, dass der Schrittmotor eine Beschleunigungsrampe benötigt um eine höhere maximal Geschwindigkeit zu erreichen. Der Schrittmotor braucht zum Anfahren ein periodisches Signal von einer Gesamtzeit von 2ms. Wobei dieses in einem linearen Verlauf auf ein periodisches Signal von 0,8ms verkürzt werden kann. Sobald man mit einer höheren Frequenz den Motor ansteuert, fängt dieser an zu blockieren. Desweiteren führen Verschmutzung der Laufschiene und Abnutzung des Riemens bei Bewegung des Schlittens zu weiteren unerwarteten Fehlverhalten.

5.3. Weiterführende Arbeit

Um bessere Ergebnisse dieses Demonstrators zu erreichen, schlage ich vor, folgende Punkte als nächstes zu testen und weiter zu entwickeln:

5.3.1. JTAG Hardware

Bei der Entwicklung eines eingebetteten Chips auf einem Board, welches nur über die serielle Schnittstelle ansprechbar ist, ist es sehr schwierig die Software auf Fehler zu prüfen. Der Arduino Due besitzt einen JTAG Anschluss, welcher dem Programmierer erlaubt direkt auf die Prozessorregister zuzugreifen und den Programmfluss Takt für Takt auszulesen. Diese Methode ermöglicht es zu jeden Zeitpunkt genau zu wissen, wie der Prozessorzustand aussieht. Nur mit einer LED bewaffnet, ist es schwer einen Fehler im Programmcode zu finden, da man nur über eine visuelle Kommunikation einer LED verfügt, aber nichts über den exakten Zustand eines Registers erfährt.

5.3.2. Raspberry Pi Kamera Modul

Zur Erkennung eines sich bewegenden Körpers braucht man möglichst viele Bilder in einem kurzen Abstand, um die Bewegung genau zu erkennen. Dabei ist die PS EYE keine schlechte Wahl für die Bilderkennung. Aber durch die geringe Auflösung von nur 320x240 Pixel bei 120 FPS und bei schwankender Beleuchtung kommt diese Kamera an ihre Grenzen. Durch die Fehlerkennungen der angebrachten Markierungen werden falsche Daten an den Mikrocontroller übertragen. Diese können dazu führen, dass der Schrittmotor zu schnell, zu langsam oder sogar komplett in die falsche Richtung gesteuert wird und somit das Pendel nicht mehr richtig reguliert werden kann.

Es ist kaum möglich eine bessere Kamera zu diesem Preis-Leistungs-Verhältnis zu bekommen. Seit 2014 hat Sony den IMX219PQ [Son] CMOS Farbsensor auf den Markt gebracht, der auf dem Raspberry Pi Kamera Modul benutzt wird. Mit diesem ist es möglich bei einer Auflösung von 1280x720 Pixel mit 180 FPS aufzunehmen. Durch die höhere Auflösung stehen mehr Pixel zur Erkennung der Marker zur Verfügung. Zudem kann die Bewegung durch die höhere Framerate genauer erfasst werden.

5.3.3. Steuerung über einen Raspberry Pi 2/3

Zum Verarbeiten der Bilddaten, welche von der Kamera [Pid] geliefert werden, würde ich einen Raspberry Pi 2 oder 3 [Pia][Pic] empfehlen, da die verwendete Software in Python geschrieben ist und die verwendete Bildverarbeitungsbibliothek OpenCV auch auf einem ARM-Prozessor ohne weitere Probleme läuft. Ein zusätzlicher Vorteil hierbei wäre, dass man die Kamera direkt an den Raspberry Pi anschließen kann, weil die Hardware und Anschlüsse direkt aufeinander abgestimmt sind.

Ein weiterer Vorteil den Raspberry Pi 2 einzusetzen ist, dass dieser mit einem real-time Kernel [Pib] ausgestattet werden kann. Mit diesem Patch ist es möglich feste Zeitkriterien zur Ausführung von Programmcode einzuhalten.

6. Zusammenfassung

Um ein inverses Pendel steuern zu können, müssen alle Komponenten richtig miteinander interagieren. Dabei muss man exakt wissen wieviel Zeit für die Berechnung der Regelgröße benötigt wird.

Bei einem Projekt mit solchen Anforderungen ist es ebenso relevant zu wissen, wieviel Zeit die Ausführung einer Funktion benötigt. Des Weiteren möchte man die volle Kontrolle über den Chip haben. Dafür sind fundierte Kenntnisse über den Aufbau der Hardware nötig um die Hardware effizient einsetzen zu können.

Bei der Steuerung eines inversen Pendels benötigt man eine kontinuierliche Bewegung des Schrittmotors. Die Implementierung der Steuerung eines Schrittmotors, wie sie in der vorliegende Arbeit vorgenommen wurde, erfüllt genau dies. Durch die direkte Programmierung ist man nicht zwingend an ein Framework bei der Entwicklung gebunden. Während die Implementierung dadurch aufwändiger wird, hat man im Gegenzug auf die gesamte Hardware Zugriff. Dies wird durch die Benutzung des Timer Counters (TC) verdeutlicht. Der TC bietet die Möglichkeit einen Prozess parallel zu steuern und somit den Prozessor so wenig wie möglich zu belasten. Die zusätzliche Nutzung des DMAC erlaubt es Speicherzugriffe parallelisiert auszuführen.

Das ist gerade bei zeitkritischen Projekten sehr wichtig.

Ein Nachteil der direkten Programmierung ist, dass man jede Komponente selbst konfigurieren und auf Korrektheit testen muss. Vor allem beim Testen sollte man sich mit einem JTAG-Debugger vertraut machen, sofern man auf einem eingebetteten System arbeitet. Durch diese Technik kann man die Ausführung des Programms direkt auf der Zielhardware verfolgen und Fehler schneller erkennen.

Literatur

- [All09] Allegro. *Allegro A4988 Microstepping Driver Datasheet*. 2009. URL: <http://www.allegromicro.com/~Media/Files/Datasheets/A4988-Datasheet.ashx>.
- [Atm12] Atmel. *Atmel ATmega 16U2*. 2012. URL: <http://www.atmel.com/devices/ATMEGA16U2.aspx>.
- [Atm15] Atmel. *Atmel SAM3X8E Microcontroller Datasheet*. 2015. URL: http://www.atmel.com/Images/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf.
- [CM3] CM3. *HD Kugelschreiber Spy Cam Gold*. URL: <http://www.cm3-shop.de/hd-kugelschreiber-spy-cam-gold>.
- [Gar15] Willow Garage. *OpenCV*. 2015. URL: <http://opencv.org/>.
- [Hac15a] Hackaday. *Kids type with their eyes, robot arm prints their words*. 2015. URL: <http://hackaday.com/2010/12/11/kids-type-with-their-eyes-robot-arm-prints-their-words/>.
- [Hac15b] Hackaday. *PS3 Eye Lives Again Thanks To Low Prices*. 2015. URL: <http://hackaday.com/2015/05/20/ps3-eye-lives-again-thanks-to-low-prices/>.
- [Hou62] P. V. C. Hough. *Method and Means for Recognizing Complex Patterns*. US 3069654. Originating Research Org. not identified, 18. Dez. 1962. URL: <http://www.osti.gov/scitech/biblio/4746348>.
- [Lun14] Jan Lunze. *Regelungstechnik 1*. 10. Aufl. Bochum, Germany: Springer Berlin Heidelberg, 2014. ISBN: 978-3-642-53909-1.
- [Nä15] Daniel Nägele. "A Demonstrator for Networked Control Systems". In: *IRE Transactions on Information Theory* (Sep. 2015), S. 46.
- [PDIWS] Prof. Dr.-Ing. M. Maurer Prof. Dr.-Ing. W. Schumacher. *Grundlagen der Regelungstechnik*. URL: https://www.ifr.ing.tu-bs.de/static/files/lehre/vorlesungen/gdr/Skript_GdR.pdf.
- [Pia] Raspberry Pi. *Raspberry Pi 2*. URL: <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>.
- [Pib] Raspberry Pi. *Raspberry Pi 2 real-time Kernel*. URL: <https://github.com/emlid/linux-rt-rpi>.
- [Pic] Raspberry Pi. *Raspberry Pi 3*. URL: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.
- [Pid] Raspberry Pi. *Raspberry Pi Cam 8Megapixel*. URL: <https://www.raspberrypi.org/blog/new-8-megapixel-camera-board-sale-25/>.
- [Pla] PlayStation. *PlayStation Eye, A Little More Info...* URL: <http://blog.us.playstation.com/2007/10/10/playstation-eye-a-little-more-info/>.

- [Ros15] Guido van Rossum. *Python*. 2015. URL: <https://www.python.org/>.
- [Shu] ShumaTech. *BOSSA*. URL: <http://www.shumatech.com/web/products/bossa>.
- [Son] Sony. *Sony IMX219PQ*. URL: http://www.sony.net/Products/SC-HP/new_pro/april_2014/imx219_e.html.
- [Str] Christopher E. Strangio. *The RS232 STANDARD - A Tutorial with Signal Names and Definitions*. URL: http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html.
- [GCC87] GCC GCC. *GCC, GNU Compiler Foundation - ArduinoIDE*. 1987. URL: <https://gcc.gnu.org/>.
- [LLC] Arduino LLC. *Arduino*. URL: <http://www.arduino.cc>.
- [LLC15a] Arduino LLC. *Arduino - ArduinoBoardDue*. 2015. URL: <http://www.arduino.cc/en/Main/ArduinoBoardDue>.
- [LLC15b] Arduino LLC. *Arduino - ArduinoBoardDueSchematics*. 2015. URL: <https://www.arduino.cc/en/uploads/Main/arduino-Due-schematic.pdf>.
- [LLC15c] Arduino LLC. *Arduino IDE - ArduinoIDE*. 2015. URL: http://arduino.cc/download_handler.php?f=/arduino-1.6.6-linux64.tar.xz.

Anhang

A. Makefile

```
1 SRC += main.c
2
3 OUTPUT = main
4
5 TTY_DEVICE_PORT = $(shell ./detect_serial_port.sh)
6
7 ASF_ROOT = /opt/asf-standalone-archive-3.29.0.41/xdk-asf
8     ↪ -3.29.0
9
10 INCLUDES += -I.
11
12 INCLUDES += -I$(ASF_ROOT)/common/boards
13 INCLUDES += -I$(ASF_ROOT)/common/services/clock
14 INCLUDES += -I$(ASF_ROOT)/common/services/gpio
15 INCLUDES += -I$(ASF_ROOT)/common/services/ioport
16 INCLUDES += -I$(ASF_ROOT)/common/utils
17 INCLUDES += -I$(ASF_ROOT)/sam/boards
18 INCLUDES += -I$(ASF_ROOT)/sam/boards/arduino_due_x
19 INCLUDES += -I$(ASF_ROOT)/sam/drivers/pio
20 INCLUDES += -I$(ASF_ROOT)/sam/drivers/pmc
21 INCLUDES += -I$(ASF_ROOT)/sam/drivers/pdc
22 INCLUDES += -I$(ASF_ROOT)/sam/drivers/uart
23 INCLUDES += -I$(ASF_ROOT)/sam/utils
24 INCLUDES += -I$(ASF_ROOT)/sam/utils/cmsis/sam3x/include
25 INCLUDES += -I$(ASF_ROOT)/sam/utils/cmsis/sam3x/source/
26     ↪ templates
27 INCLUDES += -I$(ASF_ROOT)/sam/utils/header_files
28 INCLUDES += -I$(ASF_ROOT)/sam/utils/preprocessor
29 INCLUDES += -I$(ASF_ROOT)/thirdparty/CMSIS/Include
30 INCLUDES += -I$(ASF_ROOT)/thirdparty/CMSIS/Lib/GCC
31
32 ASF_SRC += $(ASF_ROOT)/sam/utils/cmsis/sam3x/source/
33     ↪ templates/system_sam3x.c
34 ASF_SRC += $(ASF_ROOT)/sam/utils/cmsis/sam3x/source/
35     ↪ templates/gcc/startup_sam3x.c
36 ASF_SRC += $(ASF_ROOT)/sam/utils/cmsis/sam3x/source/
37     ↪ templates/exceptions.c
38 ASF_SRC += $(ASF_ROOT)/sam/drivers/pio/pio.c
39 ASF_SRC += $(ASF_ROOT)/sam/drivers/pio/pio_handler.c
40 ASF_SRC += $(ASF_ROOT)/sam/drivers/pmc/pmc.c
41 ASF_SRC += $(ASF_ROOT)/sam/drivers/pdc/pdc.c
42 ASF_SRC += $(ASF_ROOT)/sam/drivers/uart/uart.c
43 ASF_SRC += $(ASF_ROOT)/common/utils/interrupt/
44     ↪ interrupt_sam_nvic.c
```

```

39
40 CROSS = /usr/bin/arm-none-eabi-
41
42 OPTIMIZATION = -O0
43 #DEBUGCC = -g
44
45 CC = $(CROSS)gcc
46 LD = $(CROSS)g++
47 OBJCOPY = $(CROSS)objcopy
48 OBJDUMP = $(CROSS)objdump
49
50 LINKER_SCRIPT = $(ASF_ROOT)/sam/utils/linker_scripts/sam3x/
    ↪ sam3x8/gcc/flash.ld
51
52 CFLAGS += --param max-inline-insns-single=500 -mcpu=cortex-
    ↪ m3 -mthumb
53 CFLAGS += -fno-strict-aliasing -ffunction-sections -fdata-
    ↪ sections -std=gnu99
54 CFLAGS += -D BOARD=ARDUINO_DUE_X -D __SAM3X8E__ -D
    ↪ ARM_MATH_CM3=true
55 CFLAGS += -D printf=iprintf -D scanf=iscanf
56 CFLAGS += -Wall $(OPTIMIZATION) $(INCLUDES)
57 CFLAGS += $(DEBUGCC)
58
59 LDFLAGS += -L$(ASF_ROOT)/thirdparty/CMSIS/Lib/GCC
60 LDFLAGS += -Wl,--entry=Reset_Handler -Wl,--cref -mcpu=
    ↪ cortex-m3 -mthumb
61 LDFLAGS += -Wl,"-T$(LINKER_SCRIPT)" -Wl,--gc-sections
62 LDFLAGS += -Wl,-Map=$(OUTPUT).map,--cref
63 LDFLAGS += -Wl,--check-sections -Wl,--unresolved-symbols=
    ↪ report-all
64 LDFLAGS += -Wl,--warn-section-align
65 LDFLAGS += -Wl,--warn-unresolved-symbols
66 LDFLAGS += -larm_cortexM3l_math -lm
67
68 OBJ = $(SRC:.c=.o)
69 ASF_OBJ = $(ASF_SRC:.c=.o)
70
71 all: $(OUTPUT).bin
72
73 $(OUTPUT).elf: $(OBJ) $(ASF_OBJ)
74     $(LD) $(LDFLAGS) $(OBJ) $(ASF_OBJ) -o $@
75
76 $(OUTPUT).bin: $(OUTPUT).elf
77     $(OBJCOPY) -O binary $< $@
78
79 $(OUTPUT).s: all
80     $(OBJDUMP) -S $(OUTPUT).o > $(OUTPUT).s
81

```

```
82 | assamblercode: $(OUTPUT).s
83 |
84 | upload: $(OUTPUT).bin
85 |     stty -F $(TTY_DEVICE_PORT) 1200
86 |     bossac -e -w -v -b $(OUTPUT).bin -R
87 |
88 | upload-clean:
89 |     stty -F $(TTY_DEVICE_PORT) 1200
90 |     bossac -e -v
91 |
92 |
93 | serial_connect:
94 |     minicom -b 115200 -D $(TTY_DEVICE_PORT)
95 |
96 | .PHONY: clean
97 | clean:
98 |     rm -rf $(OBJ) $(ASF_OBJ) $(OUTPUT).elf $(OUTPUT).bin $(
    ↪ OUTPUT).map $(OUTPUT).s
```

B. LED Blinker Test Source Code

```
1  /**
2   * main.c
3   *
4   * test_led
5   *
6   * Copyright 2016 Institute of Parallel and Distributed
7     ↳ Systems (IPVS)
8   *
9   * Change history:
10  * - 2016-04-13, Benjamin Proelss (proelsbn@studi.
11     ↳ informatik.uni-stuttgart.de):
12  *   Initial implementation
13  */
14
15 // We compile for Arduino Due featuring a SAM3X8E
16 #include <system_sam3x.h>
17 #include <sam3x8e.h>
18 #include <pio.h>
19
20 /**
21  * Configure LED (used for showing status).
22  */
23 static void setup_led()
24 {
25     // enable digital IO
26     // PIO_PER: PIO enable register
27     PIOB->PIO_PER = PIO_PB27;
28     // configure pin as output
29     // PIO_OER: PIO output enable register
30     PIOB->PIO_OER = PIO_PB27;
31     // disable pull-up resistor
32     // PIO_PUDR: PIO pull-up disable register
33     PIOB->PIO_PUDR = PIO_PB27;
34     // Turn off LED by setting pin to high (active low)
35     // PIO_CODR: PIO clear output data register
36     PIOB->PIO_CODR = PIO_PB27;
37 }
38
39 int main()
40 {
41     // CMSIS: initialize system (clock, etc.)
42     SystemInit();
43
44     // Disable watchdog
45     WDT->WDT_MR = WDT_MR_WDDIS;
```

```

46  setup_led();
47
48  while (1) {
49      // Set LED-Pin
50      // PIO_SODR: Parallel Input/Output Set Output Data
51          ↪ Register
52      PIOB->PIO_SODR = PIO_PB27;
53      for(int i = 0; i <= 8400000; i++){
54          // Clear LED-Pin
55          // PIO_CODR: Parallel Input/Output Clear Output
56              ↪ Data Register
57      PIOB->PIO_CODR = PIO_PB27;
58      for(int i = 0; i <= 8400000; i++){
59
60  }

```

C. UART Test Source Code

```
1  /**
2   * main.c
3   *
4   * Copyright 2016 Institute of Parallel and Distributed
      ↳ Systems (IPVS)
5   *
6   * Change history:
7   * - 2016-04-13, Benjamin Proelss (proelsbn@studi.
      ↳ informatik.uni-stuttgart.de):
8   *   Initial implementation
9   */
10
11 // We compile for Arduino Due featuring a SAM3X8E
12 #include <system_sam3x.h>
13 #include <sam3x8e.h>
14 #include <pmc.h>
15 #include <pdc.h>
16 #include <uart.h>
17
18 /**
19  * Constant declaration
20  */
21 #define BUFFERSIZE 2
22
23 /**
24  * Global variable declaration
25  */
26 volatile uint8_t uart_receive_buffer[BUFFERSIZE];
27 volatile uint8_t uart_transmit_buffer[BUFFERSIZE];
28
29 /**
30  * This is a normal interrupt handler for the UART. It is
31  * executed if an interrupt is enabled in the UART
32  * interrupt enable register (UART_IER) and this interrupt
33  * is triggered.
34  */
35 void UART_Handler(void)
36 {
37     // After a read on the interrupt mask register (IMR)
38     // the interrupt bit is cleared. Save the interrupt
39     // to go into more causes
40     uint32_t uart_imr = UART->UART_IMR;
41
42     // RXBUFF      : RX Buffer full
43     // UART_RCR    : Receive counter register
44     // UART_RNCR   : Receive next counter register
45     // Flag is set when both UART_RCR and UART_RNCR
```

```

46 // reach zero
47 if (uart_imr & (UART_IMR_RXBUFF))
48 {
49     // Echo the incoming data
50     for (int i = 0; i < BUFFERSIZE; i++){
51         uart_transmit_buffer[i] = uart_receive_buffer[i
52             ↪ ];
53     }
54     // Reset the buffer counter to clear the RXBUFF
55     ↪ interrupt
56     UART->UART_RPR = (uint32_t) uart_receive_buffer;
57     UART->UART_RCR = BUFFERSIZE;
58     // Because setting the transmit buffer new
59     UART->UART_TPR = (uint32_t) uart_transmit_buffer;
60     UART->UART_TCR = BUFFERSIZE;
61 }
62 }
63 /**
64  * This function configure the power management controller
65  * (PMC) to enable all the required peripheral
66  * controllers which needed for this project.
67  */
68 static void setup_pmc(void)
69 {
70     // Set the PMC_PCER0 to 0xFFFFFFFFC and
71     // PMC_PCER1 to 0xFFFFFFFF
72     pmc_disable_all_periph_clk();
73
74     pmc_enable_periph_clk(ID_PMC);
75     pmc_enable_periph_clk(ID_PIOA);
76     pmc_enable_periph_clk(ID_DMAC);
77     pmc_enable_periph_clk(ID_UART);
78 }
79
80 /**
81  * This function configure the peripheral direct memory
82  * access controller (PDC) for the UART controller. Setting
83  * the buffer pointers and buffer sizes for the RX and TX
84  * connection. After that the receive/transmit is
85  * activated, with that the PDC check the transmit/receive
86  * holding register (THR/RHR) from the UART.
87  */
88 static void setup_pdc_uart(void)
89 {
90     // Initialize the buffer pointer at the end of the UART
91     // definition. There is an extra PDC_AREA from
92     // 0x100-0x124 offset (datasheet page 758)

```

```

93     // Init the transmit buffer
94     UART->UART_TPR = (uint32_t) uart_transmit_buffer;
95     UART->UART_TCR = BUFFERSIZE;
96
97     // Init the receive buffer
98     UART->UART_RPR = (uint32_t) uart_receive_buffer;
99     UART->UART_RCR = BUFFERSIZE;
100
101     // Activating the peripheral transmit an receive
102     UART->UART_PTCR = (UART_PTCR_TXTEN | UART_PTCR_RXTEN);
103 }
104
105 /**
106  * This function configure the peripheral input/output
107  * (PIO) and the UART itself. Set the TX,RX ports as
108  * input/output pins, setup the baudrate, disable the
109  * parity bit, disable all interrupts and activate only
110  * UART_IER_RXBUFF, UART_IER_ENDTX and enable at least
111  * the controller to transmit/receive data.
112  */
113 static void setup_uart(void)
114 {
115     uint32_t ul_sr;
116
117     // ==> Pin configuration
118     // Disable interrupts on Rx and Tx
119     PIOA->PIO_IDR = PIO_PA8A_URXD | PIO_PA9A_UTXD;
120
121     // Disable the PIO of the Rx and Tx pins so that the
122     // peripheral controller can use them
123     PIOA->PIO_PDR = PIO_PA8A_URXD | PIO_PA9A_UTXD;
124
125     // Read current peripheral AB select register and set
126     // the Rx and Tx pins to 0 (Peripheral A function)
127     ul_sr = PIOA->PIO_ABSR;
128     PIOA->PIO_ABSR &= ~(PIO_PA8A_URXD | PIO_PA9A_UTXD) &
129         ↪ ul_sr;
130
131     // Enable the pull up on the Rx and Tx pin
132     // PIO pull up enable register
133     PIOA->PIO_PUER = PIO_PA8A_URXD | PIO_PA9A_UTXD;
134
135     // ==> Actual uart configuration
136     // Reset and disable receiver & transmitter
137     // UART control register
138     UART->UART_CR = UART_CR_RSTRX | UART_CR_RSTTX;
139
140     // Set the baudrate to 115200

```



```

140 // 84000000 / 16 * x = Baudrate (write x into UART_BRGR
    ↪ )
141 UART->UART_BRGR = 45;
142
143 // No Parity
144 // UART mode register
145 UART->UART_MR = UART_MR_PAR_NO;
146
147 // Disable / Enable interrupts on end of receive
148 // UART interrupt disable register
149 UART->UART_IDR = 0xFFFFFFFF;
150 NVIC_EnableIRQ(UART_IRQn);
151 // UART interrupt enable register
152 UART->UART_IER = UART_IER_RXBUFF;
153
154 // Enable receiver and trasmitter
155 // UART control register
156 UART->UART_CR = UART_CR_RXEN | UART_CR_TXEN;
157 }
158
159 int main()
160 {
161 // CMSIS: initialize system (clock, etc.)
162 SystemInit();
163
164 // Disable watchdog
165 WDT->WDT_MR = WDT_MR_WDDIS;
166
167 // setup functions to initalize all used components
168 setup_pmc();
169 setup_pdc_uart();
170 setup_uart();
171
172 while (1) {
173 }
174
175 return 0;
176 }

```

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift