

DIPLOMARBEIT

Gesamtprojekt

FiMs

Entwicklung eines Flascheninhalt Messsystems

Daniel Lehner 5AHEL

Betreuer: Prof. Dipl.-Ing Siegbert Schrempf

Tobias Offenhuber 5AHEL


Daniel Markovic 5AHEL

ausgeführt im Schuljahr 2018/19

Abgabevermerk:

Datum: 05.04.2019

übernommen von:

	HÖHERE TECHNISCHE BUNDESLEHR- UND VERSUCHSANSTALT Salzburg
	Elektronik und Technische Informatik

Eidesstattliche Erklärung

Ich erkläre hiermit eidesstattlich, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.


Salzburg, am TT.MM.JJJJ

Verfasserinnen / Verfasser:

Daniel Lehner

Daniel Markovic

Tobias Offenhuber

	HÖHERE TECHNISCHE BUNDESLEHR- UND VERSUCHSANSTALT Salzburg
	Elektronik und Technische Informatik

DIPLOMARBEIT


DOKUMENTATION


Namen der Verfasserinnen / Verfasser	Daniel Lehner, Tobias Offenhuber, Daniel Markovic
Jahrgang Schuljahr	5AHEL 2018/2019
Thema der Diplomarbeit	FiMs – Entwicklung eines Flascheninhalt Messsystems
Kooperationspartner	Tanzschule Seifert

Aufgabenstellung	Der Barbetrieb der Tanzschule Seifert verwendet derzeit eine manuelle Füllstandmessung mithilfe flaschenspezifischer Lineale. Dies ist äußerst mühsam und fehleranfällig. Mit Hilfe von FiMs soll es möglich sein, den Füllstand der Flaschen einfach und automatisiert zu ermitteln.
------------------	---

Realisierung	Mithilfe eines Raspberry Pi wird das Gewicht der Flaschen, über das selbst konstruierte Wägemodul, erfasst. Anhand des Gewichtes und einem softwareinternen Profilingssystem wird der Inhalt der Flaschen bestimmt. Steuer- und konfigurierbar ist das Gerät über ein Touchscreen-Display.
--------------	--

Ergebnisse	Die Ermittlung und Verarbeitung der Flaschenfüllstände soll, mithilfe des Wägemoduls und des Embedded Systems, automatisiert werden. Neue Flaschen im Sortiment sollen jederzeit in das System aufnehmbar sein. FiMs soll den alltäglichen Betrieb der Tanzschulbar für Barkeeper und die Buchhaltung maßgeblich vereinfachen.
------------	--

	HÖHERE TECHNISCHE BUNDESLEHR- UND VERSUCHSANSTALT Salzburg
	Elektronik und Technische Informatik

Typische Grafik, Foto etc.	
----------------------------	--

Teilnahme an Wettbewerben, Auszeichnungen	Keine Teilnahme bei Wettbewerben.
--	-----------------------------------

Möglichkeiten der Einsicht- nahme in die Arbeit	Schulbibliothek der HTBLuVA Salzburg
--	--------------------------------------

Approbation (Datum / Unterschrift)	Prüferin / Prüfer	Direktorin / Direktor Abteilungsvorständin / Abteilungsvorstand
---------------------------------------	-------------------	--

DIPLOMA THESIS


Documentation

Author(s)	Daniel Lehner, Daniel Markovic, Tobias Offenhuber,
Form Academic year	5AHEL 2018/2019
Topic	FiMs – Development of a bottle capacity measurement system
Co-operation Partners	Dancing school Niki Seifert

Assignment of Tasks	The bar of the Seifert dance school has currently only manual means of measurement to measure bottle levels using bottle-specific rulers. This is vastly time consuming and moreover error prone. FiMs aims to provide easy and automated measurements of bottle levels with high accuracy.
---------------------	---

Realization	The bottle weight is measured with the help of a self-built scale-module for the Raspberry Pi. Bottle levels are calculated using the weight and our software-internal profiling system. The system is supposed to be navigable and configurable with a touchscreen display.
-------------	--

Results	The determination and processing of the bottle levels should be automated, with the help of the scale-module and the embedded system. New bottles in the assortment should be addable to the system at any point. FiMs is made to vastly facilitate the daily operation for barkeepers and accounting.
---------	--

<p>Illustrative Graph, Photo</p>		
<p>Participation in Competitions Awards</p>	<p>No participation.</p>	
<p>Accessibility of Diploma Thesis</p>	<p>Library of the HTBLuVA-Salzburg Online database of the HTBLuVA-Salzburg</p>	
<p>Approval (Date / Sign)</p>	<p>Examiner</p>	<p>Head of College Head of Department</p>

Vorwort

In der vorliegenden Diplomarbeit beschäftigen wir uns mit der Entwicklung eines Flascheninhalt Messsystems für die Bar der Tanzschule Seifert. Dieses Projekt lässt sich grob in drei Teile gliedern: Entwicklung und Bau des Wägemoduls und des Gehäuses, die Entwicklung der Softwarestruktur und die Entwicklung der Benutzeroberfläche.

Die Projektidee kam von Daniel Lehner. Die Tanzschule Seifert suchte schon seit längerem nach einem Weg, die bisher manuelle Flaschenfüllstandmessung zu automatisieren. Die bisherige Messmethodik mithilfe der flaschenspezifischen Linealen war äußerst mühsam und fehleranfällig.

Nahe liegende Lösungen wären unter anderem sehr teuer, so wie zum Beispiel eine komplett neue Zapfanlage. Jedoch kam Daniel, als passioniertem Tänzer und Barkeeper, die Idee, die Flaschenfüllstandsmessung mithilfe des Gewichts zu ermitteln. Diese Daten werden anschließend an die Buchhaltung vermittelt.

Die individuellen Aufgabenstellungen wurden jeweils anhand der Teilgebiete des Projekts zugeteilt. Daniel Markovic beschäftigte sich mit dem Wägemodul und dem Gehäuse, Daniel Lehner mit der Softwareentwicklung und Tobias Offenhuber übernahm sowohl Teile der Softwareentwicklung als auch die Entwicklung der Benutzeroberfläche.

Durch unser Projekt haben wir als Team gelernt, wie wichtig gute Kommunikation ist. Im Laufe der Projektarbeit kam es wiederholt zu Engpässen und Hürden, die es zu überwinden galt. Hierbei handelte es sich in den meisten Fällen um Fehleinschätzungen in der zeitlichen Einteilung. Jedoch gab es auch technische Hürden, wie die Toleranzeinhaltung, welche erfolgreich überwunden wurde. Dies ist einerseits der guten Kommunikation innerhalb des Teams zu verdanken, andererseits möchten wir uns hiermit für die großartige Unterstützung unseres Projektbetreuers bedanken, welcher uns häufig den nötigen Denkanstoß gab und uns in Zeiten des Motivationstiefs mit entsprechenden Moralpredigten erneut motiviert hat. Die Realisierung von FiMs hat uns nicht nur beim Verständnis technischer Problemlösungen geholfen, es ist auch definitiv eine unschätzbare Bereicherung unserer persönlichen Erfahrung. An dieser Stelle bedanken wir uns besonders bei unserem Projektbetreuer Prof. Dipl.-Ing. Siegbert Schrempf für sein Engagement und die wichtige Hilfestellungen in der Entwicklungsphase des Projekts. Ebenso bedanken wir uns bei Herr Prof. Mag. Dr. Hoffman, welcher uns maßgebliche Hilfe bei der Entwicklung der Software gab. Zu guter Letzt danken wir Herrn FL. Weiser BEd, ohne den wir unser Gehäuse wohl kaum anfertigen hätten können.

Vorwort	7
1. Systemspezifikation	13
1.1. Zielbestimmungen	13
1.1.1. Musskriterien	13
1.1.2. Wunschkriterien	13
1.1.3. Abgrenzungskriterien	14
1.2. Produkteinsatz	14
1.2.1. Anwendungsbereiche	14
1.2.2. Zielgruppen	14
1.2.3. Betriebsbedingungen	14
1.3. Produktumgebung	14
1.3.1. Software	14
1.3.2. Hardware	14
1.4. Produktfunktionen	14
1.4.1. Produktdaten	15
1.4.2. Produktleistungen	15
1.4.3. Benutzungsoberfläche	15
1.4.4. Dialogstruktur	15
1.5. Qualitätsbestimmungen	16
1.6. Globale Testszenarien und Testfälle	16
1.7. Entwicklungsumgebung	16
1.7.1. Software	16
1.7.2. Hardware	16
1.7.3. Orgware	17
2. Projektmanagement	17
2.1. Überblick	17
2.2. GANTT – Diagramme	18
2.3. Softwareengineering – Rapid Prototyping	21

2.3.1.	Einführung.....	21
2.3.2.	Phasen.....	21
2.3.2.1.	Analyse	21
2.3.2.2.	Design – Entwurf.....	21
2.3.2.3.	Codierung	21
2.3.2.4.	Integration	21
2.3.2.5.	Installation.....	22
2.3.2.6.	Wartung	22
2.3.3.	Struktur des Modelles	22
2.3.3.1.	Vorteile.....	22
2.3.3.2.	Nachteile	22
2.3.3.3.	Praxisanwendung	22
3.	Grundlagen und Methoden.....	23
3.1.	Überblick	24
3.2.	Wäge System.....	25
3.2.1.	Allgemeine Information	25
3.2.2.	Gewichtsermittlung.....	25
3.2.3.	Messprinzipien	25
3.2.3.1	Saitenwaage.....	26
3.2.3.2	Prinzip der elektromagnetischen Kraftkompensation.....	27
3.2.3.3.	Dehnungsmessstreifen	28
3.2.3.3.1.	Überblick	28
3.2.3.3.2.	Aufbau.....	28
3.2.3.3.3.	Temperaturabhängigkeit	29
3.2.3.3.4.	Brückenschaltung.....	29
3.2.3.4.	Analog – Digital Converter (ADC)	30
3.2.3.4.1.	Überblick	30
3.2.3.4.2.	Wägeverfahren.....	30

3.2.3.4.3. Abtast- und Halteglied	31
3.2.3.5. HX711	32
3.2.3.5.1. Überblick	32
3.2.3.5.2. Aufbau	32
3.2.3.5.3. Funktionsprinzip	33
3.2.3.6. Serielle Schnittstellen	34
3.2.3.7. Serial Peripheral Interface (SPI)	34
Überblick	34
Eigenschaften	35
Protokollablauf	35
3.3. Embedded Systems und Java-Programmierung	37
3.2.4. Überblick	37
3.2.5. Betriebssysteme	37
3.2.5.1. Raspberry Pi	39
3.2.5.2. Raspberry-Pi: verwendete Funktionen und Hardware	40
Betriebssystemauswahl	42
3.2.6. Distributed Systems	42
3.2.6.1. Funktionsweise der Speicherverwaltung bei Distributed Systems	42
3.2.6.2. FIFO-Prinzip	43
3.2.6.3. Pipes/Pipelining	44
3.2.7. Wahl der Programmiersprache	45
3.2.8. FiMs GUI	46
3.2.8.1. Navigation durch die Softwarestruktur der GUI	49
3.2.8.2. Mehrere GUI Fenster mit nur einer Klasse verwalten	51
3.4. IPC – Interprozesskommunikation	55
3.4.3. Überblick	55
3.4.4. Prozesse und Threads	55
3.4.4.3. Prozess	55

3.4.4.4.	Threads.....	56
3.4.5.	FiMs - Softwarestruktur	56
3.4.5.3.	Auswertung des Wägemoduls	56
3.4.5.3.1.	Pythonscript	56
3.4.5.4.	Standardströme	59
3.4.5.5.	Vermittlung der Daten	60
3.4.5.6.	Synchronisation	62
3.4.5.7.	Werkzeuge der Synchronisation	64
3.4.5.8.	Problemstellungen der Synchronisation.....	65
3.4.5.9.	Verarbeitung der Daten.....	67
3.4.5.10.	Filehandling mit Apache POI.....	68
3.4.5.11.	Komponenten der API.....	68
3.4.5.12.	Profilingssystem.....	70
4.	Ergebnisse	73
4.2.	Hardware.....	73
4.2.3.	Wägezelle	73
4.2.4.	Konstruktion Waage.....	73
4.2.5.	HX711 Ansteuerung (AVR)	74
4.2.6.	Bitfolgenanalyse.....	76
4.2.7.	Toleranzprüfung.....	77
4.2.7.3.	Langzeitmessung.....	77
4.2.7.4.	Werte-Kennlinie	78
4.2.8.	Gehäuse	79
4.3.	Software	80
4.3.3.	Auslesen des Pythonscripts	80
4.3.4.	Auslesen der Wägedaten in Java.....	80
4.3.5.	Profilerstellung	81
4.3.6.	Betrieb.....	81

5.	Kostenaufstellung	83
6.	Schlusswort	84
7.	Glossar	85
8.	Quellen- und Literaturverzeichnis	86
9.	Verzeichnis der Abbildungen, Tabellen und Abkürzungen	88
10.	Begleitprotokoll gemäß § 9 Abs. 2 PrO	90
10.2.	Begleitprotokoll Markovic	90
10.3.	Begleitprotokoll Offenhuber	91
10.4.	Begleitprotokoll Lehner	92
11.	Anhang	93
11.2.	Konstruktionspläne	93

1. Systemspezifikation

1.1. Zielbestimmungen

Ziel ist es, das Gewicht mithilfe eines Wägemoduls zu wiegen, dieses mithilfe eines Raspberry Pis zu verarbeiten, um sie anschließend an einen lokalen Server zu vermitteln.

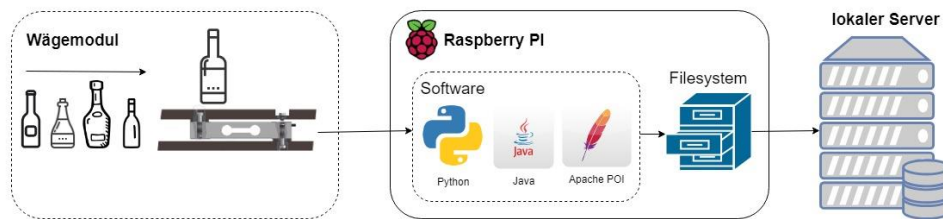


Abbildung 1-1: Systemübersicht

1.1.1. Musskriterien

Embedded System:

- Erweiterbares Flaschenprofilsystem
- LCD Touchscreen
- ADC Schnittstelle zum Embedded System

Mechanik:

- Wasserdichte Gehäusekonstruktion
- Wägemechanik entsprechend erforderlicher Toleranzen
- Kalibrierfähigkeit
- Wartungsfreundlichkeit

Benutzeroberfläche:

- Benutzerfreundliche GUI über Touchscreen Bildschirm

1.1.2. Wunschkriterien

Embedded System:

- Erkennung der diversen Flaschen mithilfe eines Barcodescanners

Mechanik:

- Modernes Gehäusedesign
- Langlebigkeit des Systems

Benutzeroberfläche:

- Smartphone Applikation
- Ausführliche Bedienungsanleitung

1.1.3. Abgrenzungskriterien

- Keine staatliche Eichung
- Kein Buchhaltungssystem
- Keine Wartungsgarantie

1.2. Produkteinsatz

1.2.1. Anwendungsbereiche

Mit FiMs kann die Bar-Abrechnung in der Tanzschule Seifert massiv erleichtert werden. Zeit und Geld werden eingespart.

1.2.2. Zielgruppen

Das Projekt ist speziell an die Bedürfnisse der Bar der Tanzschule Seifert angepasst.

1.2.3. Betriebsbedingungen

Es ist wichtig, einen stabilen Untergrund zu gewährleisten, um Messabweichungen zu vermeiden.

1.3. Produktumgebung

1.3.1. Software

Es wird die Software „FiMs“ benötigt.

1.3.2. Hardware

Es wird das fertig zusammengebaute Gehäuse inklusive Raspberry PI, Waage und Touchscreen benötigt, um das Produkt im vollen Umfang nützen zu können.

1.4. Produktfunktionen

/F0001/ Gewichtsmessung:

Das Gewicht einer Flasche sollte ermittelt und in einem System gespeichert werden.

/F0002/ Profilsystem:

Es soll möglich sein, flaschenspezifische Daten in das System aufzunehmen.

/F0003/ Bedienfreundlichkeit:

Die GUI sollte eine einfache Steuerung des Systems gewährleisten.

/F0004/ Toleranzeinhaltung:

Das System soll entsprechende Toleranzeinhaltung garantieren. Die Genauigkeit der Messung beträgt $\pm 2\%$.

/F0006/ Barbetrieb:

Das System soll im Barbetrieb einwandfrei die Flaschenfüllstände ermitteln und diese an einen lokalen Server vermitteln.

1.4.1. Produktdaten**/D0001/ Profile**

Die Profile werden in einer .xls-Datei archiviert.

/D0002/ Abrechnung

Eine Vorlage der Abrechnung steht als .xls-Datei zu Verfügung.

1.4.2. Produktleistungen**/L0001/ Toleranz**

Die Toleranz wird in allen Bereichen eingehalten.

/L0002/ Echtzeit

Die Daten des Wägemoduls werden in Echtzeit übermittelt und ausgewertet.

1.4.3. Benutzungsoberfläche

Das User Interface soll einfach zu bedienen sein, und es soll vor allem möglich sein, es zeitsparend verwenden zu können. Es existiert keine User-Hierarchie, somit kann jeder Angestellte die grundlegenden Einstellungen der Software ändern.

1.4.4. Dialogstruktur

Über das Hauptmenü der FiMs GUI ist es möglich, alle Funktionen zu nutzen wie Einstellungen zu ändern, Flaschenstände zu messen oder ein neues Flaschenprofil hinzufügen.

1.5. Qualitätsbestimmungen

Die Software ist speziell auf das Raspberry PI zugeschnitten und ist nicht portierbar.

Die folgende Tabelle zeigt die Priorität der Qualitätsanforderungen.

	Sehr wichtig	Wichtig	Weniger wichtig	unwichtig
Robustheit		X		
Wasserdichtigkeit	X			
Zuverlässigkeit	X			
Korrektheit	X			
Benutzerfreundlichkeit		X		
Effizienz	X			
Portierbarkeit				X
Kompatibilität				X

1.6. Globale Testszenarien und Testfälle

/T0001/ Hardware-Software Kommunikation

Die Waage muss jederzeit initialisiert werden können. Die Wägedaten müssen jederzeit entsprechend ausgelesen und verarbeitet werden können.

/T0003/ Profilmanagement

Neue und bereits existierende Profile müssen jederzeit abrufbar sein.

/T0004/ Ausgabe

Die Ausgabe der Wägedaten in Liter muss gewährleistet sein.

1.7. Entwicklungsumgebung

1.7.1. Software

Als Entwicklungsumgebung wurde Eclipse 2018-09 gewählt. Die Software wurde sowohl in Python als auch, vorwiegend, in Java geschrieben. Die grafische Darstellung erfolgt mit Java Swing.

1.7.2. Hardware

Als Kernstück der Hardware wurde ein Raspberry PI Model 3 B gewählt. Dieses ist über einen Raspberry PI Touchscreen steuerbar. Das Raspberry PI kommuniziert mit einem HX711 ADC, welcher die Daten der Waage konvertiert.

1.7.3. Orgware

Als Orgware wurde der Cloudservice Mega.nz gewählt. Die Software wurde mithilfe der Version-Control Plattform gitlab verwaltet. Zur externen Kontrolle des Raspberry PI wurde MobaX-Term verwendet.

2. Projektmanagement

2.1. Überblick

Im Großen wurde folgende Aufgabenverteilung vorgenommen:

Daniel Markovic:

- Entwicklung der Wägemechanik inklusive der ADC Elektronik
- Gehäusebau

Tobias Offenhuber:

- Entwicklung der GUI
- Auslesen des Wägemoduls

Daniel Lehner:

- Projektmanagement
- Auslesen des Wägemoduls
- Profilinsystem

2.2. GANTT – Diagramme

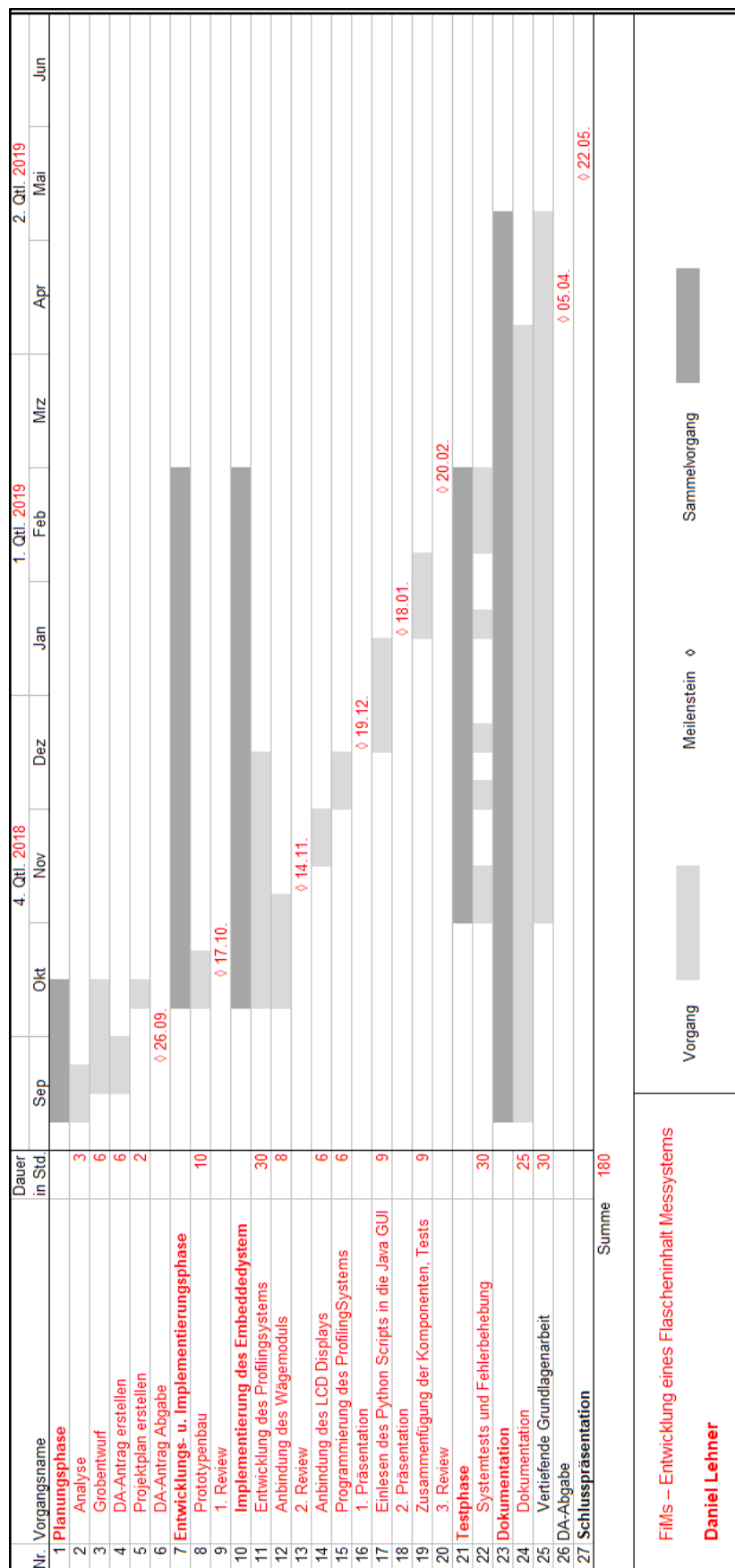


Abbildung 2-1: Gantt-Diagramm Lehner

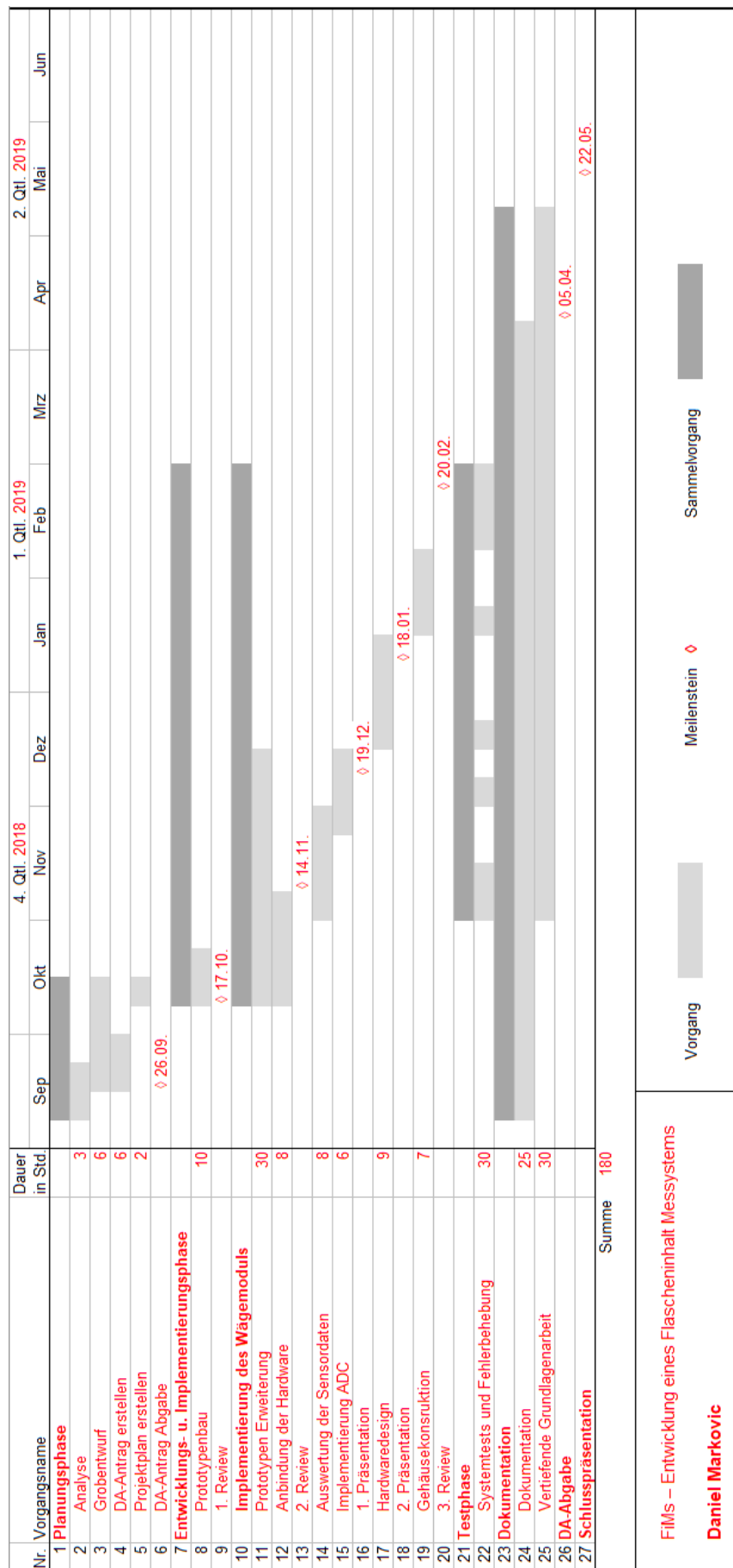


Abbildung 2-2: Gantt-Diagramm Markovic

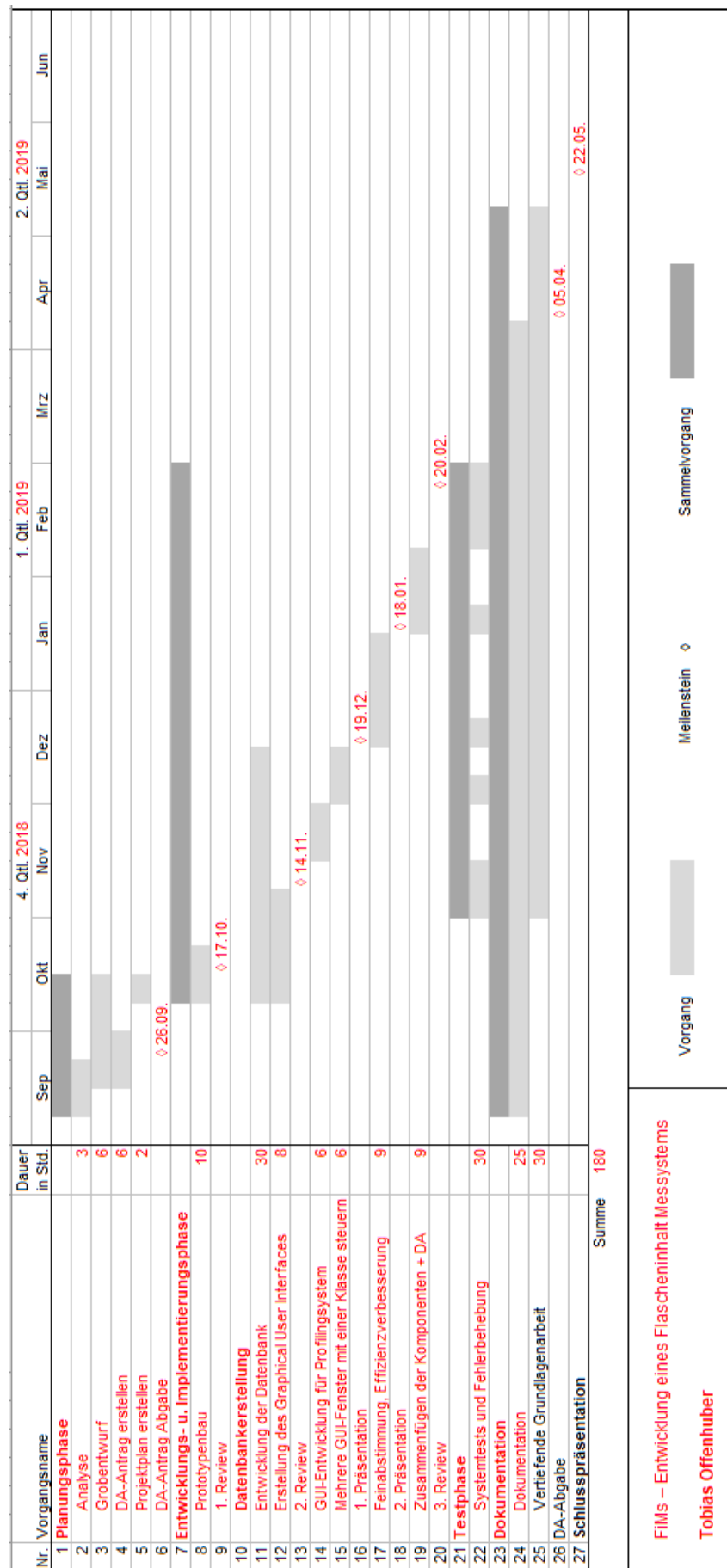


Abbildung 2-3: Gantt-Diagramm Offenhuber

2.3. Softwareengineering – Rapid Prototyping

2.3.1. Einführung

Rapid Prototyping hat ein sehr flexibles Vorgehensmodell/Phasenmodell.

Phasen können unvollständig durchlaufen werden, beziehungsweise übersprungen werden.

2.3.2. Phasen

2.3.2.1. Analyse

In der Analysephase wurde das grundlegende Konzept geplant. Zuerst wurden Anforderungen und Funktionalitäten bestimmt, welche für das Projekt benötigt werden. Auf die Frage: „Was soll das System tun?“, mussten Antworten gefunden werden. Schnittstellen nach außen wurden festgelegt- Die Leistungsfähigkeit wurde ebenfalls durchdacht, und es wurde schlussendlich festgestellt, dass das Raspberry Pi ausreichend Rechenleistung zur Verfügung stellt.

2.3.2.2. Design – Entwurf

In dieser Phase wurde über das Softwarekonzept nachgedacht. Eine Blockdiagrammstellung, Datenstrukturen wurden festgelegt, Bedienungsszenarios erstellt, Lösungsalgorithmen geplant (Profiling System) und die Datenkommunikation zwischen den einzelnen Komponenten definiert (SPI).

2.3.2.3. Codierung

Es konnte nach den vorherigen zwei Phasen mit der Codierung begonnen werden. Die Planung wurde in konkrete Programmiersprache umgesetzt und getestet.

2.3.2.4. Integration

In dieser Phase wurden alle Einzelkomponenten zusammengebaut und wiederum getestet.

2.3.2.5. *Installation*

In dieser Phase wird das fertige Produkt vor Ort, also der Tanzschule, aufgebaut und getestet. Das Ziel ist es, das Programmverhalten im konkreten Einzelfall unter entsprechenden Rahmenbedingungen zu testen.

Zu dieser Phase ist es noch nicht gekommen, der Plan ist, dies alsbald durchzuführen, sobald die letzten Kleinigkeiten optimiert wurden.

2.3.2.6. *Wartung*

Es ist sicher zu stellen, dass das System auf lange Sicht gut funktioniert. Im Falle von FiMs ist dies grundsätzlich nicht notwendig, da beim Projektentwurf auf Wartungsfreiheit geachtet wurde. Trotzdem kann es natürlich passieren, dass neue Features oder Fehlerbehebungen benötigt werden.

2.3.3. Struktur des Modelles

2.3.3.1. *Vorteile*

Diese Methode unterstützt die schrittweise Programmentwicklung, der Prototyp wird laufend verbessert/erneuert.

2.3.3.2. *Nachteile*

In Softwareprojekten kann auf zusätzliche Kundenanforderungen wenig flexibel reagiert werden.

2.3.3.3. *Praxisanwendung*

In der Praxis wird diese Entwicklungsmethode besonders gerne bei Projekten verwendet, die zum Beispiel bei der agilen Entwicklung von User-Interfaces für Kunden hilfreich ist.

3. Grundlagen und Methoden

FiMs verfügt sowohl über Hardware- als auch über Softwarekomponenten. Die einzelnen Teilaufgaben der Projektmitglieder überschneiden sich hierbei in vielen Aspekten. Das Projekt lässt sich jedoch grob in drei Teilbereiche gliedern, wie in der folgenden Abbildung (Abb. 3-1 Systemübersicht Hardware / Software) zu erkennen ist. Hierbei ist ebenso zu sehen, dass ein Großteil der Software in Kooperation entwickelt wurde.

Daniel Markovic (rot eingerahmt) kümmerte sich um die Hardware. Die Hardware besteht einerseits aus dem Wägemodul, welches aus der Waage selbst und dem ADC HX711 besteht. Ebenso konstruierte Daniel das Gehäuse für das Projekt.

Daniel Lehner (grün eingerahmt) entwickelte in erster Linie das Profilingssystem. Hierbei handelt es sich um ein Archivierungssystem, welches für jedes Produkt ein eigenes Profil anlegt, damit mit dessen Werten der entsprechende Füllstand ermittelt werden kann. Das Profilingssystem arbeitet mit .xls-Dateien. In diesem Teilbereich kommt es in vielen Aspekten zur sogenannten Interprozesskommunikation.

Tobias Offenhuber (blau eingerahmt) beschäftigte sich mit dem Embedded System. Zusätzlich programmierte er die GUI, welche zur einfachen Bedienung von FiMs dient. Ebenso entwickelte Tobias, zusammen mit Daniel Lehner, das System für das Auslesen der Wägedaten.

3.1. Überblick

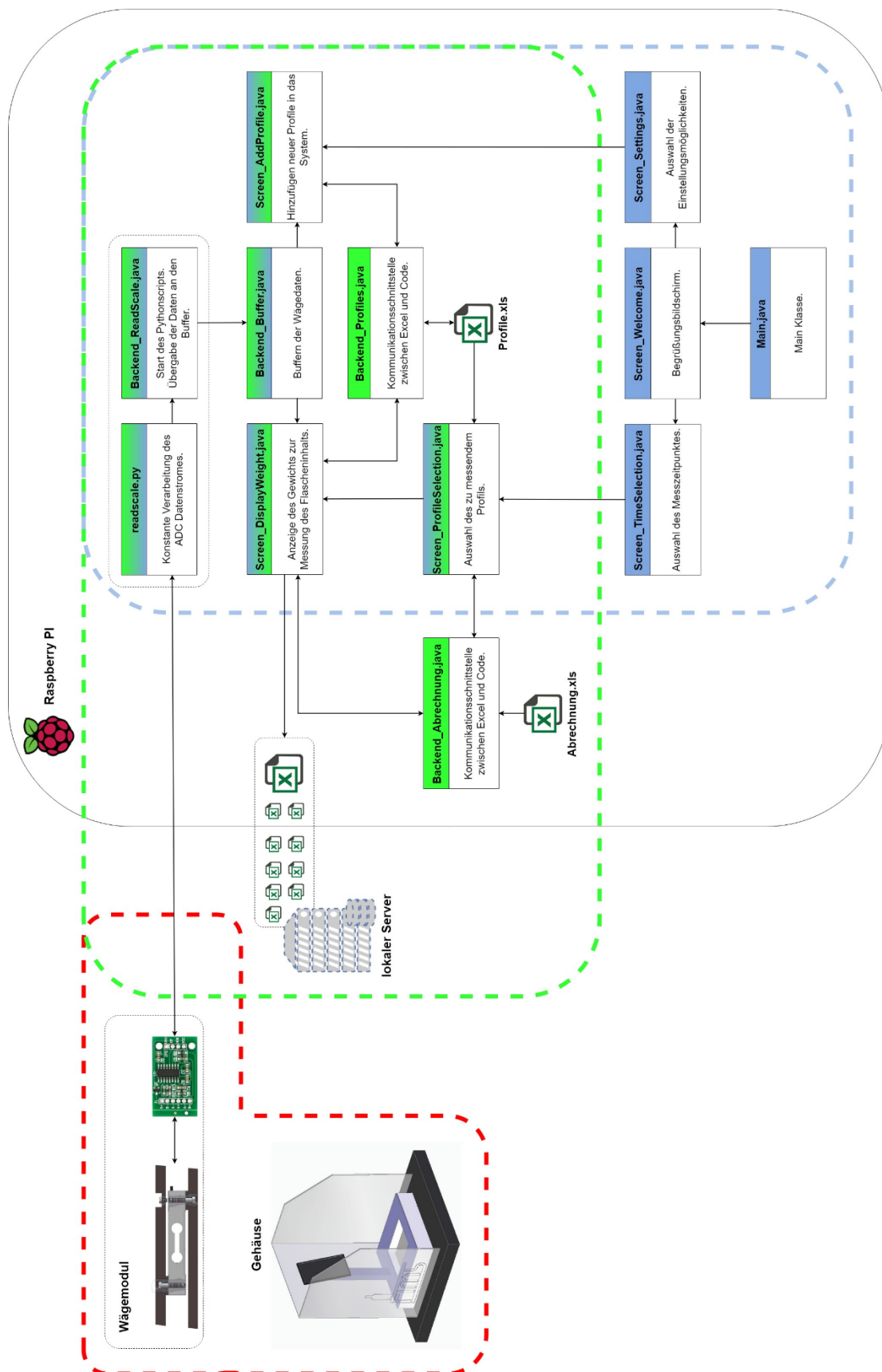


Abbildung 3-1 Systemübersicht Hardware / Software

3.2. Wäge System

3.2.1. Allgemeine Information

Wägesysteme dienen zur Bestimmung der Masse des zu wiegenden Objektes. Die Messung der Gewichtskraft erfolgt meist direkt, aber auch durch Vergleiche von bestimmten Massen.

$$\text{Masse in kg} = \frac{\text{Gewichtskraft in N}}{\text{Erdbeschleunigung in } \frac{\text{m}}{\text{s}^2}}$$

Abbildung 3-2: Zusammenhang von Masse und Erdbeschleunigung (Quelle: <https://de.wikipedia.org/wiki/Waage>)

Dass die Messgrößen beziehungsweise auch die Messabweichungen ihren festgelegten Grenzen bleiben, ist eine wichtige Eigenschaft einer Waage. [1]

3.2.2. Gewichtsermittlung

Die Füllstände werden über das Gewicht der Flaschen inklusive deren Inhalt bestimmt. Deshalb wurde für das Projekt entschieden, ein eigenes Wäge System zu entwickeln. Um die dafür passende Methode zur Bestimmung der Masse zu finden, war es notwendig, sich mit mehreren Messprinzipien auseinander zu setzen und die einzelnen Eigenschaften zu analysieren.

Anhand der Recherche ist der Entschluss gefallen, Dehnmessstreifen für die Gewichtsermittlung einzusetzen. Grund dafür waren die erfüllten Kriterien im Bereich der Kosten und Toleranz eines Systems mit Dehnmessstreifen.

3.2.3. Messprinzipien

Wie bereits erwähnt, wurden mehr Messprinzipien in Betracht gezogen. Da für das Projekt keine mechanischen Waagen in Frage kamen, lag das Augenmerk auf elektronische Waagen. Dabei wurde zwischen drei Sensor-Prinzipien unterschieden. [1]

1. Waagen mit Sensorelementen, wie zum Beispiel bei einer Saitenwaage
2. Elektromagnetische Kraftkompensationswaagen
3. Waagen mit Wägezellen, die mit Dehnmessstreifen arbeiten

3.2.3.1 Saitenwaage

Eine aus Metall bestehende, gespannte Saite wird durch elektromagnetische Anregung in Schwingungen versetzt. Ein quasidigitaler Sensor diente dabei als Ideevorlage. Zudem ist ein permanentes Magnetfeld dafür erforderlich.

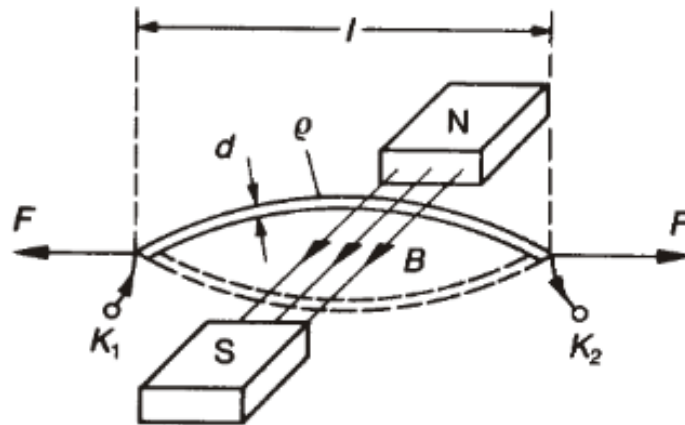


Abbildung 3-3: Prinzip der Schwingsaiten – Aufnehmer
(Quelle: PHYSIKALISCHE MESSTECHNIK B)

Der Strom wird mit der Eigenfrequenz über die Klemmen K1 und K2 (siehe Abb. 3-3) geleitet, über eine induzierte Spannung an den Klemmen ist es möglich, die Amplitude der Saiten-Schwingungen zu erfassen. Daraus resultiert ein Oszillator, dessen Schwingfrequenz gleich der Eigenfrequenz der Schwingsaite ist. Die Formel für die Grundfrequenz der Saite lautet:

$$f = \frac{1}{l \cdot d} \sqrt{\frac{F}{\pi \cdot \rho}}$$

Abbildung 3-4: Formel der Grundfrequenz (Quelle: PHYSIKALISCHE MESSTECHNIK B)

mit F = angreifende Kraft, ρ = Dichte des Saitenmaterials, l = Länge der Saite.

Ein Einsatzgebiet für diese Art von Waagen wäre beispielsweise die Messung der Drehzahl. Frequenz und Messkraft weisen im Zusammenhang eine Nichtlinearität auf. Die unbekannte Gewichtskraft kann man aus der Differenzfrequenz bestimmen. [2]

3.2.3.2 Prinzip der elektromagnetischen Kraftkompensation

Mikrowaagen, welche meistens in Laboratorien verwendet werden, funktionieren nach dem Prinzip der elektromagnetischen Kraftkompensation. Dabei wird eine Gegenkraft in einem Permanentmagnet durch eine Spule erzeugt, welche dem Gewicht des Objektes entspricht, welches sich auf der Waagschale befindet. Das Gleichgewicht wird mit Hilfe dieser Kraft gehalten.

Bewältigt wird dies durch einen Regelverstärker und einen Lagesensor, die notwendig sind, um den Gewichtszustand einzuhalten.

Es ist möglich, mittels eines Hebelsystems das Gleichgewicht auf der Waagschale mit größerer Kraft durch eine kleinere Gegenkraft zu halten.

An einem Messwiderstand wird der Spulenstrom als Spannungsabfall gemessen und von einem Analog-Digital-Wandler weiterverarbeitet und angezeigt. [3]

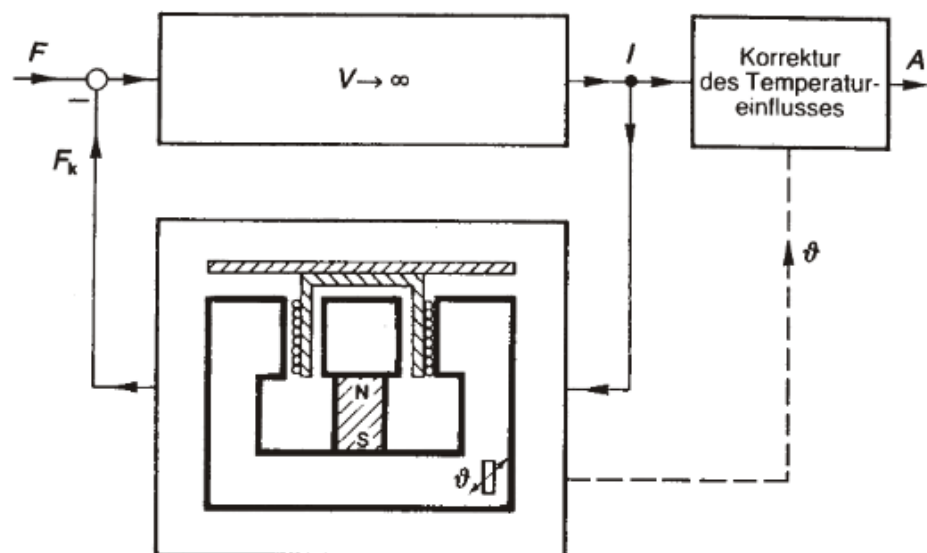


Abbildung 3-5: Elektrodynamische Kompensations-Waage (Quelle: PHYSIKALISCHE MESSTECHNIK B)

Mikrowaagen in Form einer Magnetschwebewaage werden seit einigen Jahren auch gebaut. Die Probe, die gewogen werden soll, wird dabei von der Messzelle vollständig getrennt. Die Probe befindet sich an einem Permanentmagneten, der von einem an der Waage befestigten Elektromagneten in einem freien Schwebezustand gehalten wird.

Das Gewicht des Probenkörpers wird mit dieser Magnetschwebekupplung kontaktlos aus dem Messraum auf die Mikrowaage übertragen. Die Messung beruht hierbei auf dem Prinzip der elektromagnetischen Kraftkompensation. [3]

3.2.3.3. Dehnungsmessstreifen

3.2.3.3.1. Überblick

Mittels Dehnungsmessstreifen (DMS) ist es möglich, sowohl dehnende als auch stauende Verformungen zu erfassen. Bereits bei geringen Längenänderungen ändert sich ihr elektrischer Widerstand. Dazu werden die Sensoren auf die Bauteile geklebt, die sich unter Belastung minimal verformen.

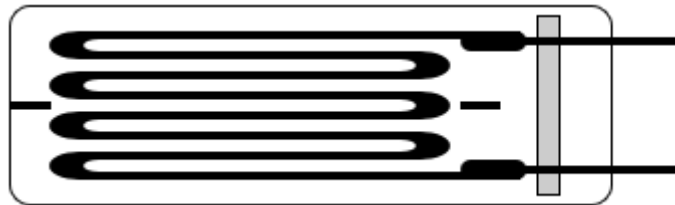


Abbildung 3-6: Folien-Dehnungsmessstreifen (Quelle: Dehnungsmessstreifen - Wikipedia)

Bei Messungen mit DMS werden vor allem Brückenschaltungen wie die Viertel-, Halb- und die Vollbrücke eingesetzt. [4]

3.2.3.3.2. Aufbau

Typische Dehnungsmessstreifen sind sogenannte Folien-DMS, d.h. die aus Widerstandsdraht bestehende Messgitterfolie wird auf einen dünnen Kunststoffträger kaschiert, ausgeätzt und zusätzlich mit Anschlüssen versehen. Die zweite dünne Folie auf der Oberseite dient dazu, das Messgitter mechanisch zu schützen. Werden mehrere DMS auf einem Träger platziert, spricht man von Rosetten-DMS.

Es stehen sowohl Messgitter aus Metall und aus Halbleitern (Silizium) zur Verfügung. Die Halbleiter-DMS beruhen auf dem piezoresistiven Effekt. Anders als bei den Metall-DMS spielt die Widerstandsänderung durch Längen- bzw. Querschnittsänderung eine Nebenrolle. Dadurch, dass der piezoresistive Effekt stärker ausgeprägt ist, sind diese DMS wesentlich empfindlicher. [4]

Bezeichnung	Zusammensetzung	k-Faktor
Konstantan	54 % Cu 45 % Ni 1 % Mn	2,05
Nichrome V	80 % Ni 20 % Cr	2,2
Chromol C	65 % Ni 20 % Fe 15 % Cr	2,5
Platin-Wolfram	92 % Pt 8 % W	4,0
Platin	100 % Pt	6,0
Silizium	100 % p-Typ Si: B (Bor in ppm-Bereich)	+80...+190
Silizium	100 % n-Typ Si: P (Phosphor in ppm-Bereich)	-25...-100

Abbildung 3-7: Werkstoffe für Metall-DMS und Halbleiter-DMS (Quelle: Dehnmessstreifen - Wikipedia)

3.2.3.3. Temperaturabhängigkeit

Durch die starke Temperaturabhängigkeit werden Halbleiter-DMS eher selten verwendet. Bei Auftreten von Temperaturfehlern schafft die Brückenschaltung durch Kompensation Abhilfe. Außerdem weichen die Effekte einzelner Brückenzweige auf demselben Chip weniger voneinander ab, als wenn vier Halbleiter-DMS geklebt und verschaltet werden würden.

In der Praxis findet man allerdings ein ganz anderes Problem vor: unabhängig vom Material, auf dem gemessen wird, kommt es bei steigender Temperatur zu einer Auslenkung. Diese Auslenkung entspricht bei anhaltendem Auftreten keiner Belastung. Unter Verwendung einer DMS-Vollbrücke wird eine vollständige Kompensation erzielt. Ebenso ist dies durch alternative Maßnahmen, bei denen die Temperaturdehnung mit einem DMS am unbelasteten Bauteil gleichen Werkstoffs zusätzlich gemessen wird, möglich.[4]

3.2.3.4. Brückenschaltung

Eine Widerstands-Messbrücke ist aus zwei parallelgeschalteten Spannungsteilern aufgebaut. Dabei wird die Verbindung zwischen zwei Potenzialen als Brücke bezeichnet. Die Gesamtspannung teilt sich an den Widerständen auf.

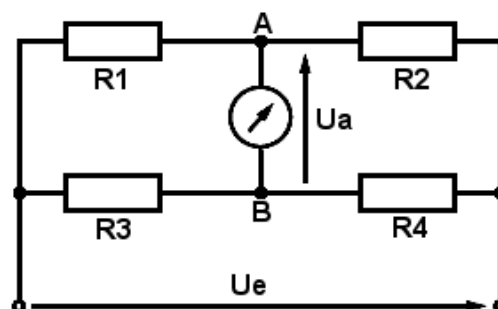


Abbildung 3-8: Brückenschaltung (Quelle: Elektronische Waage)

Wenn das Verhältnis der Spannungsteiler (abhängig von den Widerständen) gleich groß ist, dann liegt an beiden Punkten das gleiche Potential an. Herrscht allerdings ein Potentialunterschied, so fließt ein Strom von Punkt A nach Punkt B bzw. umgekehrt.

Bei Verwendung eines Microcontroller ist es von Vorteil, eine Spannungsmessung durchzuführen, und keine Strommessung, da die Eingänge des ADC extra für Spannungen konzipiert sind. [5]

Folgende Gleichung veranschaulicht eine Brückenschaltung durch den Spannungsteiler:

$$U_a = U_e * \left(\frac{R_4}{R_3 + R_4} - \frac{R_2}{R_1 + R_2} \right)$$

3.2.3.4. Analog – Digital Converter (ADC)

3.2.3.4.1. Überblick

Ein ADC ist ein Bauelement oder Teil eines Bauelements, der analoge Eingangssignale in einen digitalen Datenstrom umwandelt und diesen weiterverarbeitet oder speichert.

Diese arbeiten mit einem DAC (Digital Analog Umsetzer), der einen Vergleichswert (z.B. U_v) jedes Mal neu aufbaut.

Die Eingrenzung des Eingangssignals erfolgt mittels Intervallschachtelung. Einfache sukzessive Approximation setzt dabei pro Schritt ein Bit um (z.B. Wägeverfahren). [6]

3.2.3.4.2. Wägeverfahren

Bei diesem Verfahren werden in einem Datenspeicher (SAR) zunächst alle Bits auf null gesetzt. Danach erfolgt die Ermittlung aller Bits des Digitalwerts beginnend beim höchstwertigen Bit (MSB) bis zum niederwertigsten (LSB). Das Steuerwerk setzt während des Arbeitsprozesses die Bits probeweise auf eins, der DAC erzeugt die dem aktuellen Digitalwert entsprechende Vergleichsspannung. Diese Vergleichsspannung wird durch einen Komparator mit der Eingangsspannung verglichen und befiehlt dem Steuerwerk, die Bits wieder auf null zurückzusetzen, sobald die Vergleichsspannung größer ist als die Eingangsspannung. Ansonsten bleibt das in Arbeit befindliche Bit auf eins.

Nach der Einstellung des niederwertigsten Bits ist $U_e - U_v$ kleiner als der kleinste einstellbare Schritt.

Die Eingangsspannung muss während der Umsetzung konstant bleiben, da sonst die niederwertigen Bits auf Grundlage der festgestellten, aber nicht mehr gültigen höherwertigen Bits, gewonnen würden. Deshalb wird vor den Eingang eine Abtast-Halte-Schaltung (Sample and Hold) geschaltet. Für die Genauigkeit an jedem benötigt der ADC jeweils eine Taktzyklus Umsetzungszeit. [6]

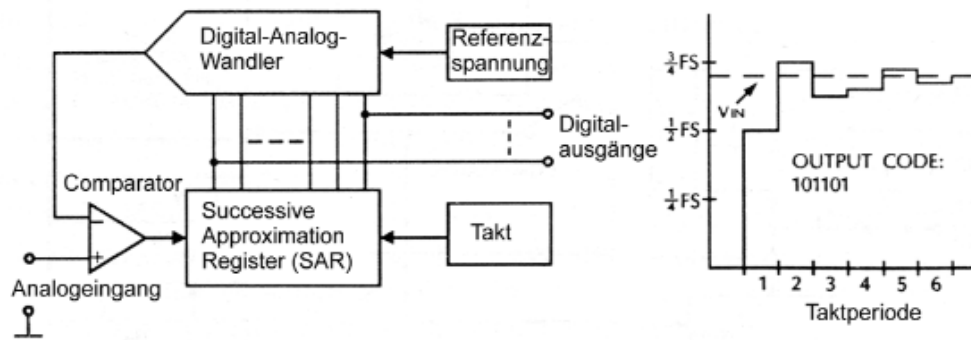


Abbildung 3-9: ADC nach dem Verfahren der sukzessiven Approximation (Quelle: Elektronik für embedded Systems)

3.2.3.4.3. Abtast- und Halteglied

Auch Sample-and-Hold-Schaltung genannt, ist eine elektronische Vorrichtung, die es erlaubt, analoge Spannungswerte kurzzeitig auf einem definierten Wert zu halten. Eine solche Schaltung hat einen Steuereingang, einen Signaleingang und einen Signalausgang. Mit dem Steuereingang wird zwischen Abtast- und Haltephase umgeschaltet; so wird bestimmt, ob das Ausgangssignal dem Eingangssignal folgt (Abtastphase) oder festgehalten wird (Haltephase).

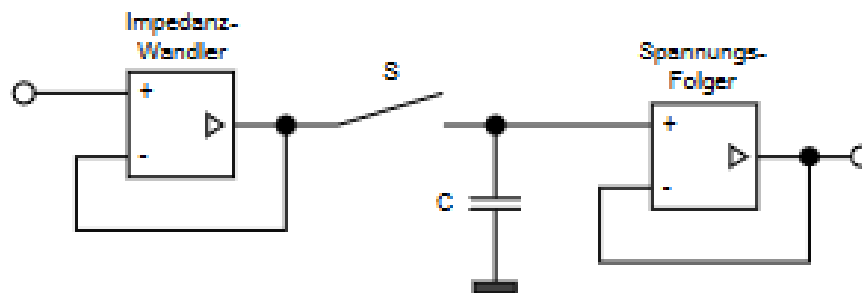


Abbildung 3-10: Schematische Anordnung eines Abtast- und Halteglieds (Quelle: Sample-and-Hold-Schaltung - Wikipedia)

Im eingeschalteten Zustand folgt die Ausgangsspannung der Sample-and-Hold-Schaltung der Eingangsspannung.

Im ausgeschalteten Zustand hält die Sample-and-Hold-Schaltung mit nullter Ordnung den Wert, den die Eingangsspannung im Augenblick des Abschaltens hatte.

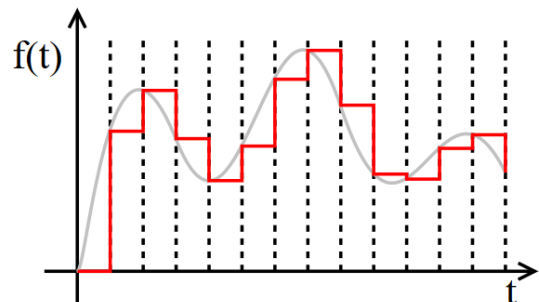


Abbildung 3-11: Signalverlauf (Quelle: S&H - Wikipedia)

3.2.3.5. HX711

3.2.3.5.1. Überblick

Der HX711 enthält nicht nur die analogen Brückenverstärker, sondern auch einen hochauflösenden 24-Bit-A/D-Wandler mit einem SPI-ähnlichen seriellen Interface zum Mikrocontroller. Was für die Auswertung des Wägesensors sehr vorteilhaft ist.

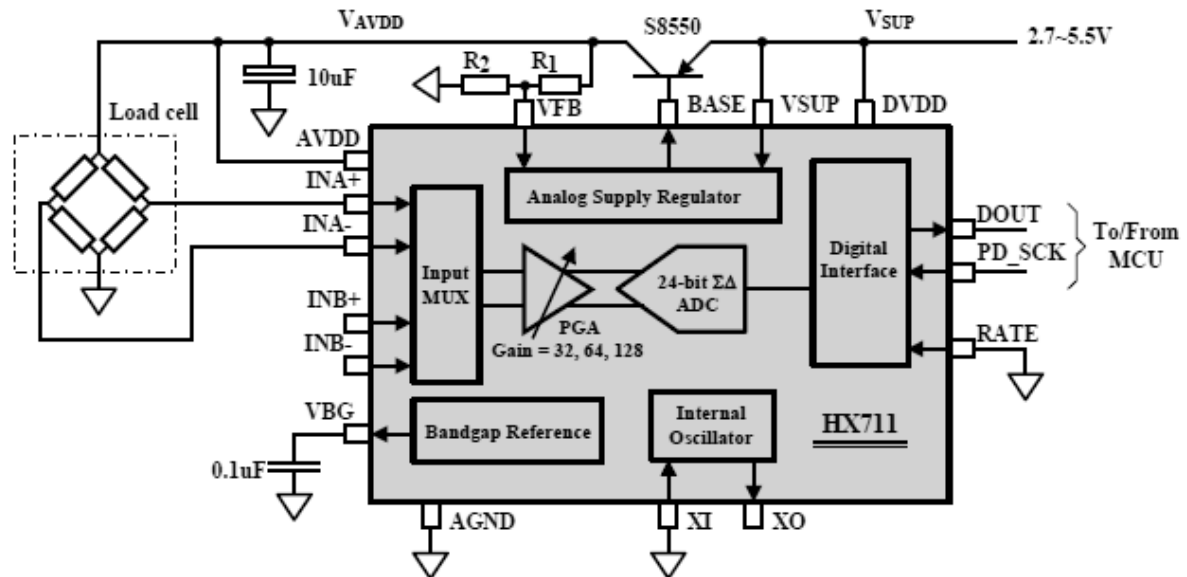


Abbildung 3-12: Beschaltung HX711 (Quelle: Datenblatt)

3.2.3.5.2. Aufbau

Das HX711-Board akzeptiert fünf Leitungen aus der Wägezelle. Diese Farben am Board entsprechen der herkömmlichen Farbcodierung der Wägezellen, wobei der rote, schwarze, grüne und weiße Draht von dem Dehnungsmessstreifen stammen und die gelbe Litze eine optionale Masseleitung ist, die nicht mit den Dehnungsmessstreifen verbunden ist, aber Verbindung zum Metallkörper der Wägezelle hat.

Auf der Controllerseite werden die Signale VDD (Stromversorgung des Digitalteils), VCC (Stromversorgung des Analogteils), DAT (Daten), CLK (Takt) und GND (Masse) angeschlossen. VCC, die analoge Spannungsversorgung und VDD, die digitale Versorgungsspannung, können in vielen Fällen miteinander verbunden werden.

Über einen Jumper am RATE-Eingang des HX711 kann die Datenrate eingestellt werden. Ist der Anschluss mit GND verbunden, beträgt die Datenrate 10 Messungen pro Sekunde, ist der RATE-Pin über einen Pullup-Widerstand mit VCC verbunden, beträgt die Datenrate 80 Messungen pro Sekunde. Das Signal ist dann aber wesentlich stärker mit Rauschen überlagert und somit nicht ganz so genau.[5]

3.2.3.5.3. Funktionsprinzip

Der Input-Multiplexer steuert entweder Kanal A oder B an den PGA (Programmable Gain Amplifier). Kanal A hat eine variable Verstärkung wo man zwischen einer Verstärkung von 128 oder 64 auswählen kann. Dies sorgt für eine differenzielle Eingangsspannung von $\pm 20\text{mV}$ oder $\pm 40\text{mV}$, bei einer angelegten Spannung von 5V am Versorgungspin AVDD.

Kanal B jedoch hat nur eine voreingestellte Verstärkung von 32 und produziert dementsprechend eine differenzielle Eingangsspannung von $\pm 80\text{mV}$.

Der Clock-input ist flexibel beim HX711. Der Takt kann entweder von einer externen Taktquelle, einem Schwingkristall oder dem am Board verbauten Oszillator kommen, welcher keine weiteren Komponenten benötigt.

Im Projektfall bekommt der Messverstärker vom RaspberryPi den Takt. Beim Benutzen einer externen Taktquelle, ist der Ausgangs Daten Rate proportional zur Taktfrequenz. Die 24-Bit Daten die der HX711 liefert, werden im zweier-Komplement angezeigt. Sollte das Input Signal außerhalb der 24-Bit Reichweite gehen dann gehen die Daten auf ihr Minimum (800000h) oder Maximum (7FFFFFFh), bis das Signal wieder im 24-Bit Bereich liegt.

PD_SCK Pulses	Input channel	Gain
25	A	128
26	B	32
27	A	64

Abbildung 3-13: Tabelle zum Auswählen des Kanals und der Verstärkung (Quelle: Datasheet HX711)

Für die Ansteuerung des Messverstärkers werden die Pins PD_SCK und DOUT verwendet.

DOUT ist konstant auf HIGH, solange die Ausgangsdaten nicht bereit zum empfangen sind. In dieser Zeit sollte der serielle Takt auf LOW sein. Wenn DOUT auf LOW gehen sollte, wird damit indiziert das die Daten bereit zum empfangen sind.

Danach müssen 25-27 Taktimpulse (abhängig vom ausgewählten Kanal und der Verstärkung) am Pin PD_SCK angelegt werden. Das sorgt dafür das die Daten geschoben werden. Jeder Taktzyklus entspricht einem Bit, das herausgeschoben wird. Angefangen mit dem MSB, bis alle restlichen Bits rausgeschoben worden sind.

Der 25. Taktimpuls stellt dabei den Pin DOUT auf HIGH und beendet somit die Bereitschaft, Daten zu empfangen.

Input Kanal und Verstärkung werden an der Menge der Taktimpulse bestimmt (siehe Abb. 12) und man sollte dabei beachten nicht mehr als 27 und weniger als 25 Impulse anzulegen, sonst kommt es zu einem seriellen Kommunikationsfehler. [8]

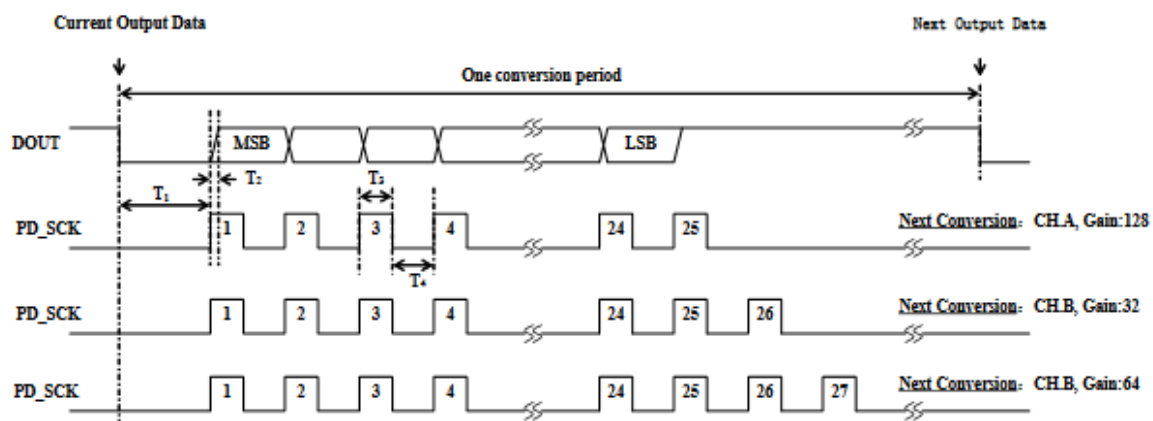


Abbildung 3-14: Bitfolge HX711 während der Datenübertragung (Quelle: Datenblatt HX711)

3.2.3.6. Serielle Schnittstellen

Die serielle Schnittstelle dient dem Datenaustausch zwischen Computern und Peripheriegeräten. Bei einer seriellen Datenübertragung werden die Bits nacheinander über eine Leitung übertragen.[9]

3.2.3.7. Serial Peripheral Interface (SPI)

Überblick

Das Serial Peripheral Interface (kurz SPI) Bus-System stellt einen gering spezifizierten Standard für einen synchronen seriellen Datenbus dar, mit dem man digitale Schaltungen nach dem Master-Slave-Prinzip miteinander verbinden kann.[10]

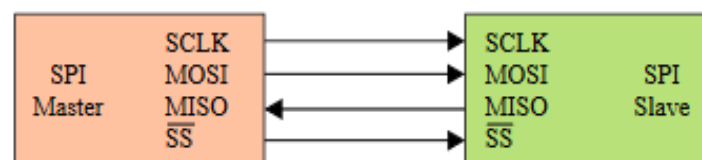


Abbildung 3-15: Einfacher SPI-Bus mit Master und Slave (Quelle: SPI - Wikipedia)

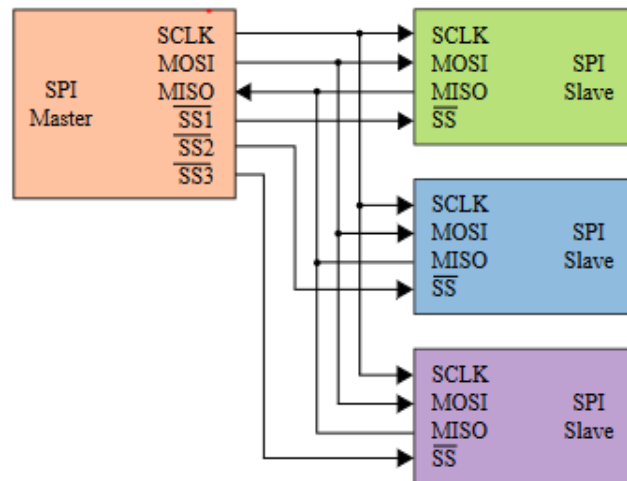


Abbildung 3-16: SPI-Sternverbindung (Quelle: SPI-Wikipedia)

Eigenschaften

Jeder Teilnehmer ist miteinander durch die Pins SCK, MISO und MOSI miteinander verbunden. Zusätzlich lässt sich einstellen, mit welcher Taktflanke eingelesen und ausgegeben wird, ebenso ist es möglich die Wortlänge einzulesen. Der Benutzer kann mitentscheiden, ob ein MSB oder LSB geschickt wird. Taktfrequenzen bis in den MHz-Bereich sind zulässig. [10]

Dadurch, dass der SPI im Vollduplex arbeitet, ist es möglich, Daten bidirektional zu übertragen. [11]

Protokollablauf

Die Anzahl der Teilnehmer an dem Bus wird durch die Slave-Select-Leitungen bestimmt, mit einem Master, der seinerseits das Clock-Signal an SCK erzeugt.

An den Bus können so viele Teilnehmer angeschlossen werden, wie Slave-Select-Leitungen vorhanden sind, einem Master, der seinerseits das Clock-Signal an SCK erzeugt. Mit der Leitung „Slave Select“ wird festgelegt, mit welcher Slave kommuniziert werden soll. Wird die Leitung gegen Masse gezogen, ist der Slave aktiv, er legt seine Daten im Takt von SCK an MISO. Ein Wort wird vom Master zum Slave transportiert und umgekehrt.

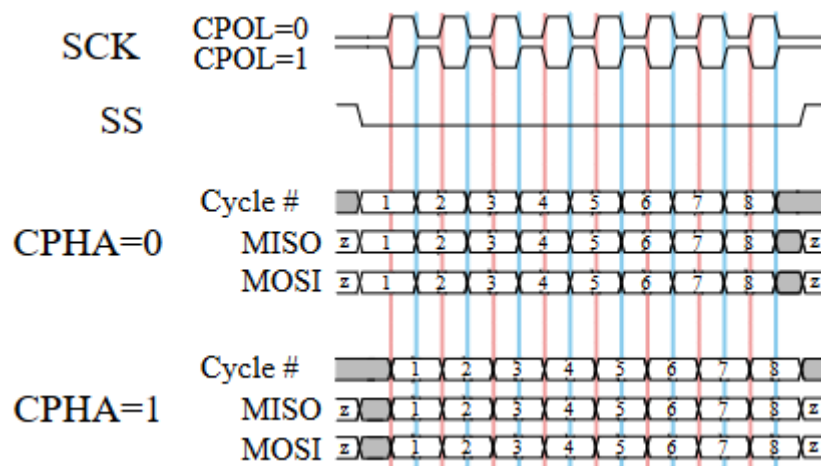


Abbildung 3-17: Datenübertragung SPI (Quelle: SPI - Wikipedia)

Diese werden durch die Parameter "Clock Polarity" (CPOL) und "Clock Phase" (CPHA) festgelegt. Bei CPOL=0 ist der Clock Idle Low, bei CPOL=1 ist der Clock Idle High. CPHA gibt nun an, bei welcher Flanke die Daten übernommen werden sollen. Bei CPHA=0 werden sie bei der ersten steigenden Flanke übernommen, nachdem SS auf Low gezogen wurde, bei CPHA=1 bei der zweiten fallenden.

Somit werden die Daten bei CPOL=0 und CPHA=0 mit der steigenden Flanke übernommen. Bei CPHA=1 wäre eine fallende Flanke. Bei CPOL=1 ist es genau andersherum, bei CPHA=0 fallende Flanke und bei CPHA=1 steigende Flanke.[10]

3.3. Embedded Systems und Java-Programmierung

3.2.4. Überblick

Eingebettete Systeme (embedded systems) sind Computersysteme, die aus Hardware und Software bestehen und die in komplexe technische Umgebungen eingebettet sind. Diese Umgebungen sind meist maschinelle Systeme, in denen das eingebettete System mit Interaktion durch einen Benutzer arbeitet oder auch vollautomatisch (autonom) agiert. Die eingebetteten Systeme übernehmen komplexe Steuerungs-, Regelungs- und Datenverarbeitungsaufgaben für bzw. in diesen technischen Systemen. [12]

Es gibt verschiedene Arten von Embedded Systems. Je nach Einsatzgebiet kann das Embedded System auf verschiedene Aspekte ressourceneffizient optimiert werden. Stromverbrauch, Stückpreis, Hardwareanforderungen, Echtzeitanforderungen, Betriebssicherheit und Integration sind einige von vielen Aspekten, die für die Entwurfsentscheidung eine tragende Rolle spielen.

Es gibt Embedded Systems, die für Multifunktionszwecke optimiert sind, wie z.B. das „RaspberryPI“. Die Entscheidung fiel auf das Raspberry Pi, da Aspekte wie Energieverbrauch, Rechenleistung und Stückpreis für FiMs eher unbedeutend sind und das Raspberry Pi sehr hohe Marktakzeptanz besitzt.

3.2.5. Betriebssysteme

Ein modernes Rechensystem besteht aus einem oder mehreren Prozessoren, Arbeitsspeicher, Platten, Druckern, einer Tastatur, einem Bildschirm, Netzwerkschnittstellen und anderen Ein-/Ausgabegeräten. Alles in allem handelt es sich um ein komplexes System. Die Erstellung von Programmen, die diese Komponenten verwalten und sie korrekt benutzen, ist eine extrem schwierige Aufgabe, selbst dann, wenn man von optimalen Lösungen absieht. Aus diesem Grund wurden Computer mit einer zusätzlichen **Softwareschicht** ausgestattet, die man **Betriebssystem** nennt. Dessen Aufgabe ist es, vorhandene Geräte zu verwalten und Benutzerprogrammen eine einfache Schnittstelle zur Hardware zur Verfügung zu stellen. [13]

Es gibt verschiedene Arten von Betriebssystemen:

Server-Betriebssysteme, die hauptsächlich auf Servern laufen, kümmern sich in erster Linie um die Verteilung von Hardware und Softwareressourcen an die Benutzer.

Multiprozessor-Betriebssysteme bestehen aus mehreren zusammengeschalteten Prozessoren. Dieses Betriebssystem wird verwendet, wenn eine sehr starke Rechenleistung gefordert ist.

Echtzeit-Betriebssysteme sind darauf spezialisiert, eine kurze und maximale Antwortzeit zu garantieren. Dies ist besonders wichtig in der Medizintechnik oder bei maschinellen Fertigungs-Anlagen.

Natürlich gibt es auch Betriebssysteme, die auf Personal Computer spezialisiert sind, welche für das Projekt FiMs von großer Bedeutung sind. Hierbei spielt es die größte Rolle, eine passende Benutzerschnittstelle zwischen Mensch und Maschine bereitzustellen.

Ob bei Embedded Systems ein Betriebssystem verwendet wird, kommt ebenfalls auf das Anwendungsgebiet an. Bei einfacheren Systemen kommt meist kein Betriebssystem zum Einsatz. Wenn doch ein Betriebssystem verwendet wird, ist dieses meist sehr speziell auf den Anwendungsfall konfiguriert, d.h. nur die wichtigsten und auch unbedingt benötigten Funktionen werden implementiert und somit können wichtige Aspekte wie Rechenzeit oder Stückpreis optimiert werden. Bei FiMs wurde das Raspberry Pi mit dem Betriebssystem Raspian gewählt, da es geeignete Funktionen zur Verfügung stellt. Details dazu in folgenden Kapiteln.

3.2.5.1. Raspberry Pi

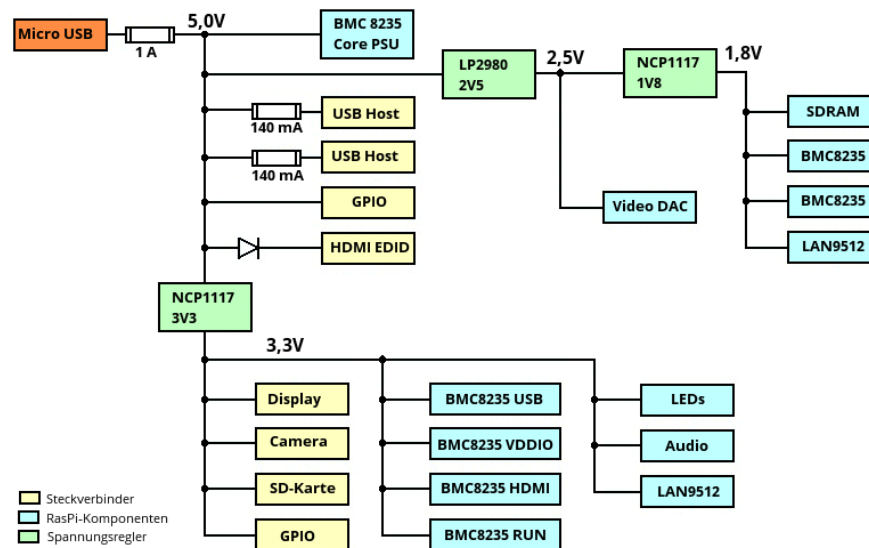


Abbildung 3.3-18: Raspberry-Pi Blockschaltbild

(Quelle: http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_Intro.html)

Der Raspberry Pi ist ein Einplatinencomputer, der von der britischen Raspberry Pi Foundation entwickelt wurde. Der Rechner enthält ein Ein-Chip-System von Broadcom mit einem ARM-Mikroprozessor, die Grundfläche der Platine entspricht etwa den Abmessungen einer Kreditkarte. [14]



Abbildung 3-19: RaspberryPi (Quelle: www.wikipedia.org)

3.2.5.2. Raspberry-Pi: verwendete Funktionen und Hardware

Da das Raspberry Pi vor allem dafür entwickelt wurde, der „neuen“ Generation die Softwareentwicklung bzw. Hardwarekenntnisse näherzubringen, existieren dafür sehr viele Zubehör- und Softwareangebote sowie Dokumentationen. Es ist also auch für unerfahrene Softwareentwickler möglich, eigene Projekte zu realisieren. Aus diesem Grund ist das Raspberry Pi auch ausgewählt worden für FiMs. Bei FiMs geht es nicht vorrangig um Rechenleistung, es sollen einfache Funktionen zuverlässig ausgeführt werden.

	Pin No.		
3.3V	1	2	5V
GPIO2	3	4	5V
GPIO3	5	6	GND
GPIO4	7	8	GPIO14
GND	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	GND
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
GND	25	26	GPIO7
DNC	27	28	DNC
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

Abbildung 3-20: Pinbelegung Raspberry Pi

(Quelle: www.raspberrypi.org)

Erwähnenswert sind die GPIO-Ports (engl. General Purpose Input/Output), welche für Allzweckeingaben und -ausgaben dienen. Abzüglich der Spannungsversorgungen und DNC's (engl. Do Not Connect) ergeben sich 26 verwendbare GPIO-Pins.

Über diese Schnittstelle können mit simplen Codestrukturen LED's, Displays, Sensoren oder andere Geräte angesteuert werden.

Der ADC HX711 (Analog Digital Converter) wird mithilfe von dem Python Script, welches die GPIO-Ports über SPI anhand SS ansteuert, ausgelesen. Die ausgelesenen Daten werden dann mithilfe des Python Scripts ausgewertet und dem Profilingssystem zur weiteren Verarbeitung übermittelt.

Serial Peripheral Interface: Verbindung vom Raspberry Pi und des HX711-Moduls:

GPIO Port	Direction	Usage
Pin 2 (5V)	Output	VCC
Pin 6 (GND)	Output	GND
Pin 29 (GPIO 5)	Input	DATA
Pin 31 (GPIO 6)	Input	SCK

Tabelle 3-3-1: Pinbelegung Raspberry Pi

Ansteuerung des LCD Touchscreen Displays:

Der verwendete Touchscreen Bildschirm ist ein Produkt von Raspberry Pi und somit auf das verwendete Embedded System zugeschnitten. Der Bildschirm kommuniziert mithilfe eines Flachbandkabels über die serielle Schnittstelle DSI. Bei der Verbindung ist es wichtig, auf die richtige Polung zu achten, da das Flachbandkabel an beiden Enden an jeweils nur einer Seite abisoliert ist. Es gilt das Prinzip "Plug n Play" - Der Bildschirm muss lediglich angesteckt und versorgt werden. Das Betriebssystem stellt die Ansteuerungssoftware zur Verfügung.

Mithilfe dieses DSI-Kabels werden Touchscreenbefehle an das Raspberry Pi, beziehungsweise Bild-/Videodaten an das LCD-Display übertragen. Die Spannungs- und Stromversorgung wird mittels eines speziellen USB-Kabels realisiert, welches mit 2.5A betrieben wird. „Normale“ USB-Kabel können nur mit 0.5-1A betrieben werden. Mögliche Folgen der Benutzung eines Standard USB-Kabels wären Systemabstürze oder auffällig langsame Rechenzeiten. Das USB-Kabel kann entweder an das Raspberry Pi oder an das Touchscreen-Controllerboard angeschlossen werden. Wichtig hierbei ist nur, +5V und GND der beiden zu verbinden. Bei FiMs wird das Controller-Board angeschlossen und das Raspberry Pi davon versorgt.

DSI - Display Serial Interface

Die serielle Schnittstelle DSI ist ein speziell für LCD-Bildschirme ausgerichtetes serielles Interface. Um das DSI zu verwenden, muss man sich an ein spezielles Kommunikationsprotokoll halten, welches im Fall von FiMs bereits vom Raspberry Pi automatisch angewendet wird

Betriebssystemauswahl

Nun stellt sich die Frage, welches Betriebssystem man für das Raspberry Pi und somit für das Projekt verwenden sollte. Hier gibt es mehrere Möglichkeiten:

Für das Raspberry Pi sind mehrere Open-Source-Betriebssysteme verfügbar. Installiert werden sie entweder durch das Schreiben eines Speicherabbilds auf die SD-Karte oder seit dem 3. Juni 2013 auch mit der einfacher zu verwendenden Eigenentwicklung NOOBS-Installer (engl. Abk. für New Out Of Box software), deren Dateien nur auf die SD-Karte kopiert werden müssen. Mit BerryBoot gibt es einen ebenso einfach zu installierenden Bootloader, der es ermöglicht, mehrere Betriebssysteme auf einer Karte parallel zu installieren und wahlweise zu verwenden. Seit Version 1.3 ist dies auch mit NOOBS möglich. [16]

FiMs verwendet die Linux-Distribution Raspbian, welche mit dem NOOBS-Installer installiert wurde. Die Entscheidung fiel für den NOOBS-Installer, weil die Installation des Betriebssystems damit am einfachsten möglich ist. Das Operating System Raspbian verfügt über eine einfach zu bedienende Desktop-Umgebung, mit der es möglich ist, Daten und Files abzuspeichern/zu bearbeiten.

Das Betriebssystem Raspbian bedient den Softwareteil des Projektes. Es laufen also das Gewichtsauslesungs-Pythonscript, die Java GUI und die Autostart-Routine darauf.

Alternativen zu Raspbian gäbe es viele, wie zum Beispiel Ubuntu MATE oder Android. Gegen Android spricht, dass das Entwicklerteam von FiMs über die meisten Erfahrungen bei UNIX-Systemen verfügt. Ubuntu MATE wurde auch nicht ausgewählt, weil Raspbian das offizielle Raspberry Pi Betriebssystem ist und es dafür um einiges mehr an Dokumentation verfügt.

3.2.6. Distributed Systems

3.2.6.1. Funktionsweise der Speicherverwaltung bei Distributed Systems

Ein verteiltes System ist einfach gesagt eine Sammlung von Rechenelementen. Es erscheint seinen Benutzern als ein einziges zusammenhängendes System.

Es macht nicht immer Sinn, Distributed Systems zu verwenden oder zu entwickeln. Es ist stark vom Anwendungsfall abhängig. Ein Distributed System soll Ressourcen so leicht wie möglich zugänglich machen und es soll verhindern, dass dem Endbenutzer auffällt, dass Ressourcen im Netzwerk verteilt sind. Ebenfalls soll es leicht editierbar und offen sein.

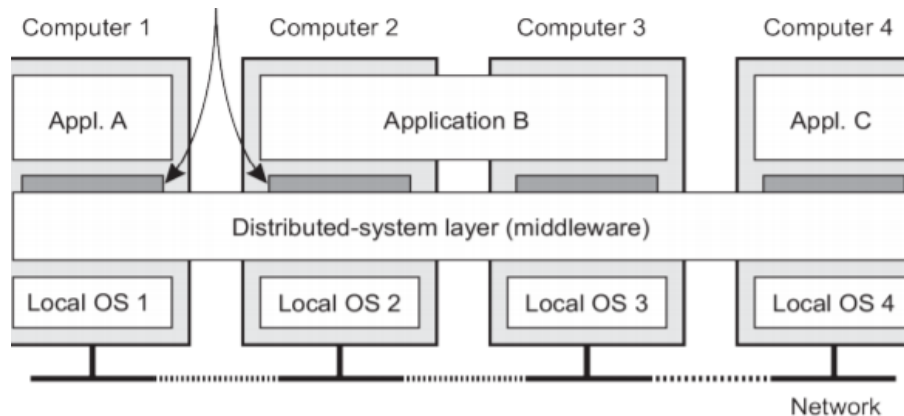


Abbildung 3-21: Distributed Systems Prinzipblockschaltbild (Quelle: [17])

Der Begriff Distributed Memory ist im Allgemeinen exakt das Gleiche wie Distributed Systems, nur eben speziell auf die Speicherverwaltung bezogen. Bei FiMs wird genau diese Methodik verwendet. Beispielsweise wird die Ausgabe des python scripts, entsprechend den Wägedaten vom ADC, in einer Textdatei gespeichert. Es gibt eine weitere Textdatei für das Flaschenprofilssystem, die komplett unabhängig voneinander agieren. Das Raspberry Pi greift auf die Dateien zu, wann immer es notwendig ist.

3.2.6.2. FIFO-Prinzip

First In – First Out, häufig abgekürzt mit FIFO, gleichbedeutend mit „First come, first served.“ bzw. FCFS (engl. für „Wer zuerst kommt, mahlt zuerst.“), bezeichnet jegliche Verfahren der Speicherung, bei denen diejenigen Elemente, die zuerst gespeichert wurden, auch zuerst wieder aus dem Speicher entnommen werden.

Eine solche Datenstruktur wird auch als (Warte-)Schlange bezeichnet. Andere Prinzipien sind das „Last In – First Out“-Verfahren (LIFO, Stapel), das „Highest In – First Out“-Verfahren (HIFO), bei dem das Element zuerst entnommen wird, welches den höchsten Wert besitzt und das „Lowest In – First Out“-Verfahren (LOFO), bei dem als erstes das niederstwertige Element entnommen wird. [17]

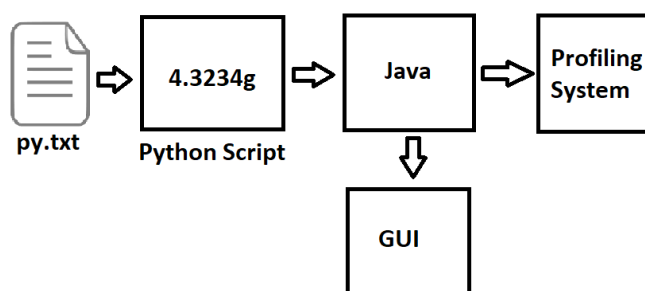


Abbildung 3-23 Blocksaltbild FIFO-Prinzip

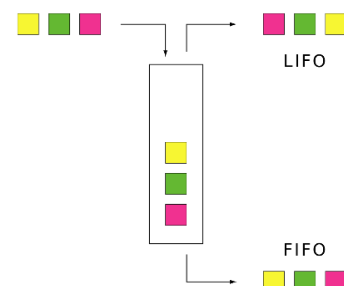


Abbildung 3-22: FIFO/LIFO-Prinzip

Das FIFO Prinzip wird für das Filehandling verwendet. Es werden vom Pythonscript, im Takt von ~1.5s, Gewichtsdaten ausgelesen und in eine neue Zeile der Textdatei geschrieben. Diese Zeit ist von mehreren Faktoren abhängig, beispielsweise der Rechenleistung. Immer der neueste/unterste Wert der Textdatei wird mithilfe einer Java-Funktion ausgelesen und weiterverarbeitet (=>FIFO- Funktion), wenn Gewichtsdaten erfasst und verarbeitet werden sollen. Dieser Wert wird danach mithilfe des Profiling-Systems des Java-Programms verarbeitet.

3.2.6.3. Pipes/Pipelining

Wie die **Ausgabe** vom Pythonscript genau als **Eingabe** in das Java-Programm übertragen wird, wird nachfolgend beschrieben:

Unix-Systemressourcen bestehen hauptsächlich aus Files. Um zwischen den verschiedenen Files zu kommunizieren, wird die Methode „Pipes“ (engl. für „Rohrleitung“) verwendet. Pipes ermöglichen den Datenstrom zwischen Prozessen durch einen Puffer mithilfe des „FIFO-Prinzips“.

Es kommt häufig vor, dass das erste Programm in einer Kommandozeile Ausgaben erzeugt, die als Eingaben für das nächste Programm dienen. In der Zeile:

sort < in | head -30

besagt der senkrechte Strich, **Pipe-Symbol** genannt, dass die Ausgabe von sort als Eingabe von head verwendet werden soll, womit die Erzeugung, Verwendung und das Löschen einer temporären Datei vermieden wird. Eine Ansammlung von Kommandos, die durch das Pipe-Symbol verbunden sind, kann beliebig viele Kommandos enthalten und wird als **Pipeline** bezeichnet. [13]

Beim Projekt wird die Pipe-Methodik verwendet, um die Ausgabe des Python-Scripts (Daten entsprechend dem gemessenen Gewicht in kg) in dem Java-Programm verwenden zu können.

3.2.7. Wahl der Programmiersprache

Die verwendete Programmiersprache ist, wie zuvor erwähnt, **JAVA**.

Zur Debatte für die Entscheidung der Programmiersprache für die GUI Programmierung standen C# und Java. Im Endeffekt fiel die Entscheidung für Java, da das Entwicklerteam bereits einige Erfahrungen mit dem Java-Tool: „WindowBuilder“ in der zweiten und dritten Klasse der HTL im FSST-Unterricht (Fachspezifische Softwaretechnik) gemacht hat und wenig Erfahrung mit C# hat.

Das Bewusstsein des Teams darüber, dass mit C# anspruchsvollere GUI's programmiert werden können, existiert, jedoch mit dem Kompromiss der höheren Komplexität der Programmiersprache C#.

Mit Java und „WindowBuilder“ ist es möglich, per Drag-and-Drop GUI-Elemente in das GUI-Fenster zu ziehen. Beispielsweise sind verschiedene Buttons (Push-Buttons, Radio Buttons, Checkboxes...), Layouts, Labels, TextAreas, Textfields, Jtables und viele mehr GUI-Elemente verfügbar.

Wenn das Drag-and-Drop ausgeführt und die Größe der Elemente eingestellt ist, wird im Hintergrund von WindowBuilder zeitgleich der entsprechende Code generiert. Dieser Vorgang ist dynamisch, damit ist gemeint, dass zum Beispiel bei Änderung der Größe diverser Elemente sofort der Code angepasst wird.

Um also eine WindowBuilder-GUI mit Java zu erstellen, benötigt es also keiner großartigen Programmierkenntnisse, lediglich einiger Tricks. Probleme dabei waren beispielsweise mehrere Fenster mithilfe eines einzigen Programms zu organisieren, also nicht für jedes GUI-Fenster eine eigene Klasse zu erstellen, damit Gewichtswerte nicht von Klasse zu Klasse weitergegeben werden müssen und das lokal passieren kann.

Zusammenfassend sind die besagten Gründe also die ausschlaggebenden Argumente für die Entscheidung für Java.

3.2.8. FiMs GUI

In diesem Kapitel werden grundlegende verwendete Programmiermethoden erläutert, mit denen die FiMs-GUI realisiert wurde. Kurz gesagt, eine GUI ist eine Form einer grafischen Schnittstelle zwischen Computer und Benutzer.

Mithilfe von Widges (entsprechen den vorher genannten GUI-Elementen), Symbolen und Steuerelementen soll die Bedienung des Programmes erleichtert werden. Im Vordergrund steht hierbei die übersichtliche Darstellung der Systemfunktionen und die einfache Handhabung des Programmes.

Da dieses Projekt für die Bar der Tanzschule Seifert entwickelt wird und hierbei eine dynamische Umgebung essentiell ist, sind einfache Handhabung und Flexibilität besonders wichtig, um die Messroutinen so schnell wie möglich erledigen zu können.

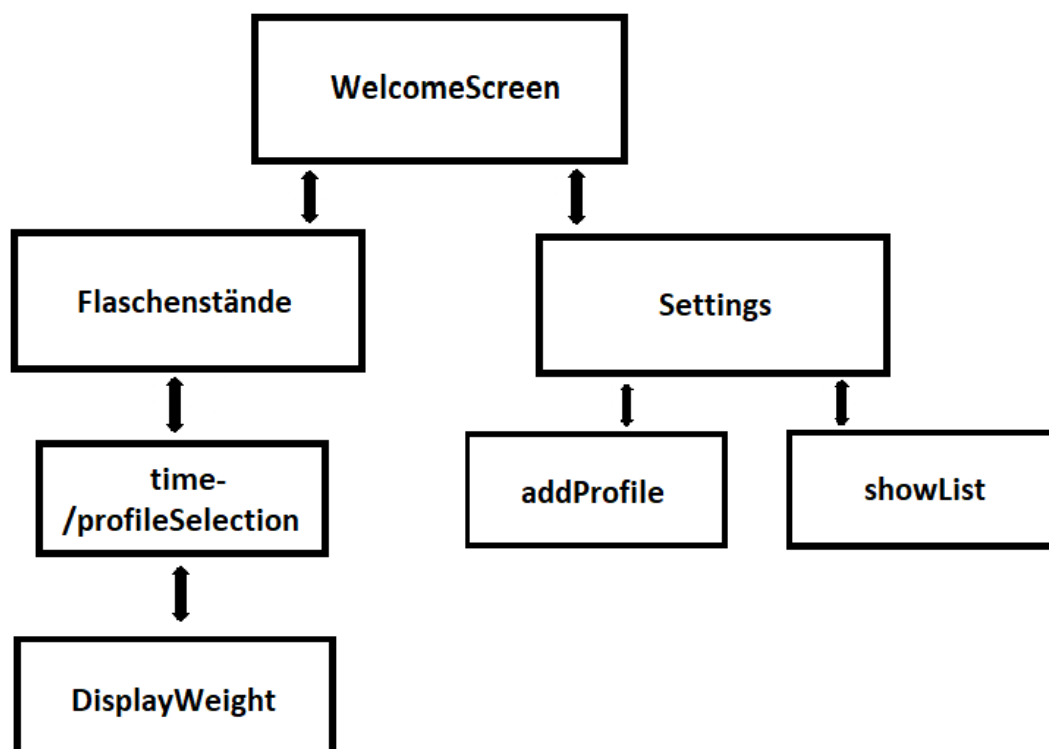


Abbildung 3-24: Blockschaltbild FiMs GUI

Folgende GUI-Fenster sind verfügbar:

➤ WelcomeScreen:

In diesem Fenster ist es möglich auszuwählen, ob man die Systemeigenschaften ändern möchte. Weiters kann man Flaschenstände vor oder nach dem Dienst messen lassen.

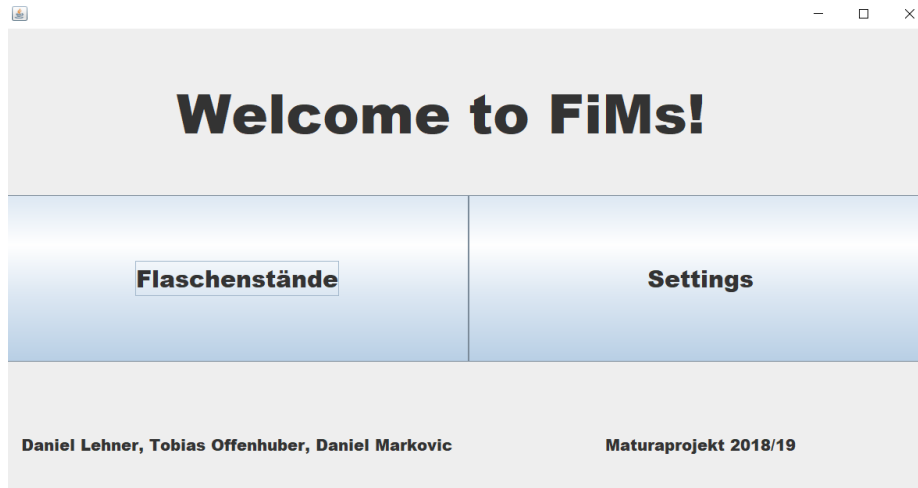


Abbildung 3-25: Welcome Screen der GUI

➤ Settings:

Es gibt zwei Auswahlmöglichkeiten. Zum einen kann man ein neues Flaschenprofil hinzufügen (AddProfile-Screen). Zum anderen ist es möglich, die gesamte Liste der bereits vorhandenen Flaschenprofile anzeigen zu lassen.

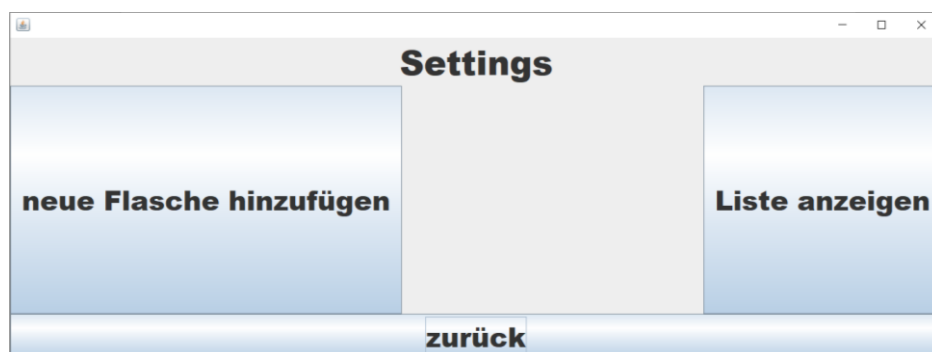


Abbildung 3-26: Settings

➤ Flaschenstände:

Hier ist es möglich, Flaschenstände zu bestimmen. Weiters folgen daraufhin die Windows TimeSelection , ProfileSelection und DisplayWeight.

➤ TimeSelection:

TimeSelectionScreen ist ein Teil der Flaschenstände Funktion. Bei TimeSelection kann man auswählen, ob der Wägevorgang am Anfang oder am Ende des Dienstes verwendet wird.



Abbildung 3-27: TimeSelection

➤ ProfileSelection:

In diesem Fenster gibt es die Auswahlmöglichkeit der verschiedenen Flaschenprofile. Es ist hier wichtig, die richtige anzukreuzen, denn bei einer Wahl des falschen Profils wird die Berechnung inkorrekt ausfallen. Man kann also die Marke der Flasche auswählen und danach erst mit der Gewichtsmessung fortfahren.

➤ DisplayWeight:

Es wird das Gewicht der Flasche bestimmt. Die Flasche wird auf die Waage gestellt, danach muss der Start-Button betätigt werden. Daraufhin wird das Gewicht angezeigt.



Abbildung 3-28: DisplayWeight

➤ AddProfile:

Diese Methode legt ein neues Flaschenprofil an. Um dies zu realisieren, sind folgende Messungen und Eingaben nötig:

- Zuerst wird der Flaschentyp/Markenname eingegeben.
- volle Flasche wiegen
- Füllmenge des Messbechers eingeben
- leeren Messbecher wiegen
- vollen Messbecher wiegen
- leere Flasche wiegen

Zum Abschluss werden alle Eingaben zur Überprüfung der Daten angezeigt. Wenn der Benutzer die Daten kontrolliert hat und diese für in Ordnung befunden hat, kann er auf „Bestätigung“ drücken und kommt in das Hauptmenü (WelcomeScreen) zurück. Im Hintergrund wird das Flaschenprofil berechnet und abgespeichert.

3.2.8.1. Navigation durch die Softwarestruktur der GUI

Um vom einen in das andere GUI-Fenster zu kommen, wurden folgende Methoden angewendet:

Beim Starten des Java Programms wird die Hauptklasse „main“ gestartet. Diese erzeugt das erste Objekt mithilfe von folgendem Code:

```
WelcomeScreen ws = new WelcomeScreen();
```

Nun ist das Hauptmenü gestartet. Um zwischen den einzelnen Objekten zu wechseln, muss man in die Design-Ansicht wechseln. Um die Design-Ansicht benutzen zu können, ist es nötig, die Klasse mit „WindowBuilder-Editor“ zu öffnen. In dieser Design-Ansicht kann man mithilfe eines Rechtsklicks auf ein Objekt Einstellungen programmieren (meistens „OK“- , „Zurück“- oder „Bestätigung“- Buttons), welche zu einem anderen Fenster führen. Mithilfe eines „action performed – Action Listener“ wird folgender Code erzeugt, der mit Zugabe weniger lines of code zur Standardroutine für die Navigation zwischen den einzelnen Fenstern wird.

```
//Neuer Button Flaschenstände wird erzeugt
JButton btnFlaschenstnde = new JButton("Flaschenst\u00E4nde");

//neuer ActionListener wird erzeugt
    btnFlaschenstnde.addActionListener(new ActionListener() {

//Wenn action performed (entsprechend einem Klick auf den Button //"Flaschenstände"),
wird das nächste Window erzeugt und das alte //disposed, also geschlossen.

public void actionPerformed(ActionEvent arg0) {

        TimeSelection ts = new TimeSelection();
        welcomeScreen.dispose();
    }
});
```

Grundstruktur einer GUI Fensterklasse anhand eines Beispiels (Welcome Screen):

```
package diplomarbeit_java;
//IMPORTS: WindowBuilder Bibliotheken
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JLabel;
import com.jgoodies.forms.factories.DefaultComponentFactory;
import java.awt.Font;
import java.awt.Toolkit;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.GridLayout;
import javax.swing.SwingConstants;
import java.awt.Frame;

public class WelcomeScreen {
    private JFrame welcomeScreen; //neue JFrame Instanz erstellen
    public WelcomeScreen() { //Hauptfunktion zur Anzeige des Frames
        initialize();
        welcomeScreen.setVisible(true); //Sichtbarkeit einschalten
    }
}
```

```
//Initialisieren des Frames
private void initialize() {

//Neues JFrame Objekt erzeugen, size festlegen,...
welcomeScreen = new JFrame();

//Vollbild für alle möglichen Bildschirme einstellen
welcomeScreen.setExtendedState(Frame.MAXIMIZED_BOTH);
Toolkit tk = Toolkit.getDefaultToolkit();
int xSize = ((int) tk.getScreenSize().getWidth());
int ySize = ((int) tk.getScreenSize().getHeight());
welcomeScreen.setSize(1040,439);
welcomeScreen.setAlwaysOnTop(true);

//Position des Fensters einstellen
welcomeScreen.setBounds(100, 100, 450, 300);

//Die Default Close Operation wird initialisiert.
welcomeScreen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```

Abb. 3-25 zeigt die Ausgabe dieses Codes, nämlich den Welcome Screen.

3.2.8.2. Mehrere GUI Fenster mit nur einer Klasse verwalten

Das Flaschenprofilssystem benötigt in den Einstellungen eine Funktion namens: „addProfile“. Diese Funktion benötigt zur Ausführung mehrere GUI-Fenster, wo nacheinander Messungen und Eingaben geschehen. Der Grund, wieso es notwendig ist, mehrere GUI-Fenster mit nur einer Klasse zu verwalten, ist, dass man auf lokale Wägedaten in einer Klasse leichter zugreifen kann als klassenübergreifend. WindowBuilder kann nur in der Design-View verwendet werden, wenn man in einer Klasse ein einziges GUI-Fenster hat. Sobald mehrere GUI-Fenster mit einer Klasse gesteuert werden, muss man einen Trick anwenden:

Um die initialize-functions anderer Fenster zu bekommen, werden sie in einer Testklasse programmiert und erstellt, danach wird der Source-Code kopiert und in die passende initialize-function bei den Buttons eingefügt, wenn ein actionPerformed (Also ein Klick auf einen Button) auftritt. Anhand folgender gekürzter Codestruktur soll dies kenntlich gemacht werden:

```
public class addProfile {  
    //Folgende JFrames werden benötigt, um addProfile auszuführen  
    public JFrame namebestaetigen; // initialize1  
    public JFrame volleFlascheWiegen; // initialize2  
    public JFrame FuellmengeMessbecher; // init3  
    public JFrame leererMessbecher; // init4  
    public JFrame vollerMessbecher; // init5  
    public JFrame leereFlasche; // init6  
    public JFrame alleDatenAnzeigen; // init7  
  
    public addProfile() {  
        initialize1(); //Hauptfunktion ausführen  
        namebestaetigen.setVisible(true);  
    }  
  
    public void initialize1() {  
        namebestaetigen = new JFrame();  
  
        //hier sollte der Code für Grafikeinstellungen sein, aus Platzgründen gekürzt. //Wurde  
        //im obigen Beispiel bereits näher gebracht.  
  
        JButton btnBesttigung = new JButton("Bestaetigung");  
        btnBesttigung.addActionListener(new ActionListener() {  
            //Wenn Bestätigt, weiter zum nächsten Window  
            public void actionPerformed(ActionEvent arg0) {  
  
                volleFlaschenWiegenAusfuehren();  
            }  
        });  
  
        // nächstes Fenster  
        public void volleFlaschenWiegenAusfuehren() {  
            namebestaetigen.dispose(); //vorheriges Fenster schließen  
  
            initialize2();  
            volleFlascheWiegen.setVisible(true);  
        }  
    }  
}
```

```
public void initialize2() {  
    // volleFlascheWiegen  
  
    volleFlascheWiegen = new JFrame();  
  
    public void actionPerformed(ActionEvent arg0) {  
  
        // weiter zu füllmenge messbecher  
        FuellmengeMessbecherAusführen();  
    }  
  
    public void actionPerformed(ActionEvent e) {  
        // zurück zu namebestätigen  
        volleFlascheWiegen.dispose();  
        initialize1();  
        namebestaetigen.setVisible(true);  
    }  
});  
  
volleFlascheWiegen.setVisible(true);  
}  
  
public void FuellmengeMessbecherAusführen() {  
    volleFlascheWiegen.dispose();  
  
    initialize3();  
    FuellmengeMessbecher.setVisible(true);  
}
```

```
private void initialize3() {  
    // FuellmengeMessbecher  
    FuellmengeMessbecher = new JFrame();  
    JButton btnOk = new JButton("OK");  
    btnOk.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent e) {  
            // leerer Messbecher ausführen  
            leererMessbecherAusführen();  
        }  
    });  
    FuellmengeMessbecher.getContentPane().add(btnOk, BorderLayout.EAST);  
}  
//und so weiter
```

Wie man sieht, wird einfach nacheinander jedes Fenster geöffnet. Beim Klick auf „Bestätigung“, „Zurück“ oder „OK“ wird die Funktion „Action Performed“ aufgerufen, und hier wird das alte Fenster jeweils geschlossen und das neue Fenster geöffnet, indem die jeweilig passende initialise-Funktion aufgerufen wird und mit setVisible() sichtbar gemacht wird.

3.4. IPC – Interprozesskommunikation

3.4.3. Überblick

Unter Interprozesskommunikation versteht man die Kommunikation zwischen zwei oder mehreren Prozessen. Wichtig ist hierbei die Gewährleistung einer gut strukturierten und vor allem ununterbrochenen Kommunikation. Hierbei spielen folgende Punkte eine wichtige Rolle:

- Das Weiterreichen von Informationen zwischen Prozessen,
- dass Prozesse sich, bei kritischen Aktivitäten, nicht in die Quere kommen und
- das saubere Ablaufen, bei Abhängigkeiten, das heißt, welcher Prozess, wann, auf welchen Prozess warten muss.

Prinzipiell lassen sich diese Punkte nicht nur bei Prozessen, sondern auch bei Threads anwenden. Der erste Punkt – das Weiterreichen von Informationen – spielt hierbei jedoch keine Rolle, da Threads innerhalb des selbem Adressraumes kommunizieren. Ist dies nicht der Fall, werden Threads wie Prozesse behandelt. Für Punkt zwei und drei gelten die gleichen Problemstellungen und Lösungsansätze wie bei Prozessen.

3.4.4. Prozesse und Threads

Interprozesskommunikation tritt, vor allem unter Berücksichtigung von Threads, bei einem Großteil aller Softwareprojekte auf; so auch bei FiMs. Hierbei ist es wichtig zu verstehen, worum es sich bei Prozessen und Threads genau handelt.

3.4.4.3. Prozess

Ein Prozess ist einfach ein Programm in Ausführung, inklusive des aktuellen Wertes des Befehlszählers, der Registerinhalte und der Belegungen der Variablen. [13]

Einfach gesagt versteht man unter einem Prozess ein laufendes Programm, beziehungsweise den Ablauf eines Programmes. Hierbei handelt es sich nicht nur um Programme, die der Benutzer aktiv nutzt, sondern auch um jegliche Aktivitäten, die im Hintergrund ablaufen. Jeder Prozess besitzt seinen eigenen Adressraum sowie einen oder mehrere Threads. Der Adressraum beinhaltet unter anderem den Quelltext, die Daten und weitere Ressourcen, wie zum Beispiel geöffnete Dateien. Threads wird jeweils eine bestimmte Aufgabe zugewiesen. Sie dienen somit der überschaubaren Gliederung verschiedenster Aufgaben innerhalb eines Prozesses. [13]

3.4.4.4. Threads

Threads sind in vielen Aspekten ähnlich den Prozessen. Der Unterschied liegt hauptsächlich darin, dass Threads in ein und demselben Adressraum arbeiten. In anderen Worten: Ein Prozess besitzt einen Adressraum, in welchem wiederum die einzelnen Threads des Prozesses arbeiten. Threads können nicht eigenständig handeln, sondern müssen einem Prozess zugehörig sein.

Ein Thread besitzt einen Befehlszähler, der angibt, welcher Befehl als nächstes ausgeführt werden soll. Er besitzt ferner Register, die seine lokalen Variablen beinhalten und einen Keller. [13]

Beim Keller handelt es sich um eine dynamische Datenstruktur; oft auch Stack genannt. Im Keller wird für jede noch nicht beendete Aufgabe ein sogenannter Rahmen angelegt. Somit lässt sich anhand des Kellers auf den vorhergegangenen Ablauf schließen.
[13]

3.4.5. FiMs - Softwarestruktur

Interprozesskommunikation spiegelt sich in vielen Bereichen von FiMs wider. Betrachtet man die Softwarestruktur, ausgehend von einem praktischen Anwendungsfall, so beginnt man bei der Auswertung des Wägemoduls. Die ausgelesenen Daten werden anschließend entsprechend vermittelt und verarbeitet. In diesen drei Teilbereichen - der Auswertung, der Vermittlung und der Verarbeitung - kommt es wiederholt zu Anwendungsfällen und Problemstellungen, die in Relation zur Interprozesskommunikation stehen. (siehe Abbildung 3-23)

3.4.5.3. Auswertung des Wägemoduls

Der ADC sendet die Wägedaten über die GPIO Pins an das Raspberry PI. Diese Daten werden mithilfe eines Pythonscripts ausgewertet und zu brauchbaren Werten verarbeitet.

3.4.5.3.1. Pythonscript

Basierend auf einer Open-Source Bibliothek [19], welche auf der Version-Control Plattform gitlab zur Verfügung steht, wurde ein einfaches Pythonscript programmiert. Dieses Script dient lediglich zur Auswertung der vom ADC kommenden Bitfolgen, sodass diese als das Gewicht ausgegeben werden können.


```
12 hx = HX711(5, 6)
13 hx.set_reading_format("MSB", "MSB")
14
15 hx.set_reference_unit(404)
16
17 hx.reset()
18 hx.tare()
19
20 while True:
21     try:
22         val = hx.get_weight(5)
23         print val
24         sys.stdout.flush()
25
26         hx.power_down()
27         hx.power_up()
28         time.sleep(0.1)
29
30 except (KeyboardInterrupt, SystemExit):
```

Abbildung 3-29: Pythonscript

Zuerst wird ein Objekt der Bibliotheksklasse HX711 erstellt, dem die Pin-Belegung des Data und SCK Pins übergeben wird. (siehe Tabelle 3-3-1) Anschließend wird festgelegt, in welcher Reihenfolge das Script liest. Daraufhin wird die Reference-Unit definiert. Es handelt sich um den Wert eines Gramms. Dieser Wert wurde bereits bei der Bitfolgenanalyse des Prototypens ermittelt.

Der Kern des Scripts besteht aus einer while-Schleife, vor welcher die Waage zurückgesetzt und tariert wird. Innerhalb der while-Schleife wird bei jedem Zyklus das aktuelle Gewicht auf eine Variable geschrieben, welche anschließend ausgegeben wird. Bevor der nächste Schleifenzyklus begonnen wird, ist es wichtig, mit der System nativen Funktion flush() den Standardausgabestrom zu leeren. In anderen Worten: Der Standardausgabestrom übergibt gezwungenermaßen die Daten an sein Ziel. Hier ist das Ziel die Konsolenausgabe (Standardströme – siehe Kapitel 3.4.3.2). Wird dies nicht gemacht, kommt es erst zu einer Ausgabe, nachdem das Programm beendet wurde.

Mit power_down() wird das Auslesen pausiert. So wird sichergestellt, dass andere Prozesse einen definierten Zeitraum haben, in welchem sie die Ausgabe auslesen können, ohne zu Synchronisationsproblem zu kommen (Synchronisation – siehe Kapitel 3.4.3.4).

Innerhalb der Klasse HX711 wird das Gewicht mit folgenden Funktionen, welche sich in gegebener Reihenfolge aufrufen, ermittelt. Bei self handelt es sich um die aufgerufene Instanzierung der Klasse selbst.

- `readNextBit(self)`
 - Gibt das zuletzt am Data Pin angekommen Bit zurück.
- `readNextByte(self)`
 - Fügt neue Bits in einer for-Schleife zu Bytes zusammen.
 - Gibt jedes vollständige Byte zurück.
- `readRawBytes(self)`
 - Gibt ein Array der letzten drei vollständigen Bytes zurück. ($3 * 8 \text{ Bit} = 24\text{Bit}$)
- `read_long(self)`
 - Ermittelt mithilfe der Funktion `convertFromTwoCompliment24bit(self, input-Value)` das Zweierkompliment des Arrays.
 - Fügt das komplementierte Byte Array zu einer einzigen long-Variable zusammen und gibt diese zurück.
- `read_average(self, times)`
 - Gibt den Mittelwert von n long-Variablen zurück.
- `get_value(self, times)`
 - Gibt den Mittelwert, minus dem Offset, zurück.
- `get_weight(self, times)`
 - Dividiert den erhaltenen Wert um die Reference Unit und gibt dies zurück.

Aufruf des Pythonscripts

Das Pythonscript wird innerhalb des Javacodes ausgeführt. Hierbei handelt es sich um Interprozesskommunikation. Das Java Program in Ausführung entspricht einem Prozess. Dieser Prozess führt das Pythonscript als Programm aus, welches wiederum einen Prozess erzeugt.

```
20     p = Runtime.getRuntime().exec("python /home/pi/Documents/da/readscale.py");
21
22     in = new BufferedReader(new InputStreamReader(p.getInputStream()));
```

Abbildung 3-30: Aufruf Pythonscript

p ist ein Objekt der Klasse Process. Bei Process handelt es sich um eine Subklasse der Java-nativen Klasse ProcessBuilder. Mit Process ist es möglich, einen Prozess auszuführen, einen Prozess zu beenden, sowie die Ein- und Ausgabe eines Prozesses zu verwalten.

Dem Process p wird, mithilfe der Java-nativen Klasse Runtime und ihrer Funktionen, ein laufender Prozess zugeteilt.

Jede Java-Anwendung verfügt über eine einzige Instanz der Klasse Runtime, über die die Anwendung mit der Umgebung kommunizieren kann, in der die Anwendung ausgeführt wird. Die aktuelle Laufzeit kann von der getRuntime-Methode abgerufen werden. [20]

Mit der Funktion exec() wird das Script gestartet. exec() ist eine Funktion der Klasse Runtime, welche Programme ausführen kann. Das Script schreibt in jedem while-Schleifen Zyklus einen Wert auf den Standardausgabestrom.

„in“ ist ein Objekt der Klasse BufferedReader. Wie der Name besagt, handelt es sich um einen gebufferten Leser, der unter anderem fähig ist, Standardströme zu lesen. (Buffering – siehe Kapitel 3.4.3.3, Standardströme – siehe Kapitel 3.4.3.2) wird im späteren Verlauf genauer eingegangen. Die Daten, welche im Script dem Standardausgabestrom übergeben werden, werden wiederum dem Standardeingabestrom des Javaprozesses übergeben. Diese Daten werden mithilfe eines InputStreamReader auf den BufferedReader in geschrieben. [21]

3.4.5.4. Standardströme

Ströme sind geordnete Datenfolgen, die eine Quelle (Eingabeströme) oder ein Ziel (Ausgabeströme) haben. [21]

Die Unix-nativen Standardströme sind in den meisten Programmiersprachen eingebunden, auch in Java. Die drei Standardströme unterteilen sich in Eingabe- und Ausgabeströme; namentlich Standardeingabe (stdin), Standardausgabe(stdout) und Standardfehlerausgabe(stderr).

Alle Standardströme werden bei herkömmlichen UNIX Programmen beim Start des Programmes geöffnet.

Zeichen- und Byteströme

Es wird zwischen Zeichen- und Byteströmen unterschieden. Zeichen sind 16-Bit-Unicode-Zeichen. Prinzipiell besteht die Ein- und Ausgabe immer aus Text oder aus binären Daten. Um Text sinnvoll zu übertragen, ist bei heutigen Zeichennormen die Verwendung eines Zeichenstrom notwendig. Für datenbasierte Ein- und Ausgabe werden Byteströme verwendet. Historisch gesehen, sind Byteströme älter als Zeichenströme. Zeichenströme wurden eingeführt, um die Arbeit mit Unicode-Zeichen zu unterstützen.

Für Java gilt: Die Byteströme werden als Eingabe- und Ausgabeströme und die Zeichenströme als Reader und Writer bezeichnet. Alle Ströme sind in dem Paket `java.io` definiert.

Standardeingabe

Dem Standardeingabestrom `stdin` werden sämtliche Daten übergeben, die als Eingabe zu betrachten sind. Tippt man nun beispielsweise auf der Tastatur etwas ein, wird dies zuerst dem Standardeingabestrom übergeben, welcher diese Daten einem Ausgabestrom weiterleitet. Grundsätzlich kann `stdin` als Leser bezeichnet werden.

Standardausgabe

Der Standardausgabestrom `stdout` dient der Ausgabe sämtlicher Daten, beziehungsweise Nachrichten, die entweder angezeigt oder verarbeitet werden müssen. Grundsätzlich kann `stdout` als Schreiber bezeichnet werden.

Standardfehlerausgabe

Der Standardfehlerausgabestrom `stderr` beinhaltet sämtliche Fehler- und Statusmeldungen, die nicht über den normalen Standardausgabestrom ausgegeben werden.

Doch Anwendungen müssen auch Fehlermeldungen ausgeben können, die der Benutzer auch dann zu sehen bekommt, wenn die Standardausgabe umgeleitet wird. Der `err`-Strom dient insbesondere für Fehlermeldungen, die nicht mit der normalen Ausgabe neu geroutet werden.

[21]

3.4.5.5. Vermittlung der Daten

Die erfassten Daten müssen an die entsprechenden Klassen übermittelt werden, damit sie verarbeitet werden können. Dies wird mithilfe eines selbst programmierten Buffers realisiert.

```
30         while ((line = in.readLine()) != null && !isInterrupted()) {  
31             double v = Double.parseDouble(in.readLine());  
32             b.addValue(v);  
33         }
```

Abbildung 3-31: Buffer- Schleife

Zuerst wird in der Klasse, in der das Pythonscript aufgerufen wurde, ein neuer Thread erzeugt. In diesem Thread läuft eine while-Schleife. Solange die letzte Zeile des BufferedReaders in nicht null ist und die Funktion isInterrupted() nicht true zurück gibt, wird die letzte Zeile des BufferReaders auf eine double Variable geschrieben, welche dem Buffer übergeben wird.

Der Buffer ist eine simple Klasse, die bei jedem neuen Eintrag ein double-Array um eine Stelle vergrößert und den neuen Wert an der neuen Stelle speichert. Hierbei ist darauf zu achten, dass alle Funktionen der Buffer-Klasse über das Schlüsselwort synchronized verfügen. (siehe Kapitel 3.4.3.3) Für das Auslesen des Buffers wird ein weiterer Thread erzeugt, in welchem wiederum geprüft wird, ob die Funktion isInterrupted() true zurück gibt. Die Funktion gibt true zurück, wenn der arbeitende Thread frühzeitig beendet wird. Ist dies nicht der Fall, wird einfach der letzte Wert des double-Arrays des Buffers zurückgegeben.

Im Fall der Interprozesskommunikation, in Bezug auf Threads, kommen verschiedenste Themengebiete zum Tragen: das Prinzip des Bufferns, Multithreading und die Synchronisation dieser Vorgänge.

Buffern

Buffern ist im Prinzip nur das Zwischenspeichern von Daten. So kann man dafür sorgen, dass keinerlei Daten verloren gehen, selbst wenn es zu Verarbeitungsengpässen kommt.

```
3     private double[] values;
4     private int size;
5     private int i;
6
7     Backend_Buffer(int size) {
8         values = new double[size];
9         this.size = size;
10        i = 0;
11    }
12
13    synchronized public void addValue(double v) {
14        values[i] = v;
15        i = (i + 1) % size;
16    }
17
18    synchronized public double getValue() {
19        return values[0];
20    }
```

Abbildung 3-32: Funktion der Klasse Backend_Buffer

Zusehen sind die Funktion des verwendeten Buffers. Bei Aufruf des Konstruktors wird ein neues double-Array der Länge size und eine Hilfsvariable i deklariert.

Mit der Funktion `addValue(double)` wird dem Array an der Stelle `i` ein neuer Wert zugewiesen. Anschließend wird `i`, solange die Variable nicht größer als `size` minus eins ist, um eins erhöht. Somit wird beim erneuten Aufrufen der Funktion der Wert an die nächste Stelle des Arrays geschrieben.

Das Schlüsselwort `synchronized` sorgt dafür, dass nur der Thread der die Funktion öffnet innerhalb dieser arbeiten darf. Andere Threads können erst wieder Zugriff auf diese Funktion erlangen, sobald der ursprüngliche Thread diese freigibt.

Multithreading

Unter Multithreading versteht man die Verwendung mehrerer Threads innerhalb eines Prozesses. Hierbei kann es zu erheblichen Problemen in der zeitlichen Einteilung kommen. Es müssen wichtige Maßnahmen ergriffen werden, damit eine entsprechende Synchronisation gewährleistet ist.

[21]

3.4.5.6. Synchronisation

Unter Synchronisation versteht man grundsätzlich jegliche Maßnahmen, die zur richtigen und möglichst effizienten zeitlichen Einteilung aller Abläufe in einem Betriebssystem führen. Da real immer nur ein einziger Prozess zu einem bestimmten Zeitpunkt auf der CPU läuft, ist es notwendig, die richtigen Maßnahmen zu ergreifen. Hierbei gibt es einige Grundbegriffe, die von Bedeutung sind, wie zum Beispiel sogenannte Race Conditions.

Race Conditions

Race Conditions sind Situationen, in denen zwei oder mehr Prozesse einen gemeinsamen Speicher lesen oder beschreiben, bei welchen das Ergebnis abhängig davon ist wer wann läuft.

Race Conditions sind insbesondere gefährlich, da der Programmablauf in den meisten Fällen problemlos ablaufen wird. Jedoch kann es in wenigen Fällen zu scheinbar unerklärlichen Fehlern kommen, die äußerst schwer zu finden sind.

Ohne Verwendung des Schlüsselwortes `synchronized` würde es im Falle von FiMs in der Datenvermittlung zu einer Vielzahl von Race Conditions kommen.

Kritische Abschnitte

Kritische Abschnitte sind Teile eines Programmes, bei welchen auf einen gemeinsamen Speicher zugegriffen wird. Prinzipiell lassen sich somit Race Conditions vermeiden, indem man immer nur einen Prozess zu einem Zeitpunkt auf einen Speicher zugreifen lässt. Dies allein gewährleistet jedoch noch nicht die geforderte Effizienz, die das Programm an den Tag legen muss. Hierfür gilt es weitere Maßnahmen zu ergreifen, um folgende Bedingungen zu erfüllen: [13]

- Keine zwei Prozesse dürfen gleichzeitig in ihren kritischen Regionen sein.
- Es dürfen keine Annahmen über Geschwindigkeit und Anzahl der CPUs gemacht werden.
- Kein Prozess, der außerhalb seiner kritischen Region läuft, darf andere Prozesse blockieren.
- Kein Prozess sollte ewig darauf warten müssen, in seine kritische Region einzutreten.

Um das Eintreten in kritische Abschnitte zu vermeiden, gibt es verschiedenste Vorgehensweisen, wie zum Beispiel den wechselseitigen Ausschluss mit aktivem Warten.

Wechselseitiger Ausschluss mit aktivem Warten

Diese Methodik ist vergleichbar mit der Verwendung der `isInterrupted()` Funktion, welche für die Anforderungen von FiMs vollkommen ausreichend ist.

In vielen Fällen wird die Eliminierung von Race Conditions mit sogenanntem wechselseitigem Ausschluss mit aktivem Warten angegangen. Viele herkömmliche Methoden, wie die Peterson Lösung oder mit TSL Anweisungen, funktionieren zwar, jedoch beinhalten beide aktives Warten. Dies bedeutet, wenn ein Prozess in einen kritischen Abschnitt eintritt überprüft er, ob der Eintritt erlaubt ist. Ist dies nicht der Fall hängt der Prozess so lange in einer Schleife, bis er eintreten darf.

Somit wird aktiv CPU-Zeit verschwendet, wenn aktiv gewartet wird. Lösungsansätze, welche dies umgehen, sperren Prozesse, wenn es ihnen nicht erlaubt ist, den kritischen Abschnitt zu betreten. Der Unterschied liegt darin, dass keine Zeit auf der CPU verschwendet wird, indem der wartende Prozess nicht in einer Schleife hängt, sondern tatsächlich beendet wird, bis er wieder erneut gestartet wird.

[13]

3.4.5.7. *Werkzeuge der Synchronisation*

Sleep n Wakeup

„Sleep and Wakeup“ zählt zu einer der einfachsten Interprozesskommunikation-Primitiven, die Prozesse sperren, wenn sie nicht auf ihren kritischen Abschnitt zugreifen dürfen. Sie besteht aus dem Paar *sleep* und *wakeup*.

sleep ist ein Systemaufruf, der den Aufrufer veranlasst zu sperren. Dieser Zustand bleibt so lange bestehen, bis ein weiterer Prozess ihn wieder aufweckt. Der *wakeup*-Aufruf hat nur einen Parameter, den aufzuweckenden Prozess. [13]

Es besteht die Möglichkeit, dass ein Weckruf an einen noch nicht schlafenden Prozess geschickt wird. Dies hat zur Folge, dass der Weckruf verloren geht. Wenn nun der noch nicht schlafende Prozess beginnt zu schlafen, dann denkt der Weckprozess, dass der zu weckende Prozess bereits geweckt wurde, wodurch dieser nicht mehr die Möglichkeit besitzt, wieder geweckt zu werden.

Eine schnelle Korrektur wäre ein Ändern der Regeln derart, dass ein Weckruf-Warte-Bit in das Bild eingefügt wird. Wenn ein Weckruf an einen Prozess, der noch wach ist, gesendet wird, ist dieses Bit gesetzt. Später, wenn der Prozess versucht schlafen zu gehen, wird das Weckruf-Warte-Bit gelöscht, falls es noch gesetzt ist, aber der Prozess bleibt noch wach. [13]

Ein Weckruf-Warte-Bit liefert jedoch nur eine Lösung für zwei kommunizierende Prozesse. Bei drei oder mehr Prozessen müsste man entsprechend viele Weckruf-Warte-Bits hinzufügen, wodurch dies schnell zu einer ineffizienten und unsicheren Lösungsmethode wird.

Semaphore

Semaphore sind eine eigene Variablenart. Sie bieten eine elegante Lösung für das Weckruf-Warte-Bit Problem. Anstatt eines Bits wird bei Semaphoren eine ganzzahlige Variable verwendet. Dies bietet die Möglichkeit, die Anzahl der Weckrufe für die zukünftige Verwendung zu speichern. Semaphore ähneln im Prinzip sehr der Sleep and Wakeup Methode. Analog zu dieser gibt es bei Semaphoren ebenfalls zwei Systemaufrufe: *up* und *down*.

Diese sind als Vereinfachung von *sleep* und *wakeup* zu verstehen und dienen lediglich der Inkrementierung, beziehungsweise der Dekrementierung der Weckruf-Warte-Variable.

Die *down*-Operation eines Semaphors prüft, ob der Wert größer 0 ist. Falls dem so ist, erniedrigt sie den Wert um eins und macht einfach weiter. Falls der Wert 0 ist, wird der Prozess sofort schlafen gelegt, ohne *down* vollständig auszuführen. [13]

Die *up*-Operation erhöht den vom Semaphor adressierten Wert um eins. In beiden Fällen handelt es sich um sogenannte atomare Aktionen. Dies bedeutet, wenn eine Operation eines Semaphors beginnt, ist dieser Semaphor gesperrt und kein anderer Prozess kann auf diesen zugreifen, bevor die Operation nicht abgeschlossen ist.

Semaphore ermöglichen es somit, mithilfe nur eines Semaphors, eine beliebige Anzahl an Prozessen zu synchronisieren.

[13]

3.4.5.8. Problemstellungen der Synchronisation

Synchronisation geht immer Hand in Hand mit verschiedensten Problemstellungen. Die Nichtbeachtung folgender Möglichkeiten kann zu schwerwiegenden Fehlern führen: von einem kompletten Stopp des Programmablaufes bis hin zur falschen Reihenfolge in der Ausführung der Aufgaben. In allen Fällen ist der gewünschte Programmablauf und somit die korrekte Ausführung der ursprünglichen Aufgabenstellung nicht gewährleistet.

Philosophenproblem

An einem Tisch sitzen fünf Philosophen. Vor ihnen jeweils ein Teller Spaghetti und zwischen allen Tellern jeweils eine Gabel. Ebenso wird angenommen, dass ein Philosoph zum Essen seiner Spaghetti zwei Gabeln benötigt, da diese sonst zu schlüpfrig sind. Das Leben eines Philosophen besteht daraus, zu essen und zu denken. Ist der Philosoph hungrig, so versucht er zu essen, und ist der Philosoph satt, so beginnt er zu denken. Grundsätzlich simpel, jedoch gibt es ein Problem. Tritt nun der unwahrscheinliche Fall auf, dass alle Philosophen gleichzeitig versuchen zu essen, so kommt es zu einem sogenannten Deadlock. Jeder Philosoph nimmt zuerst eine seiner Gabeln, möchte dann die zweite aufheben, aber keiner der Philosophen ist dazu im Stande, da bereits alle Gabeln in Verwendung sind.

Erteilt man nun den Philosophen die Aufgabe zu überprüfen, ob die zweite Gabel frei ist, nach dem sie die erste Gabel genommen haben, so erscheint dies im ersten Moment als korrekte Lösung. Wenn die zweite Gabel belegt ist, legt ein Philosoph die erste Gabel einfach zurück und wartet eine gewisse Zeit und versucht es erneut. Hierbei kann jedoch der Fall eintreten, dass alle Philosophen gleichzeitig die erste Gabel nehmen und so eine Endlosschleife aus Hinlegen und Aufheben entsteht.

Deadlocks

Eine Menge von Prozessen befindet sich in einem Deadlock-Zustand, wenn jeder Prozess aus der Menge auf ein Ereignis wartet, das nur ein anderer Prozess aus der Menge auslösen kann. [13]

Ist dies der Fall, ist es dem Betriebssystem nicht mehr möglich, die erwünschte Aufgabe auszuführen, da es keinem der Prozesse mehr möglich ist zu handeln.

Vorraussetzungen für Deadlocks

- Wechselseitiger Ausschluss: Jede Ressource ist entweder verfügbar oder genau einem Prozess zugeordnet.
- Hold-and-wait Bedingung: Prozesse, die schon Ressourcen reserviert haben, können noch weitere Ressourcen anfordern.
- Ununterbrechbarkeit: Ressourcen, die einem Prozess bewilligt wurden, können diesem nicht gewaltsam wieder entzogen werden. Der Prozess muss sie explizit freigeben.
- Zyklische Wartebedingung: Es muss eine zyklische Kette von Prozessen geben, von denen jeder auf eine Ressource wartet, die dem nächsten Prozess in der Kette gehört.

Es müssen alle Bedingungen erfüllt sein, andernfalls handelt es sich nicht um ein Deadlock.

[13]

Producer/Consumer Problem

Beispiel: Zwei Prozesse müssen miteinander interagieren; ein Erzeuger und ein Verbraucher. Diese Prozesse teilen sich einen gemeinsamen allgemeinen Buffer mit fester Größe. Der Erzeuger legt Informationen in den Buffer, die vom Verbraucher herausgenommen werden. Wenn der Erzeuger eine neue Nachricht in den Buffer legen möchte, dieser aber schon voll ist, entsteht ein Problem.

Hierbei kommt es zu Race Conditions. Handelt es sich um lediglich zwei Prozesse, die auf den Buffer zugreifen, ist die Sleep and Wakeup Methodik, mithilfe des Weckruf-Warte-Bits, ausreichend. Greifen drei oder mehr Prozesse auf den Buffer zu, so arbeitet man bevorzugt mit Semaphoren.

Diese Problematik tritt bei FiMs bei der Datenauswertung und der Vermittlung auf. Im Falle der Auswertung übernehmen die verwendeten Java-nativen Klassen und Funktionen die Synchronisation. Bei der Vermittlung wird mit `synchronized` und der `isInterrupted()` Funktion gearbeitet.

Leser-Schreiber-Problem

Muss ein Schreiber und n Leser auf eine Datei zugreifen, so kommt es zu dem sogenannten Leser-Schreiber-Problem. Schreibt nun der Schreiber etwas in die Datei und gibt seine Zugriffsberechtigung ab, so kann der erste Leser auf die Datei zugreifen. Während ein Leser auf die Datei zugreift, stellt es keine Schwierigkeit dar, weitere Leser hinzuzufügen. Hierbei kann es zu Problemen kommen. Der Schreiber kann erst wieder die Berechtigung erlangen, auf die Datei zuzugreifen, sobald alle Leser ihre Berechtigung abgegeben haben. Kommt nun beispielsweise alle zwei Sekunden ein neuer Leser dazu, aber braucht jeder Leser fünf Sekunden, so besteht nie wieder die Möglichkeit, dass der Schreiber auf die Datei zugreifen kann, da konstant mindestens ein Leser auf die Datei zugreift.

Die Lösung ist einfach – Man stellt die Priorität des Schreibers immer höher als die aller kommenden Leser. Dies hat zur Folge, dass der Schreiber nur auf noch lesende Leser warten muss. Der Nachteil dieser Lösung ist, dass sie weniger Parallelität und somit weniger Geschwindigkeit erreicht. [13]

Dieses Problem kommt beim Filehandling, im Falle der Datenverarbeitung, zum Tragen. Hier übernimmt die verwendete Bibliothek die Synchronisation.
[13]

3.4.5.9. Verarbeitung der Daten

Die dritte und letzte Begegnung mit Interprozesskommunikation findet bei der Verarbeitung der Daten statt. Es handelt sich um das bereits angesprochene Profilingssystem. Hierbei kommt es unter anderem zum Einsatz von Filehandling.

3.4.5.10. Filehandling mit Apache POI

Filehandling ist ein Überbegriff für das Arbeiten mit Dateien. Bei Filehandling kann es zu Synchronisationsproblemen kommen. Hierbei übernimmt die verwendete Bibliothek, Apache POI, den Einsatz nötiger Maßnahmen.

Bei Apache POI handelt es sich um ein Projekt der Apache Software Foundation. Das Ziel des Projektes ist, Nutzern den Umgang mit verschiedensten Dateitypen zu erleichtern. Im Falle von FiMs wird lediglich auf den Teil der Bibliothek zurückgegriffen, welcher für den Umgang mit .xls-Dateien zuständig ist. Das .xls-Dateiformat entspricht einer herkömmlichen Exceldatei.

Es Dateien im .xls Format verwendet, da der Auftragsgeber bereits mit diesen arbeitet und das System auf die bisherige Buchhaltung angepasst wird, wodurch für diese keine Umstellung von Nöten ist.

3.4.5.11. Komponenten der API

Apache POI unterteilt eine .xls-Datei hierarchisch. Ein Workbook besteht aus Sheets, ein Sheet besteht aus Rows, eine Row besteht aus Cells.

Workbook

Ein Workbook stellt die .xls-Datei an sich dar.

```
18 private static HSSFWorkbook readFile(String filename) throws IOException {  
19     try (FileInputStream fis = new FileInputStream(filename)) {  
20         return new HSSFWorkbook(fis); // NOSONAR - should not be closed here  
21     }  
22 }
```

Abbildung 3-33: .xls-Datei öffnen

Die Funktion `readFile(String)` stellt den Standardablauf für das Erstellen eines neuen Workbooks dar. Es wird überprüft, ob ein neues Objekt der Klasse `FileInputStream`, im Bezug auf die zu öffnende Datei, initialisiert werden kann. Ist dies der Fall, gibt `readFile(String)` ein neues Objekt der Klasse `HSSFWorkbook()` zurück. Es handelt sich um ein Workbook der geöffneten Datei.

Sheet

Wie aus einer herkömmlichen Exceldatei bekannt, besteht eine Datei aus mehreren Sheets.

```
36     HSSFWorkbook wb = readfile("abrechnung.xls");  
37     HSSFSheet sheet = wb.getSheetAt(0);
```

Abbildung 3-34: Instanzierungsbeispiel, Workbook und Sheet

Anhand dieses Beispiels ist zu erkennen, dass Sheets Objekte der Klasse HSSFSheet sind, welche über ein zuvor erstelltes Workbook initialisiert werden können.

Row

Rows bilden ein Array aus einzelnen Cells.

Das oben genannte Beispiel lässt sich auch auf Rows beziehen, welche ein Objekt der Klasse HSSFRow sind und über ein zuvor erstelltes Sheet initialisiert werden.

Cell

Für Cells gilt dies ebenso. Jede initialisierte Zelle besitzt einen Wert. Hierbei kann eine Cell jeden bekannten Datentyp annehmen.

```
114     setB_w_e(cell_b_w_e.getNumericCellValue());
```

Abbildung 3-35: Auslesen einer Cell

Hier wird der Wert der Cell cell_b_w_e ausgelesen und auf die Variable b_w_e geschrieben. Es handelt sich hierbei um einen double-Wert, deshalb wird hier die Funktion getNumericCellValue() verwendet, welche einen double-Wert zurückgibt.

```
111     HSSFCell cell_b_w_e = row.getCell(2);
```

Abbildung 3-36: Instanzieren einer Cell

Alle Elemente können erst abgerufen werden, nachdem sie initialisiert wurden, wie hier die Cell cell_b_w_e.

3.4.5.12. Profilingssystem

Das Profilingssystem besteht grundlegend aus zwei lokal gespeicherten .xls-Dateien. Eine dieser Dateien dient als Archiv für angelegte Profile, die andere dient als Vorlage für die tägliche Abrechnung, welche als editierte Kopie dem lokalen Server übergeben wird.

Es wird zwischen zwei Anwendungsfällen unterschieden: dem Betrieb und der Konfiguration.

Barkassa-Abrechnung 5. Stock - TS Seifert				20.11.2018
Anfangsbestand				
€	500,00	€	-	€ 2,00
€	200,00	€	-	€ 1,00
€	100,00	€	-	€ 0,50
€	50,00	€	-	€ 0,20
€	20,00	€	-	€ 0,10
€	10,00	€	-	€ 0,05
€	5,00	€	-	€ 0,02
				€ 0,01
				€ -
				€ -
Artikel	Anfangs - zählerstand	End - zählerstand	Verkaufte Anzahl	
Soda 1/4				
Apfelsaft				
Soda 1/4				Summe 1/4
Soda 1/8				
Leitungswasser 1/8				
Leitungswasser 1/4				
Veltliner 1L				Summe 1/8
Zweigelt 1L				Summe 1/8
Landwein 1L				Summe 1/8
Cuvée 0,75L				Summe 1/8
Rotgipfler 0,75L				Summe 1/8
Prosecco 0,75L				Summe 0,1
Punsch Orange				Summe 6cl.
Punsch Kirsche				Summe 6cl.
Punsch Mandarine				Summe 6cl.
Punsch Bratapfel				Summe 6cl.
Punsch Apfel-Nuß				Summe 6cl.
Punsch Pfirsich				Summe 6cl.
Punsch Alkoholfrei				Summe 6cl.
Punsch Walderdbeere				Summe 6cl.
Fassbier kg				100g=100ml
Capuccino				Capuccino
Latte Macchiato				Lat. Macch.
Caffee Latte				Gr. Brauner
Melange				Caffee Lat.
Verlängerter				Melange
Kleiner Espresso				Vanillemilch
Großer Brauner				Verlängerte
Milchschaum				Espresso
2x Capuccino				Lat. Pralino
2x Latte Macchiato				Hei. Schok.
2x Caffee Latte				
2x Melange				
2x Verlängerter				Laugenbreze 15
2x Espresso				Speckstange 8
Kakao				Bag. Huhn 6
Vanillemilch				Bag. Chili 6
Heißwasser				Bag. Schin. 6
Kakao kalte Milch				Knoblauchst 8
Latte Pralino				
2x Latte Pralino				
Chocochino				
Ice Shot 400				
Ananassaft				
Orangensaft				
Holunderblütensyrup				
Aperol				
Zitronensaft Pulco				
Limetensaft Pulco				

Abbildung 3-37: Abrechnungsvorlage der Tanzschule Seifert

Betrieb

Während des alltäglichen Betriebes gilt es, die Füllstände der Flaschen zu messen. Hierbei muss zuerst das entsprechende Profil aufgerufen werden, um den Inhalt in Liter zu berechnen, da jede Flüssigkeit ihre eigene Dichte besitzt, wodurch nicht jede Flüssigkeit bei gleicher Menge gleich viel wiegt.

Hierzu muss zuerst das Profilarchiv geöffnet werden. Anschließend wird auf die Vorlage der Abrechnung zugegriffen. In dieser werden die ermittelten Dateien eingeschrieben und als Kopie gespeichert. In beiden Fällen ist es nötig, eine bestimmte Row anhand des entsprechenden Cell Wertes zu ermitteln. Hierbei wird folgende Funktion betrachtet, die zusätzlich einen Einblick auf die Funktionalität von Apache POI bietet.

```
24 private static int findRow(HSSFSheet sheet, String cellContent) {  
25     for (Row row : sheet) {  
26         for (Cell cell : row) {  
27             if (cell.getRichStringCellValue().getString().trim().equals(cellContent)) {  
28                 return row.getRowNum();  
29             }  
30         }  
31     }  
32     return 0;  
33 }
```

Abbildung 3-38: findRow() – Funktion des Java-Programmes

Die Funktion findRow() dient zur Ermittlung der entsprechenden Row, in welcher die benötigten Daten gespeichert sind. Weiß man, in welcher Row die Daten stehen, so kann man diese über die Row-spezifischen Cells abrufen. In zwei verschachtelten for-Schleifen wird Cell für Cell jeder Wert abgefragt, bis die entsprechende Cell gefunden wurde. Ist dies der Fall, wird die Nummer der Row zurückgegeben.

Konfiguration

Unter Konfiguration versteht man vor allem die Erstellung neuer Profile. Sie ist im Prinzip ähnlich dem Betriebsfall. Es wird ebenfalls mit den beiden gezeigten Funktionen gearbeitet. Der Unterschied liegt in der Funktionalität. Bei dem Erstellen eines neuen Profils werden schrittweise alle benötigten Werte eingelesen. Hierbei handelt es sich um:

- den Produktnamen
- das Gewicht der leeren Flasche
- das Gewicht der vollen Flasche
- das Gewicht des leeren Messbechers
- das Gewicht des vollen Messbechers

Mithilfe dieser Werte lassen sich alle für das Profil wichtigen Werte berechnen. Final werden im Profil folgende Werte gespeichert:

- der Produktname
- das Gewicht der leeren Flasche
- das Gewicht der vollen Flasche
- das Gewicht eines Zentiliters
- die Füllmenge einer Flasche

Mithilfe dieser Werte ist es möglich, im Betrieb das gemessene Gewicht in eine Füllmenge umzuberechnen.

4. Ergebnisse

4.2. Hardware

4.2.3. Wägezelle

Nach der Recherche über Messprinzipien und der Entscheidung für den Dehnmessstreifen, mussten wir uns darüber Gedanken machen wie wir ein Gewicht mit der Wägezelle auslesen können. Bei der im Projekt verwendeten Wägezelle handelt es sich um eine 5kg Wägezelle mit 4 Drähten, zum anschließen.



Abbildung 4-1: Wägezelle (Quelle: Elektrische Waage - Netzmafia.de)

4.2.4. Konstruktion Waage

Um die Wägezelle in Betrieb nehmen zu können musste ein Prototyp angefertigt werden, welcher über eine Wägeplatte verfügte, die bei Gewicht eine Dehnung an der Wägezelle verursacht. Die mechanischen Voraussetzungen (biegen der Zelle) mussten gegeben sein damit das Auslesen erst möglich ist.

Der Biegebalken wurde in einer Z-Form aufgebaut, damit das Gewicht der Flaschen gleich auf die vier Dehnmessstreifen wirkt. Jeweils zwei der Dehnmessstreifen dehnen sich und die anderen werden zusammengedrückt. So kommt es zu einer Widerstandsänderung an der Zelle.

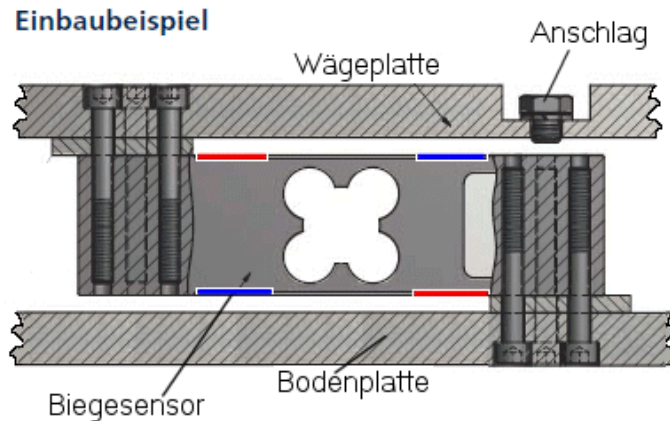


Abbildung 4-2: Wägezelle Z-Form (Quelle: Elektrische Waage - Netzmafia.de)

4.2.5. HX711 Ansteuerung (AVR)

Die Erstellung des ersten Prototyps erfolgte mittels eines Atmel Micro-controllers.

Da die Widerstandsänderung viel zu klein ist musste mit einem Verstärker gearbeitet werden. Da haben wir uns für den HX711 entschieden. Da dieser nicht nur einen analogen Messbrückenverstärker enthält, sondern auch einen 24-Bit Analog/Digital – Wandler. Der HX711 ist auch ausgestattet mit einem Interface, das dem SPI sehr ähnlich ist.

Nun hatten wir eine Möglichkeit die Daten zu verstärken, umzuwandeln und zu verschicken. Nach näherer Recherche im Datenblatt konnten wir herausfinden wie wir die Daten an der Wägezelle schicken können.

Dafür wurde ein AVR Programm geschrieben um 25 Taktimpulse zu schicken sobald der Chip bereit war Daten zu empfangen.

```
while(1)
{
    if(start_retrieve == true)
    {
        for(int i = 0; i <= 24; i++){
            PORTA |= (1<<PA4);
            _delay_us(th);
            PORTA &= ~(1<<PA4);
            _delay_us(tl);
        }
        start_retrieve = false;
        PORTD |= (1<<PD7);
        _delay_ms(1);
    }
}
```

Abbildung 4-3: Clock Generator für Verstärkung von 128

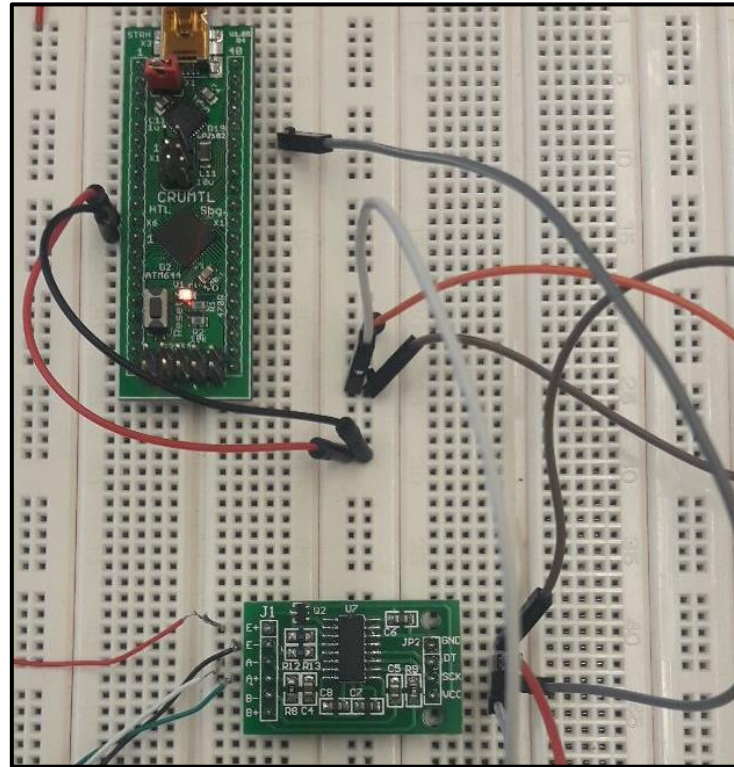


Abbildung 4-4: Steckbrett mit HX711 und CrumbTL

In Abb. 4-4 ist der Schaltungsaufbau, bestehend aus dem CrumbTL mit dem ATmega644 Chip und dem HX711 Verstärker. Der Vcc Pin am HX711 wurde mit +5V versorgt.

4.2.6. Bitfolgenanalyse

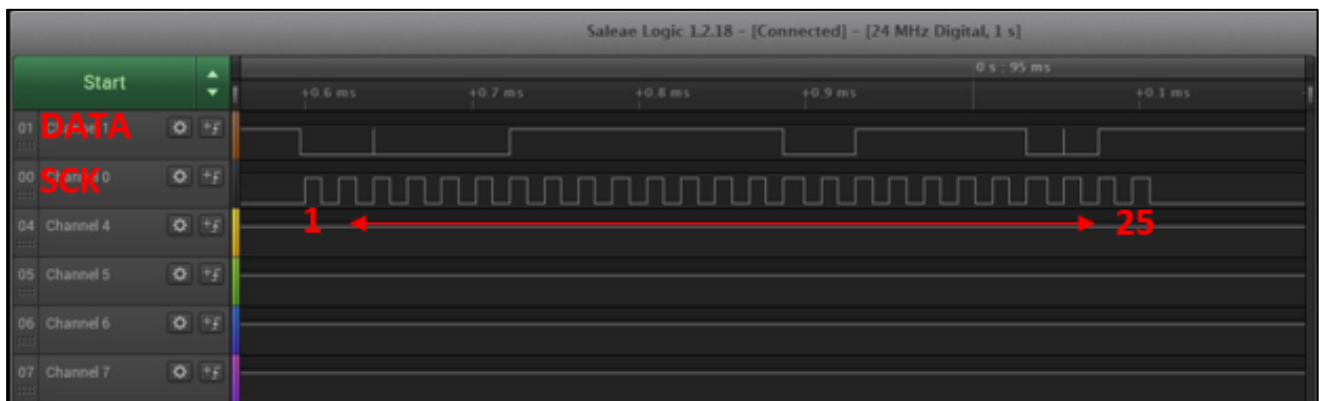


Abbildung 4-5: Bitfolge Wägezelle

Der Daten- und Clock Pin des HX711 wurde am Logic Analyzer angeschlossen, um die Datenausgabe zu überprüfen. Zum Erkennen waren die 25 Clock Impulse, die nach einer fallenden Data Flanke zum senden programmiert wurden. Gleichzeitig kann man die Daten der Waage in Form einer Bitfolge erkennen, wobei jeder Clockimpuls ein Bit darstellt. Nach der Aufzeichnung der Bitfolge war es unsere Aufgabe die Binärfolge zu analysieren. Um bessere und genauere Ergebnisse zu erzielen wurden Langzeitmessungen getätigt um die Verlässlichkeit der Waage zu gewährleisten.

Für die Messungen wurden 4 Gewichte gewählt. Unbelastet (0 kg) und belastet mit 1-3 kg.

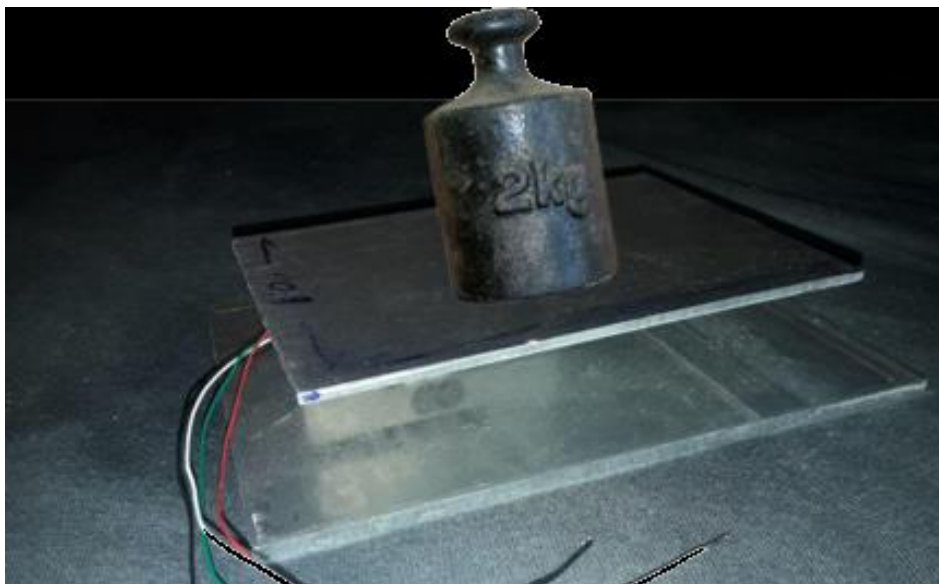


Abbildung 4-6: Messung mit Gewichten

4.2.7. Toleranzprüfung

4.2.7.3. Langzeitmessung

Diese Messungen erfolgten innerhalb von 4 Wochen:

Bitfolge: MSB-LSB	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Gewicht:
17.10.2018	0000				0010					1000				1101					0010					1000	0kg
	0000				0010					1000				1011					1011					0000	
	0000				0010					1000				1011					0100					0000	
	0000																								1kg
	0000					1000				1011				1011					0110					0100	
	0000					1000				1011				1100					1000					1110	
	0000					1000				1011				1011					0011					1101	2kg
	0000					1110				1110				0011					1110					1010	
	0000					1110				1110				0100					0111					0010	
	0000					1110				1110				0101					0011					0001	3kg
	0001					0101				0001				0001					0000					1110	
	0001					0101				0001				0001					0111					1101	
0001					0101				0001				0000					1101					1011		

Bitfolge: MSB-LSB	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Gewicht:
24.10.2018	0000				0010					1010				0101					1010					0011	0kg
	0000				0010					1010				0110					1111					0011	
	0000				0010					1010				0111					0100					0100	
	0000																								1kg
	0000					1000				1101				0010					1101					0110	
	0000					1000				1101				0100					0010					1001	
	0000					1000				1101				1100					0001					0111	2kg
	0000					1110				1111				1100					1111					1001	
	0000					1110				1111				1100					0100					0011	
	0000					1110				1111				1100					0110					1010	3kg
	0001					0101				0010				1001					0101					0010	
	0001					0101				0010				1001					0110					0010	
0001					0101				0010				1000					1100					1100		

Bitfolge: MSB-LSB	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Gewicht:
07.11.2018	0000				0011					1111				1010					0101					1110	0kg
	0000				0011					1111				1001					1101					1000	
	0000				0011					1111				1010					0101					1101	
	0000																								1kg
	0000					1010				0010				0110					0011					1001	
	0000					1010				0010				0110					0110					0011	
	0000					1010				0010				0110					0100					0011	2kg
	0001					0000				0100				1111					1110					1101	
	0001					0000				0101				1111					1010					0001	
	0001					0000				0100				1111					1101					1100	3kg
	0001					0110				0111				1100					1000					1001	
	0001					0110				0111				1100					0011					0101	
0001					0110				0111				1100					0000					0101		

Bitfolge: MSB-LSB	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Gewicht:
14.11.2018	0000				0011					1111				0110					1110					0111	0kg
	0000				0011					1111				0110					1001					0001	
	0000				0011					1111				0110					1010					1100	
	0000																								1kg
	0000					1010				0010				0001					1010					0000	
	0000					1010				0010				0010					1110					0010	
	0000					1010				0010				0011					0001					0000	2kg
	0001					0000				0100				1100					0011					1110	
	0001					0000				0100				1100					0000					0111	
	0001					0000				0100				1100					1011					1111	3kg
	0001					0110				0111				1000					0101					0011	
	0001					0110				0111				1011					1111					1001	
0001					0110				0111				1000					0111					0011		

Abbildung 4-7: Bitfolgenanalyse (Woche 1-4)

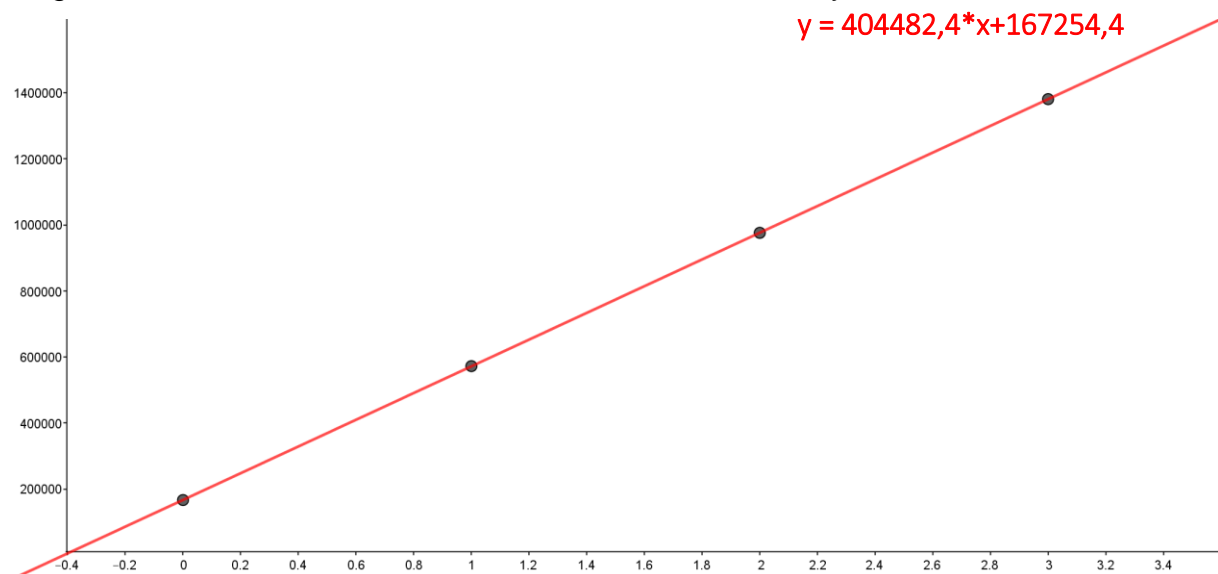
Hexadezimalwert	Dezimalwert	Mittelwert:	Mittelwert – Offset:	1 Gramm	Kilogramm Umrechnung [kg]
0 2 8 D 2 8 028D28	167208			1kg/1000	
0 2 8 B B 0 028BB0	166832	166920	0		0,00000
0 2 8 B 4 0 028B40	166720				
0 8 B B 6 4 08BB64	572260				
0 8 B C 8 E 08BC8E	572558	572346	405426	405	1,00000
0 8 B B 3 D 08BB3D	572221				
0 E E 3 E A 0EE3EA	975850				
0 E E 4 7 2 0EE472	975986	976004	809084		1,99564
0 E E 5 3 1 0EE531	976177				
1 5 1 1 0 E 15110E	1380622				
1 5 1 1 7 D 15117D	1380733	1380642	1213722		2,99369
1 5 1 0 D B 1510DB	1380571				

Abbildung 4-8: Umrechnung der Bitfolge (Woche 1)

Mit Hilfe der Messungen konnte vom Dezimalwert auf das Gewicht zurück geschlossen werden. Abb. 4-8 ist eine Weiterführung der ersten Tabelle aus Abb. 4-7. Dort wurde die Zahl umgerechnet um einen Dezimalwert zu bekommen und damit ein Gewicht.

4.2.7.4. Werte-Kennlinie

Mit den gesammelten Werten aus Abb. 22 konnte eine Gerade ermittelt werden. Diese zeigt die Linearität der Werte und der Stabilität des Messsystems.



Damit war der Prototyp Waage abgeschlossen und es konnte mit der Einbindung des Python Scripts am RaspberryPi begonnen werden. Der Umstieg auf das Script ist erfolgreich gelungen.

4.2.8. Gehäuse

Das Design für das Gehäuse wurde in Solid Edge S7 erstellt. Der leichte Einstieg in die Zeichensoftware und die vielen Werkzeuge die das Programm bereit hielt, erleichterten die Designphase erheblich.

Für das Gehäuse wurden bestimmte Kriterien aufgestellt, die man beachten musste:

Die Elektronik soll wasserdicht verbaut werden. Die Basis soll aus schwerem Material sein um dem umkippen vorzubeugen. Für das Material für die Basis, ist die Entscheidung auf Holz gefallen, welches schwarz lackiert wurde und Wasserfest versigelt wurde. Das hat vor allem einen Vorteil, da es leicht zu bearbeiten ist und sehr kostengünstig.

Das gleiche auch für die Außenwände der Elektronik-Box welche zusammengeschaubt wurden und weiß lackiert wurden.

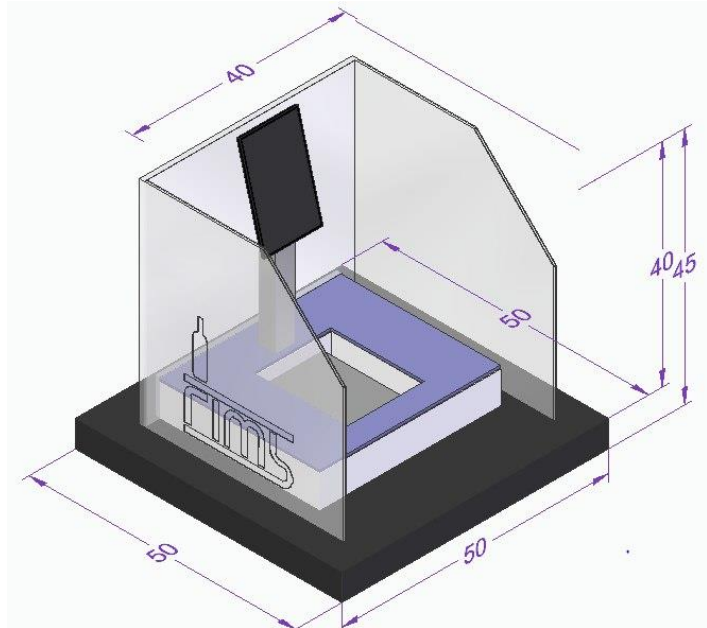


Abbildung 4-9: Gehäusezeichnung

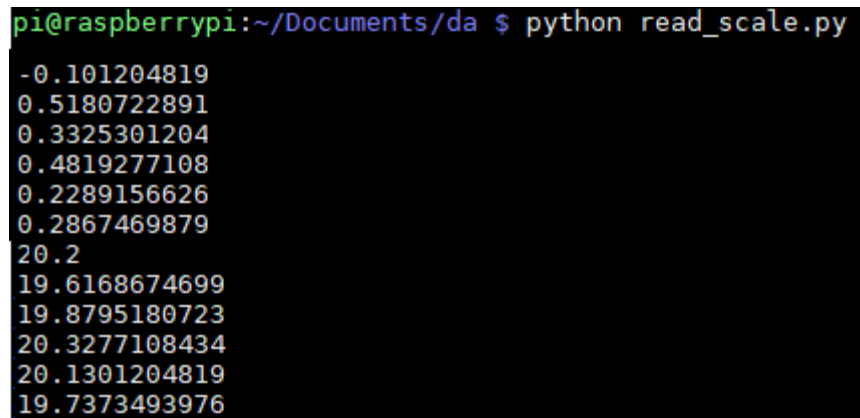
Der „Deckel“ der Box wurde aus dünnem PVC tiefgezogen, damit sich in der Mitte eine Wanne für die Waage bildet und wurde dann auf die Box geschraubt. Der Kabelkanal, welcher bis zum LCD-Bildschirm geht, wurde aus Aluminium gefertigt, damit alles fest verschraubt werden kann.

Die Außenwände sind aus dünnem durchsichtigem Plexiglas gefertigt worden.

4.3. Software

4.3.3. Auslesen des Pythonscripts

Die nötige Toleranzeinhaltung ist nicht nur bei der händischen Analyse der Wägedaten wichtig. Auf Grund dessen muss auch die Funktionalität der Software überprüft werden, die die Daten ausliest und verarbeitet. Hierzu wird die Ausgabe des Pythonscripts genauer betrachtet.



```
pi@raspberrypi:~/Documents/da $ python read_scale.py
-0.101204819
0.5180722891
0.3325301204
0.4819277108
0.2289156626
0.2867469879
20.2
19.6168674699
19.8795180723
20.3277108434
20.1301204819
19.7373493976
```

Abbildung 4-10:Ausgabe Pythonscript

In diesem Beispiel wurde das Script gestartet, somit die Waage initialisiert und ein Gewicht von zwanzig Gramm gemessen. Zu Beginn muss das System, initialisiert werden. Dieser Vorgang benötigt einige Sekunden. Ist die Waage initialisiert, so beginnt diese die entsprechenden Daten auszugeben. Zusehen ist, dass zu Beginn der Wert um null Gramm schwankt. Hierbei kann es zu negativen Werten kommen, da der Offsetbetrag vom ausgelesenen Betrag abgezogen wird. Ist der Offsetbetrag größer gibt das Script einen negativen Wert aus. Es kommt zu leichten Schwankungen um den gewünschten Wert, jedoch liegen diese innerhalb der Toleranz. Das Auslesen der Wägedaten funktioniert den Anforderungen entsprechend.

4.3.4. Auslesen der Wägedaten in Java

Das Java-Programm führt das Pythonscript aus und verarbeitet dessen Daten. Hierbei werden diese einerseits bei der Erstellung neuer Profile benötigt, andererseits bei der Verwendung von FiMs im alltäglichen Betrieb.

4.3.5. Profilerstellung

Im Falle der Profilerstellung werden mehrere Werte gemessen, die zur Berechnung der Profildaten benötigt werden.

11	Aperol	1159,07952	182,828916	976,250602	9,03996803
----	--------	------------	------------	------------	------------

In der Abbildung sieht man die Werte eines erstellten Profils. Der Name Aperol wurde beispielhaft gewählt. Es wurde ein Messbecher mit einem Liter Wasser gefüllt. Dieser wurde gewogen und dessen Gewicht gespeichert. Anschließend wurde die Füllmenge des kleineren Messbechers angegeben; hier 3cl. Daraufhin wurde der leere und volle, kleine Messbecher gemessen. Zu guter Letzt wurde noch der leere Messbecher gemessen. Aus diesen Werten ergaben sich, wie in der Abbildung zu sehen ist, folgende Werte: der volle Messbecher wiegt 1159g, der leere Messbecher wiegt 182.8g, der Flascheninhalt wiegt 975g und 1cl wiegt 9.0399g.

Vergleicht man die Dichte mit der gemessenen, so lässt sich eine Abweichung erkennen.

$$\rho_{\text{Wasser}} - \rho_{\text{gemessen}} = 9,97 \frac{\text{g}}{\text{cl}} - 9,0399 \frac{\text{g}}{\text{cl}} = 0,93 \frac{\text{g}}{\text{cl}}$$

Hierbei tritt eine Abweichung von fast einem Gramm pro Zentiliter auf. Wie im weiteren Verlauf erkennbar sein wird schlägt sich dies auf die Toleranzeinhaltung aus. Es kann jedoch behauptet werden, dass Profile erfolgreich erstellt werden können. Somit erfüllt die Profilerstellung weitestgehend ihre Funktionalität.

4.3.6. Betrieb

Im alltäglichen Betrieb wird zunächst der Zeitpunkt der Messung angegeben. In dem folgenden Beispiel wurde der Dienstbeginn als Zeitpunkt gewählt, jedoch basiert die Auswahl des Dienstende auf der gleichen Codestruktur. Anschließend wird das zu messende Profil ausgewählt.

Daraufhin wird der Flascheninhalt gemessen. Hierbei wird mithilfe der Profildaten der Inhalt in Liter ermittelt. Dieser wird in einer Kopie der .xls-Datei „Abrechnung.xls“ gespeichert.

59	Holunderblütensyrup		
60	Aperol	0,73411071	
61	Zitronensaft Pulco		

In diesem Beispiel wurde das Profil, welches zuvor erstellt wurde verwendet. Der Wert steht in der entsprechenden Spalte. Der Messbecher wurde mit 70cl gefüllt. Es ist zu erkennen, dass der Messwert um 3,4cl abweicht. Dies liegt knapp außerhalb des Toleranzbereiches. Das Ziel war ein Maximum von 2cl Messabweichung. Somit wurde die Toleranzeinhaltung im aktuellem Entwicklungsstadium nicht gewährleistet. Es ist jedoch erkennbar, dass die grundsätzliche Funktion des Profilsystems und des Wägemoduls funktionieren.

5. Kostenaufstellung

Kostenpunkt	Preis
Raspberry Pi	€ 31,49
Raspberry Pi - LCD Touchscreen	€ 68,90
HX711	€ 6,99
Wägezelle (Leihgabe)	€ 0,00
USB Kabel 2A für Raspberry Pi	€ 10,90
SD für Betriebssystem Raspberry Pi	€ 12,96
Gehäusekosten (Grundplatte, Lack, Holzplatte, Kunststoffelemente)	€ 65,00
Prototyp Waage (HTL Eigenproduktion)	€ 0,00
Summe:	€ 196,24

6. Schlusswort

Durch die Realisierung von FiMs wissen wir nun, wie wichtig korrekte Absprache innerhalb eines Entwicklungsteams und Zeitmanagement im Vorhinein ist. Auch die Priorität einer guten ausführlichen Dokumentation, die am besten immer direkt nach dem Erreichen bestimmter Meilensteine zu erstellen ist, ist uns nun bewusst.

FiMs wurde soweit realisiert, dass all im Zuge der Diplomarbeit gesetzten Ziele erreicht wurden. Das Flaschenprofilssystem und die GUI, sowie das Gehäuse mit der integrierten Waage sind fertiggestellt und getestet. Die Wägetoleranz ist eingehalten, sowie die GUI ist schnell und zuverlässig, sowie einfach und übersichtlich gestaltet.

Speziell aufgrund der während der Realisierung entstandenen zusätzlichen Arbeitsstunden, ist es im Ermessen des gesamten Teams, Verbesserungspotentiale ganz auszuschöpfen. Verbesserungspotential besitzt das Design der GUI und das Flaschenprofilssystem, mit dem Design des Gehäuses ist das Team zurzeit sehr zufrieden.

Die Zusammenarbeit innerhalb des Teams funktioniert perfekt, ebenso die Zusammenarbeit mit unserem Projektbetreuer Herr Prof. Dipl.-Ing. Siegbert Schrempf, welcher uns immer tatkräftig zur Seite stand.



Abbildung 6-1: Teamfoto FiMs (von links nach rechts: Prof. Dipl.-Ing. Siegbert Schrempf, Daniel Markovic, Tobias Offenhuber, Daniel Lehner)

7. Glossar

FiMs	Flascheninhalt Messsystem
DMS	Dehnmessstreifen
ADC	Analog Digital Converter
DAC	Digital Analog Converter
GUI	Graphical GUI Interface
SPI	Serial Peripheral Interface
LCD	Liquid Crystal Display
DSI	Digital Serial Interface
USB	Universal Serial Bus
NOOBS	New Out Of Box Software
HTL	Höhere Technische Lehranstalt
FSST	Fachspezifische Softwaretechnik
Apache POI	Apache Poor Obfuscation Implementation
API	Application Programming Interface
HSSF	Horrible Spreadsheet Format

8. Quellen- und Literaturverzeichnis

[1] vgl. "Waage - Wikipedia". [Online]. Verfügbar unter: <https://de.wikipedia.org/wiki/Waage>. [Zugegriffen: 22-Mär-2019].

[2] vgl. Ziegler H. "PHYSIKALISCHE MESSTECHNIK B". [Online] Verfügbar unter: http://groups.uni-paderborn.de/physik/studieninfos/praktika/versuche_anleitungen/messmethoden/mtb_kapf.pdf. [Zugegriffen: 22-März-2019]

[3] vgl. "Mikrowaage - Wikipedia". [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Mikrowaage#Elektromagnetische_Kraftkompensation. [Zugegriffen: 22-Mär-2019]

[4] vgl. "Dehnungsmessstreifen - Wikipedia". [Online]. Verfügbar unter: <https://de.wikipedia.org/wiki/Dehnungsmessstreifen>. [Zugegriffen: 23-Mär-2019]

[5] vgl. Plate Jürgen. "Elektronische Waage". [Online]. Verfügbar unter: <http://www.netzmafia.de/skripten/hardware/Arduino/Waage/index.html>. [Zugegriffen: 23-Mär-2019]

[6] vgl. "Analog-Digital-Umsetzer - Wikipedia". [Online]. Verfügbar unter: <https://de.wikipedia.org/wiki/Analog-Digital-Umsetzer>. [Zugegriffen: 23-Mär-2019]

[7] vgl. "Sample-and-Hold-Schaltung - Wikipedia" [Online]. Verfügbar unter: <https://de.wikipedia.org/wiki/Sample-and-Hold-Schaltung>. [Zugegriffen: 23-Mär-2019]

[8] vgl. "HX711 - Datasheet". [Online]. Verfügbar unter: "...". [Zugegriffen: 24-Mär-2019]

[9] vgl. "Serielle Schnittstellen - Wikipedia" [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Serielle_Schnittstelle. [Zugegriffen: 24-Mär-2019]

[10] vgl. "Serial Peripheral Interface - Wikipedia" [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Serial_Peripheral_Interface. [Zugegriffen: 24-Mär-2019]

[11] vgl. "Duplex (Nachrichtentechnik) - Wikipedia". [Online]. Verfügbar unter: [https://de.wikipedia.org/wiki/Duplex_\(Nachrichtentechnik\)](https://de.wikipedia.org/wiki/Duplex_(Nachrichtentechnik)). [Zugegriffen: 24-Mär-2019]

- [12] C. Prof. Dr. Siemers. "Embedded_Systems_Engineering_Handbuch_V0_61a.pdf." 2002.
- [13] [18] vgl. A. S. Tanenbaum, Moderne Betriebssysteme, 2nd ed. 2001.
- [14] "Raspberry Pi," Wikipedia. 06-Feb-2019.
- [15] "Raspberry Pi – Wikipedia." [Online]. Available: https://de.wikipedia.org/wiki/Raspberry_Pi. [Accessed: 20-Feb-2019].
- [16] A. S. Tanenbaum, Distributed Systems 3rd edition (2017). Maarten van Steen, 2017.
- [17] "First In – First Out – Wikipedia." [Online]. Available: https://de.wikipedia.org/wiki/First_In_-_First_Out. [Accessed: 26-Feb-2019].
- [19] tatobari, HX711 Python Library for Raspberry Pi.
- [20] „Java Platform, Standard Edition 8 API Specification“. Oracle.
- [21] vgl. K. Arnold, J. Gosling, und D. Holmes, Die Programmiersprache Java, 1. Aufl., Bd. 1, 1 Bde. Addison-Wesley, 2001.

9. Verzeichnis der Abbildungen, Tabellen und Abkürzungen

Abbildung 1-1: Systemübersicht.....	13
Abbildung 2-1: Gantt-Diagramm Lehner	18
Abbildung 2-2: Gantt-Diagramm Markovic.....	19
Abbildung 2-3: Gantt-Diagramm Offenhuber	20
Abbildung 3-1 Systemübersicht Hardware / Software	24
Abbildung 3-2:Zusammenhang von Masse und Erdbeschleunigung (Quelle: https://de.wikipedia.org/wiki/Waage)	25
Abbildung 3-3: Prinzip der Schwingsaiten – Aufnehmer (Quelle: PHYSIKALISCHE MESSTECHNIK B)	26
Abbildung 3-4: Formel der Grundfrequenz (Quelle: PHYSIKALISCHE MESSTECHNIK B)	26
Abbildung 3-5: Elektrodynamische Kompensations-Waage (Quelle: PHYSIKALISCHE MESSTECHNIK B)	27
Abbildung 3-6: Folien-Dehnungsmessstreifen (Quelle: Dehnungsmessstreifen - Wikipedia).....	28
Abbildung 3-7: Werkstoffe für Metall-DMS und Halbleiter-DMS (Quelle: Dehnmessstreifen - Wikipedia).....	29
Abbildung 3-8: Brückenschaltung (Quelle: Elektronische Waage).....	29
Abbildung 3-9: ADC nach dem Verfahren der sukzessiven Approximation (Quelle: Elektronik für embedded Systems).....	31
Abbildung 3-10:Schematische Anordnung eines Abtast- und Halteglieds (Quelle: Sample- and-Hold-Schaltung - Wikipedia)	31
Abbildung 3-11: Signalverlauf (Quelle: S&H - Wikipedia)	31
Abbildung 3-12:Beschaltung HX711 (Quelle: Datenblatt).....	32
Abbildung 3-13: Tabelle zum Auswählen des Kanals und der Verstärkung (Quelle: Datasheet HX711)	33
Abbildung 3-14:Bitfolge HX711 während der Datenübertragung (Quelle: Datenblatt HX711)	34
Abbildung 3-15: Einfacher SPI-Bus mit Master und Slave (Quelle: SPI - Wikipedia).....	34
Abbildung 3-16:SPI-Sternverbindung (Quelle: SPI-Wikipedia)	35
Abbildung 3-17:Datenübertragung SPI (Quelle: SPI - Wikipedia).....	36
Abbildung 3.3-18: Raspberry-Pi Blockschaltbild (Quelle: http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_Intro.html)	39

Abbildung 3-19: RaspberryPi (Quelle: www.wikipedia.org)	39
Abbildung 3-20: Pinbelegung Raspberry Pi (Quelle: www.raspberrypi.org)	40
Abbildung 3-21: Distributed Systems Prinzipblockschaltbild (Quelle: [17])	43
Abbildung 3-22: FIFO/LIFO-Prinzip	43
Abbildung 3-23: Blockschaltbild FIFO-Prinzip.....	43
Abbildung 3-24: Blockschaltbild FiMs GUI.....	46
Abbildung 3-25: Welcome Screen der GUI.....	47
Abbildung 3-26: Settings	47
Abbildung 3-27: TimeSelection.....	48
Abbildung 3-28: DisplayWeight.....	48
Abbildung 3-29: Pythonscript.....	57
Abbildung 3-30: Aufruf Pythonscript	58
Abbildung 3-31: Buffer- Schleife	60
Abbildung 3-32: Funktion der Klasse Backend_Buffer	61
Abbildung 3-33: .xls-Datei öffnen.....	68
Abbildung 3-34: Instanziierungsbeispiel, Workbook und Sheet.....	69
Abbildung 3-35: Auslesen einer Cell.....	69
Abbildung 3-36: Instanzieren einer Cell.....	69
Abbildung 3-37: Abrechnungsvorlage der Tanzschule Seifert.....	70
Abbildung 3-38: findRow() – Funktion des Java-Programmes.....	71
Abbildung 4-1: Wägezelle (Quelle: Elektrische Waage - Netzmafia.de)	73
Abbildung 4-2:Wägezelle Z-Form (Quelle: Elektrische Waage - Netzmafia.de)	74
Abbildung 4-3: Clock Generator für Verstärkung von 128	74
Abbildung 4-4: Steckbrett mit HX711 und CrumbTL.....	75
Abbildung 4-5: Bitfolge Wägezelle	76
Abbildung 4-6: Messung mit Gewichten	76
Abbildung 4-7: Bitfolgenanalyse (Woche 1-4).....	77
Abbildung 4-8: Umrechnung der Bitfolge (Woche 1)	78
Abbildung 4-9: Gehäusezeichnung.....	79
Abbildung 4-10: Ausgabe Pythonscript.....	80
Abbildung 6-1: Teamfoto FiMs (von links nach rechts: Prof. Dipl.-Ing. Siegbert Schrempf, Daniel Markovic, Tobias Offenhuber, Daniel Lehner)	84

10. Begleitprotokoll gemäß § 9 Abs. 2 Pro

10.2. Begleitprotokoll Markovic

Name: Hr. Daniel Markovic

Diplomarbeitstitel: FiMs – Entwicklung eines Flascheninhalt Messsystems

KW	Beschreibung	Zeitaufwand
38	Diplomarbeitsantrag erstellen, Projektplan	4,5h
39	Diplomarbeitsantrag einreichen, Projektplanung, Laufwerkerstellung	3h
40	Projektplanung, Aufgabenverfeinerung, Blockschaltbilder, Projektnamen Festlegung	4h
41	Gantt Diagramm, Prototyp Gehäuse	6h
42	Test Waage, Ansteuerung HX711	5h
43	Auslesen und analysieren der Bitfolge, Dokumentation, Excel Tabelle erstellen	4h
44	Wöchentliche Auslesung der Bitfolge, Dokumentation, Excel Tabelle erweitern	5h
45	Wöchentliche Auslesung der Bitfolge, Dokumentation, Excel Tabelle erweitern	8h
46	Schaltplanerstellung vom Prototyp, Planerstellung von mechanischer Konstruktion, Dokumentation, Diplomarbeit	8h
47	Besprechung Firma	5h
48	Krankheit	0h
49	Werkstätte Materialbeschaffung	7h
50	Krankheit	0h
51	Routen Platine, Schaltplan finalisieren	7h
52	Weihnachtsferien/Diplomarbeit	10h
1	Weihnachtsferien/Diplomarbeit	10h
2	Planung der Vertiefenden bei der Diplomarbeit	5h
3	Krankheit	0h
4	Review TdoT: Fertigstellung vorzeigbarer Projektversion	9h
5	Planung komplettes Inhaltsverzeichnis der Diplomarbeit	5h
6	Krankheit	0h
7	Semesterferien/Diplomarbeit	1h
8	Krankheit	0h
9	Designänderung Gehäuse (Material)	5h
10	PVC Bearbeitung	7h
11	Krankheit	0h
12	Kabelkanal, Materialsuche	4h
13	Diplomarbeit, Gehäusebau, Platine	5h
14	Gehäusebau	7h
15	Zusammenstellung der Diplomarbeit, Gehäusebau	6h

KW ...Kalenderwoche

10.3. Begleitprotokoll Offenhuber

Name: Hr. Tobias Offenhuber

Diplomarbeitstitel: FiMs – Entwicklung eines Flascheninhalt Messsystems

KW	Beschreibung	Zeitaufwand
38	Diplomarbeitsantrag erstellen, Projektplanung	3,5h
39	Diplomarbeitsantrag einreichen, Projektplanung, Laufwerkerstellung	3h
40	Projektplanung, Aufgabenverfeinerung, Blockschaltbilder, Namensfindung	8h
41	Mechanische Prototypenkonstruktion und Projektplanung	8h
42	Fertigstellung des Prototyps und Projektplanerstellung	7h
43	Diplomarbeit Eintragungen, Bitfolge der Waage auslesen und dokumentieren	8h
44	Auslesung und Interpretation der 24 ADC Bits, Dokumentation	7h
45	Erstellung eines Excel-Files für die Auswertung des Wägemoduls mit automatischer Umwandlung der Binärwerte, Datenbankplanung	6h
46	Erstellung des Projektschaltplans, Fotoshooting für Maturaprojekttitlebild, letzte Testwiegung der Wägezelle für die Doku in die Exceltabelle	8h
47	krank	0h
48	Erweiterung des Schaltplans, Datenbankplanung, Wiegen der leeren Flaschen um Mittelwert/Standardabweichung zu erhalten	8h
49	Erstellung eines Grundgerüsts der Softwarestruktur	8h
50	Beginn Planung Projektpräsentation	8h
51	„kleiner Elefant“ -> Prototypfertigung, Gewichtsdaten am LCD Touchscreen anzeigen lassen, 2 Szenarios (GUI, Profiling-System) erstellen	8h
52	Weihnachtsferien/Diplomarbeit	5h
1	Weihnachtsferien/Diplomarbeit	10h
2	Genaue Planung der vertiefenden Arbeit der Diplomarbeit	8h
3	Projektflyer planen, Logo fertiggestellt	8h
4	GUI verschönern, Vorzeigeversion für TdoT fertigstellen	8h
5	Einigung auf komplettes Inhaltsverzeichnis der Diplomarbeit, GUI-Programmierung	8h
6	Problemstellung: Ausgabe des python scripts in die GUI einlesen -> Entscheidung für Pipeliningcodestruktur	8h
7	Semesterferien/Diplomarbeit	9h
8	Problembeseitigung/ Entwicklung der GUI-Fenster	9h
9	Krank	4h
10	Problemstellung: mehrere GUI-Fenster mit einer Klasse steuern für Profiling-System	9h
11	Designverbesserungen, Codeumstrukturierungen	9h
12	Fertigstellung der Grundstruktur der GUI – erste Tests am LCD-Display	9h
13	Zusammenfügen der Diplomarbeit; GANTT-Diagramme vervollständigen, Begleitprotokoll fertigstellen...	9h
14	Feinarbeit Source Code, letzten Funktionen fertigstellen	
15	Zusammenfügen aller Komponenten, Fertigstellung des Projektes	10h

KW ...Kalenderwoche

10.4. Begleitprotokoll Lehner

Name: Hr. Daniel Lehner

Diplomarbeitstitel: FiMs – Entwicklung eines Flascheninhalt Messsystems

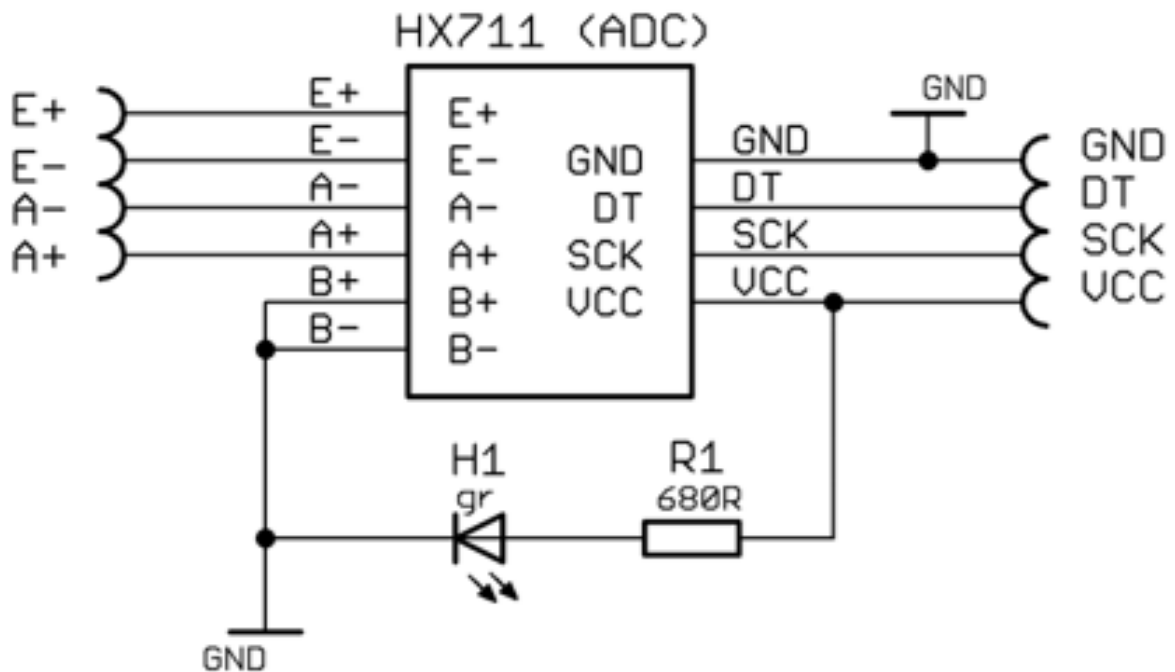
KW	Beschreibung	Zeitaufwand
38	Diplomarbeitsantrag erstellen, Projektplanung	5h
39	Diplomarbeitsantrag einreichen, Projektplanung, Laufwerkerstellung	5h
40	Projektplanung, Aufgabenverfeinerung, Blockschaltbilder, Namensfindung	6h
41	Mechanische Prototypenkonstruktion, Projektplan	5h
42	Prototypenbau, Bigpicture	7h
43	Prototypentestung, Dokumentation	6h
44	Besprechung mit Partnerfirma, Prototypentestung, Dokumentation	5h
45	Prototypentestung, Softwareplanung	5h
46	Krankheit	0h
47	Krankheit	0h
48	Flyerdesign, Softwareplanung	5h
49	Softwareplanung, Skizzendigitalisierung	7h
50	Softwareplanung Finalisierung, Erstellen erster Präsentation, Anfänge der Software	8h
51	„kleiner Elefant“ -> Prototypfertigstellung, Gewichtsdaten am LCD Touchscreen anzeigen lassen, 2 Szenarios (GUI, Profiling-System) erstellen	
52	Weihnachtsferien/Diplomarbeit	10h
1	Weihnachtsferien/Diplomarbeit	10h
2	Planung der Vertiefenden bei der Diplomarbeit	
3	Messung auf +-1g funktioniert	
4	Review TdoT: Fertigstellung vorzeigbarer Projektversion	
5	Planung komplettes Inhaltsverzeichnis der Diplomarbeit	
6	Krankheit	
7	Semesterferien/Diplomarbeit	
8	Krankheit	
9	Weiterarbeiten am Flaschenprofilssystem	
10	Einlesen der Gewichtsdaten in das Java Programm fertig gestellt	
11	Feinarbeit am Source Code	
12	Testphase Profilsystem/GUI	
13	Zusammenfügen der Diplomarbeit	
14	Feinarbeit Source Code Profilsystem, Test	
15	Zusammenfügen der einzelnen Komponenten, Fertigstellung, Tests	

KW ...Kalenderwoche

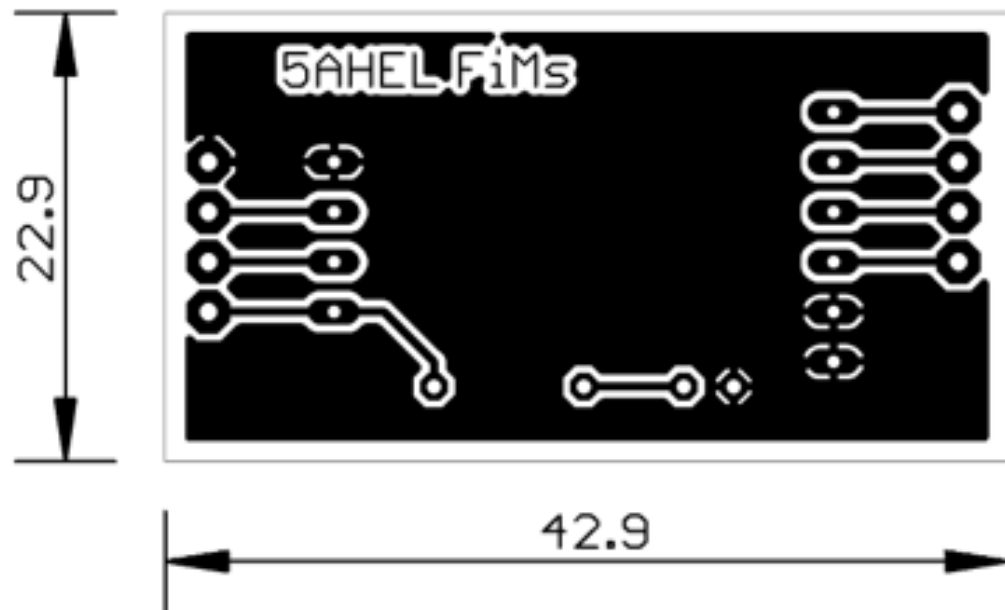
11. Anhang

11.2. Konstruktionspläne

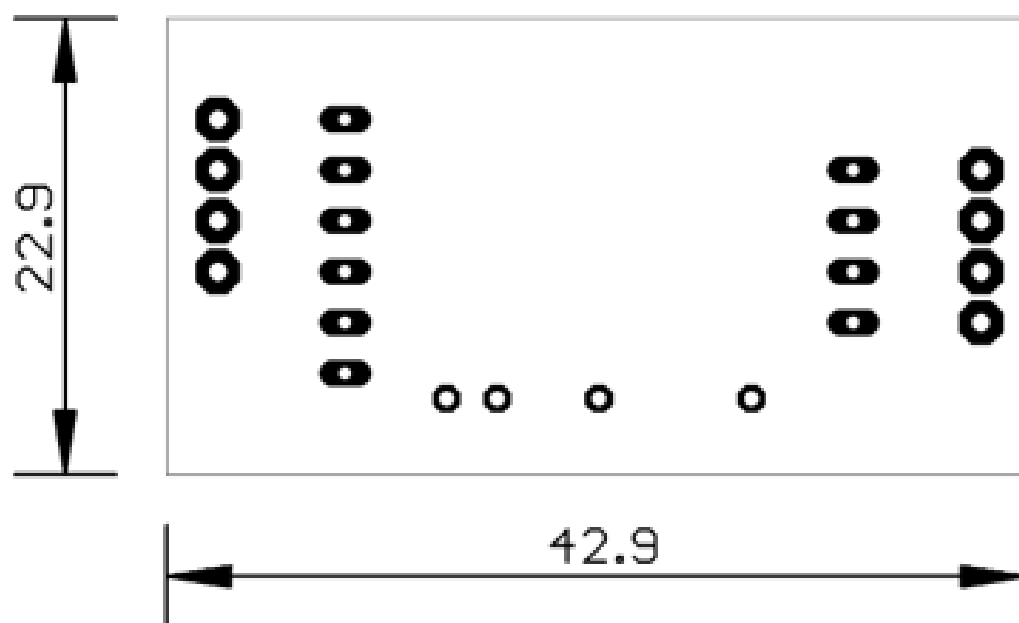
- I. Schalplan
- II. Layout Bot
- III. Layout Top
- IV. Bauteile Top
- V. Bohrplan



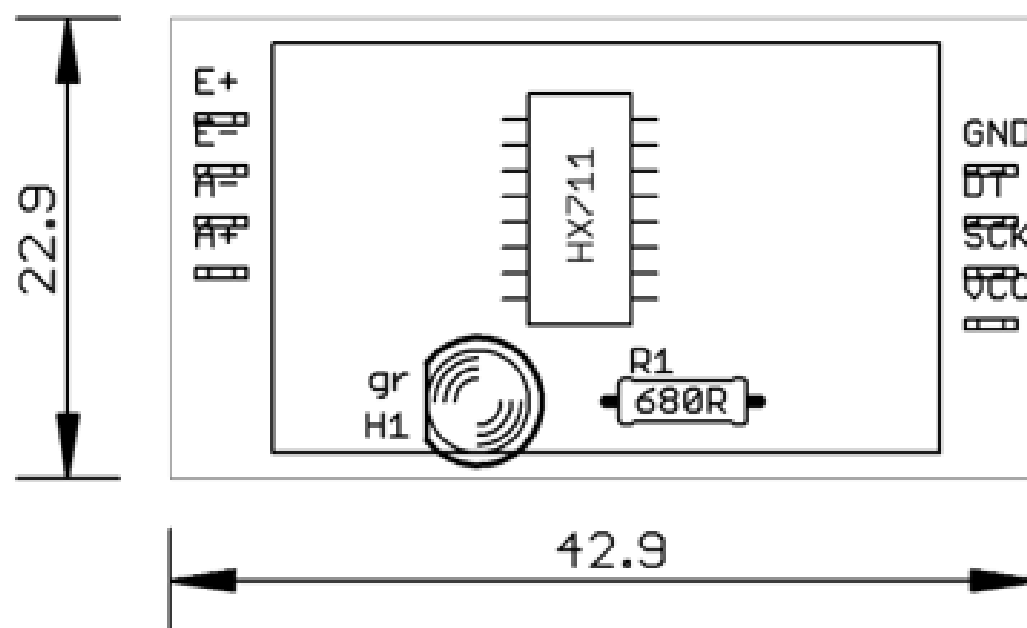
Sachseminar, Datenname:		Name:				Toleranz:		Werkstoff:												
Beilage1		Daniel Markovic																		
 HTBLuVA Salzburg HTBLuVA Salzburg Technische Informatik	ID-Nr.:	5AHEL 09	Gez.:	MarD	Geprüft:		Benutzer:	SreS	Dokumentart:	Schaltplan		Freigebe:	WenN							
	Benennung:					FiMs					Version:	1.0	Revisions:	001	Dokumentstatus:	Fertigung				
	Aufsteckplatine										Mediennr.:	ohne	Dr.:	DE	Datum:	27.03.2019	Blatt:	1	von:	1



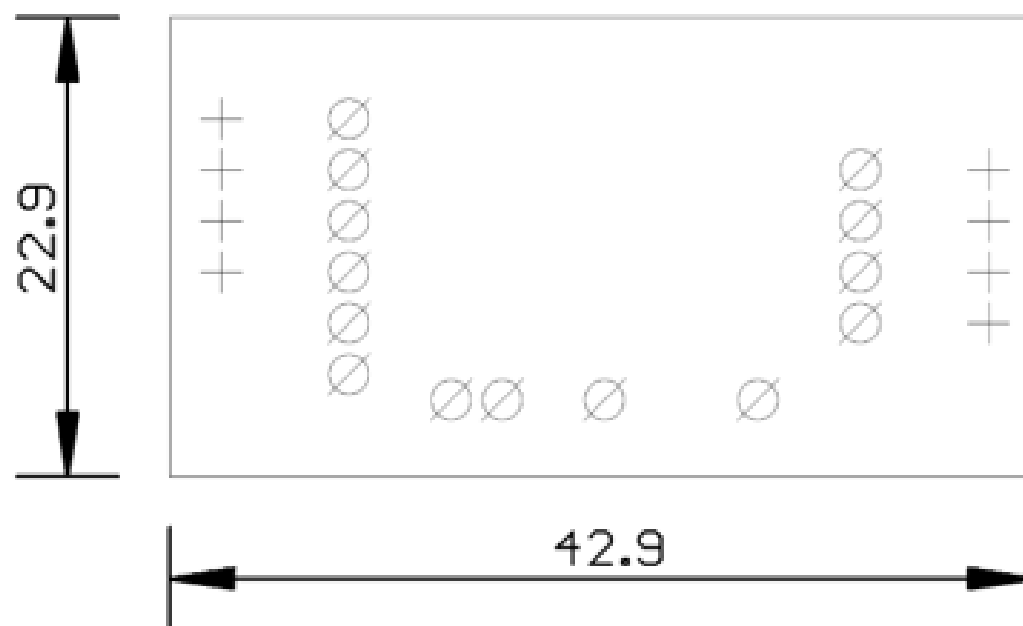
Sachnummer, Dozentname: Beilage2		Name: Daniel Markovic				Toleranz:		Werkstoff:	
 HTBLuVA Salzburg HTBLuVA Salzburg Technische Informatik	ID-Nr.:	Ges.:	Geprüft:	Gezeichnet:	Dokumententyp:		Freigegeben:		
	5AHEL 09	MarD		SreS	Layout Bot		WenN		
	Benennung:				Version:	Revision:	Dokumentationsstatus:		
	FiMs Aufsteckplatine				1.0	001	Fertigung		
	Material:	Spr.:	Datum:	Blz.:	Blz.:				
	ohne	DE	27.03.2019	1	1				



Zeichenummer, Beschreibung: Beilage3		Name: Daniel Markovic				Toleranz:		Werkstoff:						
 HTBLuVA Salzburg HTBLuVA Salzburg Technische Informatik	ID-Nr.:	5AHEL 09	Gez.:	MarD	Geprüft:	SreS	Dokumentiert:		Layout Top	Freigegeben:	WenN			
	Genehmigung:					Version:		Revision:	Dokumentationsstatus:					
	FiMs Aufsteckplatine					1.0		001	Fertigung					
	Maßstab:		ohne		Spr.:		DE		Datum:		27.03.2019	Blatt:	1	Blätter:



Seitennummer, Dokument:		Name:		Titelname:		Verantwortl.	
Beilage4		Daniel Markovic					
 HTBLuVA Salzburg HTBLuVA Salzburg Technische Informatik	ID-Nr.:	Grü.:	Geprüft:	Gezeichnet:	Dokumententitel:		Erstellt:
	5AHEL 09	MarD		SreS	Bauteile Top		WenN
	Benennung:				Version:	Revidiert:	Dokumentstatus:
	FiMs Aufsteckplatine				1.0	001	Fertigung
Mafinanz:		Spr.:	Datum:		Blatt:	Blätter:	
ohne		DE	27.03.2019		1	1	



Zeichnungsnummer, Dokumentname: Beilage5		Name: Daniel Markovic			Toleranz:		Werkstoff:		
 HTBLuVA Salzburg HTBLuVA Salzburg Technische Informatik		ID-Nr.: 5AHEL 09	Entw.: MarD	GeprÜf.: 	Gezeichnet: SreS	Dokumententitel: Bohrplan		Freigegeben: WenN	
		Benennung: FiMs Aufsteckplatine				Version: 1.0	Revision: 001	Dokumentationsart: Fertigung	
						Material: ohne	Spr.: DE	Datum: 27.03.2019	