Bachelorarbeit

Informatik-Ingenieurwesen

Reverse Engineering eines Kaffeevollautomaten

von

Niklas Joachim Eberhard Krüger

Februar 2019

Betreut von

Florian Meyer
Institut für Telematik, Technische Universität Hamburg

Erstprüfer	Prof. Dr. Volker Turau Institut für Telematik Technische Universität Hamburg
Zweitprüfer	Florian Meyer Institut für Telematik Technische Universität Hamburg

Institut für Telematik
Technische Universität Hamburg
TUHH

Eidesstattliche Erklärung

Ich, NIKLAS JOACHIM EBERHARD KRÜGER (Student im Studiengang Informatik-Ingenieurwesen an der Technischen Universität Hamburg, Matr.-Nr. 21491319), versichere an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner Prüfungskommission vorgelegt.

Hamburg, 22. Februar 2019

Niklas Joachim Eberhard Krüger

Abstract

Im Rahmen dieser Arbeit wurde der Speicher des Kaffeevollautomaten „Jura Impressa S9“ reverse engineered. Dafür wurde eine strukturierte Vorgehensweise entwickelt. Einstellung für Einstellung und Funktion für Funktion ergab sich die Bedeutung immer neuer Speicherstellen. Neu gebündelt wurden diese über eine leicht zugängliche API nach außen offen gelegt und abschließend auf einer kleinen Webseite visuell präsentiert. Dabei wurde die Vorgehensweise und die Aussagekraft der Ergebnisse kritisch hinterfragt und in den Kontext des Reverse Engineering eingeordnet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

IoT Internet of Things . . . 1
EEPROM Electrically Erasable Programmable Read-Only Memory . . . 2
RAM Ramdom-Access Memory . . . 2
MCU Microcontroller Unit . . . 5
SPI Serial Peripheral Interface . . . 5
UART Universal Asynchronous Receiver Transmitter . . . 5
ASCII American Standard Code for Information Interchange . . . 8
IDE Integrated Development Environment . . . 9
API Programmierschnittstelle (engl. Application Programming Interface) . . . 10
POSIX Portable Operating System Interface . . . 11
JSON JavaScript Object Notation . . . 11
XML Extensible Markup Language . . . 11
HTML Hypertext Markup Language . . . 12
CSS Cascading Style Sheets . . . 12
JS JavaScript . . . 12
AJAX Asynchronous JavaScript and XML . . . 12
CDN Content Delivery Network . . . 12
ROM Read Only Memory . . . 46
WAMP Web Application Messaging Protocol . . . 46

Kapitel 1

Einführung

Heutige elektronisch gesteuerte Geräte versprechen viel Komfort und eine einfache Handhabung im Alltag. Über das Internet werden die Geräte zunehmend vernetzt und smart. Dabei sind diese Geräte als Gegenstände erst einmal *Objekte* in der realen Welt, wie Menschen, Tiere, Pflanzen oder Produkte wie Autos oder eben auch Kaffeemaschinen, die von S. Madakam [Mad15] unter anderem aufgezählt werden.

Intelligente Dinge (engl. „Smart Things“) machen unsere Welt nutzerfreundlicher im oben angeführten und versprochenen Sinne. Sie setzen sich aus einer Gruppe kontrollierbarer und steuerbarer Gegenstände mit einigen Sensorfunktionen zusammen, die an das Internet angebunden sind. Dies ermöglicht es, jeder Zeit von überall her auf die Produkte zuzugreifen. Dabei ist der Begriff „Smart Things“ ein Schlagwort des Internet of Things (IoT). [Mad15]

Unsere Welt benötigt jedoch nicht nur neue und bessere Produkte, sondern auch wiederverwertbare und reparierbare Dinge. Nun möchte diese Bachelorarbeit eine seit Jahren gut arbeitende Maschine smart machen und sie um eine neue Schnittstelle erweitern.

Daher befasst sich diese Arbeit mit einer eingangs schon erwähnten Kaffeemaschine und in erster Linie mit ihrem Speicher. Der merkt sich nicht nur Betriebszustände, sondern auch Einstellungen, Zählerstände und evtl. manches mehr. Die vorhandenen Informationen können nicht nur gelesen, sondern zum Teil auch verändert werden. Einige weiterführende, smarte Funktionen sind in Maschinen dieser Art bereits ohne eine entsprechende graphische Schnittstelle vorhanden. Sie müssen nur nutzbar gemacht werden.

A. Kamilaris, A. Pitsillides und V. Trifa stellen in ihrem Paper von 2011 [KPT11] beispielsweise dar, dass in naher Zukunft Kaffeemaschinen automatisch einen Kaffee nach Benutzerpräferenzen zubereiten können. Die Möglichkeit, eigene Geräte nun über das Netzwerk zu verwalten, führe zum *Web-enabled Smart Home*.

1.1 Aufgabenstellung

Diese Arbeit reverse engineered ein erhaltenswertes *Objekt* unseres Alltags. Dafür werden die Funktionen und Abläufe eines *Jura Impressa S9 Kaffeevollautomaten* untersucht. Das Gerät soll um eine neue Schnittstelle zur Interaktion erweitert werden, nachdem dessen Speicher entschlüsselt ist. Welches Wort / Byte / Bit speichert welche Information? Welche Bedeutung haben diese für den Betrieb? Hierfür sollen Skripte erstellt und der Speicher systematisch untersucht werden.

Wenn „die Denkweise“ der Kaffeemaschine bekannt ist, sollen Werte im Electrically Erasable Programmable Read-Only Memory (EEPROM) abgefragt, aber auch gezielt verändert sowie Statusinformationen aus dem Ramdom-Access Memory (RAM) ausgelesen werden. Die erhaltenen Rohinformationen werden nach den gewonnenen Erkenntnissen aufbereitet und stehen so für weitere Projekte zur Verfügung.

Als praktische Demonstration des Nutzwerts smarter Programmierung, soll am Ende eine Webseite, mit der Profile anlegen werden können, sodass der Nutzer auf Knopfdruck einen Kaffee nach seinen Lieblings-Präferenzen zubereitet bekommt, entstehen.

1.2 Aufbau der Arbeit

Das nächste Kapitel erörtert den Begriff *Reverse Engineering* und zeigt weitere Arbeiten auf. In Kapitel 3 werden der reale Kaffeevollautomat, dessen Verkabelung und weitere Abhängigkeiten aufgeführt. Darauf aufbauend werden in Kapitel 4 das strategische Vorgehen und die nötigen Skripte in Form eines größeren C++ Programms entwickelt. Im Anschluss werden die Ergebnisse in Kapitel 5 aufbereitet. Kapitel 6 diskutiert aufgetretene Probleme, zeigt die Grenzen dieser Arbeit auf und gliedert sie in das Feld des *Reverse Engineering* ein. Abschließend fasst Kapitel 7 die Arbeit zusammen.

Kapitel 2

Grundlagen

Dieses Kapitel differenziert den Begriff des *Reverse Engineering* und zeigt weitere Arbeiten zu Kaffeemaschinen sowie intelligenten Geräten auf.

2.1 Begrifflichkeiten

Der Titel dieser Arbeit lautet `Reverse Engineering eines Kaffeevoll-automaten`, aber hinter der Terminologie des Begriffs *Reverse Engineering* steckt ein weites Spektrum an Bedeutungen. E. J. Chikofsky und J. H. Cross differenzieren in ihrem Paper [CC90] die Begriffe *Forward Engineering*, *Reverse Engineering*, *Redocumentation*, *Design Recovery*, *Restructuring*, und *Reengineering*.

Zugrunde liegt ein Produkt, welches während seiner Entwicklung mehrere Lebenszyklen durchlaufen hat. Kassem A. Saleh beschreibt in seinem Buch [Sal09] ausführlicher Entwicklungsaktivitäten, wie die Anforderungsanalyse, das Design, die Implementation, die Tests, die Installation und den Einsatz. Während der Implementation nennt er Vorgehensmodelle der Softwareentwicklung, wie das Wasserfallmodell, den Prototypenbau, das Spiralmodell, den objektorientierten Ansatz, das inkrementelle und iterative Modell, sowie das agile Modell. Jede (Hardware- und) Softwareentwicklung durchläuft dabei, unabhängig vom Modell, die verschiedenen Entwicklungsaktivitäten. Das Voranschreiten zur nächsten Stufe, bis zur Fertigstellung des Produkts, stellt das *Forward Engineering* dar.

Der zweite Begriff, das *Reverse Engineering*, führt in die Gegenrichtung. Aus dem implementierten Produkt wird auf das Design, bzw. aus dem Design auf die Spezifikationen geschlossen. Dabei werden zum einen die Komponenten und ihr Zusammenspiel identifiziert, zum anderen wird aber immer eine höhere Abstraktionsebene, eine vorherige Stufe, rekonstruiert. Ein wichtiges Zitat, welches in Abschnitt 6.3 aufgegriffen wird, lautet: „Reverse engineering in and of itself does not involve changing the subject system or creating a new system based on the reverse-engineered subject system. It is a process of examination, not a process of change or replication.“[CC90]

Redocumentation arbeitet auf einer Ebene und bringt primär eine andere Darstellung. Das Paper nennt „dataflow“, „data structure“ und „control flow“ als Beispiele, die über Werkzeuge wie Diagramm Generatoren, Syntaxhervorhebung und Querverweis Generatoren erzeugt werden können. Das Kernziel sei es die Zusammenhänge und Ablaufpfade hervorzuheben.

Breiter angelegt ist das *Design Recovery*, das für die Designwiederherstellung externe Informationen, Schlussfolgerungen, und Unschärfelogik mit einbezieht, um sinnvolle Abstraktionen auf höherer Ebene zu identifizieren, welche nicht aus dem System selbst hätten gewonnen werden können.

Restructuring umfasst das Nachbauen einer Darstellung innerhalb einer Ebene. Funktionalität und Semantik bleiben erhalten, während die Darstellung umgestaltet wird. Als Beispiel wird die Umstellung von unstrukturiertem Spaghetti Code zu strukturiertem „goto“ freien Code genannt. Der Begriff umfasst auch Datenmodelle, Entwurfsmuster und Anforderungsstrukturen. Dabei genügt das Wissen über die Struktur, ohne die Bedeutung dahinter zu verstehen, beispielsweise können „if“ und „case“ Ausdrücke ineinander überführt werden, ohne zu verstehen wann welcher Fall eintritt. Normalerweise werden daher ohne Anpassung der Spezifikation keine Veränderungen vorgenommen. *Restructuring* ist oft eine Form der präventiven Instandhaltung.

Zuletzt wird der Begriff *Reengineering* oder auch „Renovation and Reclamation“ als nachträgliche, neu erstellte Form beschrieben. Dafür geht *Reverse Engineering* zum abstrakten Verständnis dem *Forward Engineering* oder *Restructuring* voraus. Beim *Reengineering* sind Anpassung der Spezifikation und Veränderungen durchaus möglich. Ähnlich zum *Restructuring* verändert das *Reengineering* die unterliegende Struktur, ohne die Funktionalität zu beeinträchtigen. Jedoch passiert es selten, dass beim *Reengineering* keine weiteren Funktionalitäten hinzugefügt werden. Damit ist der Begriff *Reengineering* allgemeiner gefasst als das *Restructuring*. Aber *Reengineering* ist kein Überbegriff für *Reverse Engineering* und *Forward Engineering*, nur weil es beides beinhaltet. Beide Disziplinen entwickeln sich unabhängig vom *Reengineering* weiter.

Auch K. A. Ingle [Ing94] grenzt das Reverse Engineering von anderen Konzepten ab. Das *Reengineering* wird, im Gegensatz zu dem *Reverse Engineering*, beschrieben als Neustrukturierung des Programmcodes ohne funktionale Änderungen am System. Diese Beschreibung passt zum dem Begriff des *Restructuring* aus dem Paper von E. J. Chikofsky und J. H. Cross [CC90].

Beim *Simultaneous Engineering* greifen während einer Produktentwicklung mehrere Arbeitsabläufe frühzeitig verzahnt ineinander, um einen Zeitgewinn gegenüber eines traditionellen Projektablaufs zu erzielen. Die schnelle und einfache Reproduzierbarkeit sei beispielsweise nicht das direkte Ziel des *Reverse Engineering*, sondern ein Ziel des *Simultaneous Engineering*, schreibt K. A. Ingle. Dennoch könne aber die Idee des *Simultaneous Engineering* in das *Reverse Engineering* aufgenommen werden.

Eine weitere Differenzierung findet zwischen dem *Hardware-* und dem *Software Reverse Engineering* statt. *Software Reverse Engineeging* verfolgt das Ziel aus dem Programm Code ein übergeordnetes Design oder Spezifikationsangaben zu extrahieren. *Hardware Reverse Engineeging* hingegen befasst sich mit dem Fertigungs- oder Produktionssystem. Das Buch von 1994 sah aber schon ein Zusammenwachsen beider Bereiche durch den Einsatz und die Einbettung von immer mehr Software in Hardwaresystemen kommen. Die Relevanz des Reverse Engineering würde hoffentlich weiter steigen, um keine vorhandenen Softwaresysteme neu bauen zu müssen. K. A. Ingle schreibt: „The aim of reverse engineering is to increase productivity through improved documentation.“[Ing94]

Diese Arbeit greift den Gedanken auf und versucht den Einfluss und die Bedeutung einzelner Speicherstellen für weitere Arbeiten zu dokumentieren.

2.2 Stand der Technik

Eine Bachelorarbeit der Universität Magdeburg zum Thema „Reverse-engineering a De'Longhi Coffee Maker to precisely bill Coffee Consumption“[Off18] behandelt eine De'Longhi Kaffeemaschine. Das Ziel ist es, den Verbrauch exakt zu bestimmen und das System damit um ein Abrechnungssystem zu erweitern. Dafür wird eine Microcontroller Unit (MCU) über das Serial Peripheral Interface (SPI) zwischen Master und Slave Einheiten geschaltet und Informationen werden aus dem prorietären De'Longhi Protokoll ausgelesen. Das Ergebnis der Arbeit ist unter anderem ein Verständnis über das interne Bus Protokoll der Maschine.

Ein Buch von E. Eilam [Eil05] stellt den Begriff des Reverse Engineering ähnlich wie zu Beginn in Abschnitt 2.1 dar, geht aber im Folgenden tiefer auf das Reverse Engineering von Low-Level Software ein. Ein Hauptaugenmerk des Buches liegt auf auf der sicherheitsrelevanten Seite sowie den Vorgänge und Lücken im Low-Level Bereich.

Diese Arbeit hingegen nutzt als Ansatz eine gegebene serielle Schnittstelle des Jura Kaffeevollautomaten mit einem ebenfalls proprietären Universal Asynchronous Receiver Transmitter (UART) Protokoll. Dabei wird aber nicht die Hardware geöffnet und beispielsweise die Firmware disassembliert und analysiert.

Soucek, Russ und Tamarit haben sich in ihrem Paper [SRT00] mit dem Aufbau eines Feldbusses befasst, um mehrere Küchengeräte zu vernetzen und über ein Gateway in das Internet zu bringen.

Diese Arbeit erstellt auf der Basis der Ergebnisse in Kapitel 5 eine Schnittstelle für einen Computer, der wiederum als Server mehrere Endgeräte anbinden kann.

Kapitel 3

Hardware und Software

Dieses Kapitel führt die gegebenen und verwendeten Hard- und Software Komponenten in dieser Arbeit auf und schafft die Grundlage für die eigene Implementierung. Am Ende dieses Kapitels kann der Kaffeevollautomat bereits angesteuert werden.

3.1 Der Kaffeevollautomat

Der „Jura Impressa S9“, siehe Abbildung 3.1(a), ist ein Kaffeevollautomat mit fünf Kaffeebezugstasten: Spezialkaffee, 1 große Tasse Kaffee, 2 große Tassen Kaffee, 1 kleine Tasse Kaffee und 2 kleine Tassen Kaffee. Auf der rechten Seite befinden sich Bedienelemente für heißes (Tee-)Wasser und Wasserdampf zum Milchaufschäumen. Der Kaffeevollautomat wird für seinen Betrieb direkt mit der Netzspannung versorgt.

3.1.1 Aufbau und Verkabelung

Hinter der linken Wartungsklappe an der Vorderseite des Kaffeevollautomaten befindet sich neben mehreren Menü-Tasten eine serielle Schnittstelle. Über ein eigenes UART Protokoll kann hierüber mit der Maschine kommuniziert werden.

Abbildung 3.1(b) illustriert die Pinbelegung. Die 5 Volt Leitung kann für ein autark laufendes Projekt genutzt werden. In dieser Arbeit bezieht der Arduino seine Versorgungsspannung über den am USB Kabel befindlichen Computer. Von `TX` nach `RxD` werden Befehle an den Kaffeevollautomaten verschickt. Auf der Rückrichtung von `TxD` nach `RX` werden Antworten des Kaffeevollautomaten gelesen. `GND` ist abschließend die gemeinsame Erdung und Bezugsleitung für die serielle Kommunikation. Auf der Abbildung kreuzen sich die Leitungen `RxD` und `GND` am Stecker auf Seiten des Kaffeevollautomaten. `GND` ist über eine schwarze Markierung gekennzeichnet.

Ein Arduino Uno übersetzt als „Man in the middle“ die bekannten Kommandos in das Format der Kaffeemaschine. Die serielle Kommunikation läuft über die Pins Nummer 12 und 13. Über eine weitere serielle Verbindung per USB lässt sich der Arduino ansteuern.

(a) Der ganze Kaffeevollautomat (b) Pinbelegung zum Arduino Uno

Abbildung 3.1: Der „Jura Impressa S9“ Kaffeevollautomat

Aus Sicht des Computers ist der Arduino ein Gerätelaufwerk unter `/dev/ttyACM0` mit einer Baudrate von 9600. Diese Zahl findet sich zu Beginn des Arduino Uno Skripts wieder.

3.1.2 Serielle Kommunikation

Diese Arbeit baut auf das „CoffeeMachine“ Projekt [Kö18] auf und nutzt das Arduino Skript als Grundlage der Kommunikation, ebenso werden Kommandos und erste Speicherstellen aus der Weboberfläche aufgegriffen. Das Arduino Skript kodiert die UART Kommandos von und zu dem Kaffeevollautomaten.

Die Kommandos werden zeichenweise ASCII Zeichen für Zeichen übertragen. Dafür wird ein an den Kaffeevollautomaten adressiertes Byte (ein ASCII-Zeichen), bestehend aus 8 Bits, in je 4 Bytes aufgeteilt. Die dritte und sechste Stelle der neuen Bytes repräsentieren je zwei Bits des ursprünglichen Bytes. Bei der Übertragung an den Kaffeevollautomaten werden die restlichen Bits mit Nullen aufgefüllt. Abbildung 3.2 veranschaulicht dies an dem ersten Byte des Einschaltbefehls, die entsprechende American Standard Code for Information Interchange (ASCII)-Kodierung ist in Abbildung A.4 der zweiten und dritten Spalte zu entnehmen. Die Kodierung der, von dem Kaffeevollautomaten kommenden, Bytes erfolgt analog. Laut „Protocoljura“[1] bestehen die irrelevanten Bits der ankommenden Bytes aus einer Null und weiteren fünf Einsen pro Byte.

[1] http://protocoljura.wiki-site.com/index.php/Protocol_to_coffeemaker

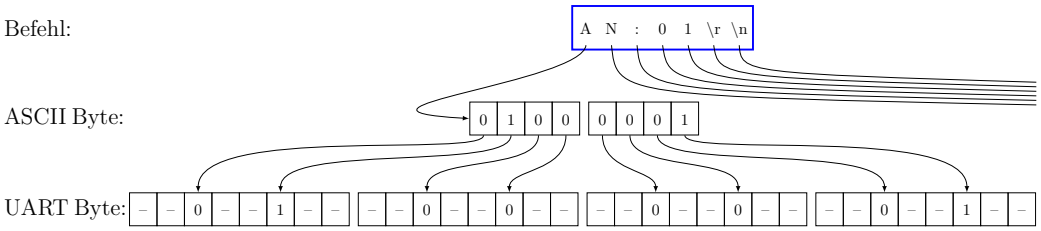


Abbildung 3.2: Umrechnung eines Befehls an den Kaffeevollautomaten

Vier Bytes kodieren auf diese Weise ein ASCII Zeichen und werden fortan als *Gruppe* bezeichnet. Zwischen jeder *Gruppe* gibt es eine Verzögerung von 8ms. Diese Verzögerung begrenzt hauptsächlich die Übertragungsgeschwindigkeit, was in Abschnitt 6.1.1 diskutiert wird.

3.1.3 Kommandos

Die Tabelle A.1 zeigt die Befehlsgruppen, sowie ausgewählte Befehle, die der Kaffeevollautomat versteht. Diese Befehle stammen aus dem „CoffeeMachine“ Projekt [Kö18].

Befehle beginnen entweder mit zwei Großbuchstaben, gefolgt von einem Doppelpunkt und i.d.R. einer zweistelligen Hexadezimalzahl, oder sie beginnen mit einem Fragezeichen gefolgt von i.d.R. zwei ASCII-Zeichen. Positionsnummern, egal ob Speicherposition, Betriebszustands-, Bezugstasten- oder Steuerungskomponenten-Nummer, werden durch zweistellige Hexadezimalzahlen repräsentiert und reichen von $0_{16} = 0_{10}$ bis $\mathrm{FF}_{16} = 255_{10}$.

Zwei Ausnahmen des ersten Typs sind z.B. `TY:` zum Abfragen des Maschinen Typs und `WE:00,01FF` zum Schreiben des Wertes $01\mathrm{FF}_{16} = 511_{10}$ an die Position 00. Eine Ausnahme des zweiten Typs ist z.B. `?D1DISPLAY␣` und `?D2␣␣TEST␣␣`. Der volle Umfang möglicher Displayzeichen, sowie deren Benutzung, wird in Abschnitt 5.4 ausgeführt.

Einige Befehle, die aus weiteren Projekten mit Maschinen der S-Reihe bekannt sind, sind in dieser „Jura Impressa S9“ leider nicht implementiert. Dazu zählen `IC:` zum Auslesen aller Eingaben, `FA:0A` eine unbelegte Bezugstaste, `CM:` für weitere Status Informationen, `CS:` für Sensor Informationen und `PM:` ein Befehl um Musik abzuspielen. Weitere Befehlsgruppen werden in der `README.rst` eines Github Repositories aufgeführt[2].

Die jetzt vorhandene Umgebung kann bereits über den seriellen Monitor der Arduino Integrated Development Environment (IDE) genutzt werden.

[2] https://github.com/PromyLOPh/juramote

3.1.4 Speicher

Über die Befehle `RE:<address>`, `RT:<address>` und `RR:<address>` sind Lesebefehle zu zwei Speichereinheiten des Kaffeevollautomaten bekannt: dem Electrically Erasable Programmable Read-Only Memory (EEPROM) und dem Ramdom-Access Memory (RAM).

EEPROM

Über den Befehl `RT:<address>`, siehe Tabelle A.1, lässt sich eine Zeile EEPROM Speicher abfragen. Die Adresse reicht von `0x00` bis `0xF0` in sechzehner Sprüngen.[3] Als Antwort erhält man hinter dem kleingeschriebenen Kommando eine Zeichenkette bestehend aus 64 Hexadezimalzahlen. Da immer zwei Hexadezimalzahlen ein Byte[4] repräsentieren, umfasst eine EEPROM Zeile **32** Bytes. Der gesamte EEPROM umfasst damit insgesamt **512** Bytes.

Über den Befehl `RE:<address>` können direkt Einheiten im EEPROM abgefragt werden. Die Adresse reicht von `0x00` bis `0xFF`. Die kleinste adressierbare Einheit, *das Wort*, besteht daher aus 4 Hexadezimalzahlen, also **2** Bytes[5].
Abbildung A.1(a) zeigt das Speicherschema des EEPROM.

RAM

Der Lesebefehl für eine RAM Zeile lautet: `RR:<address>`. Man ist aber nicht gezwungen, am Zeilenanfang zu starten. Die Adresse reicht hier von `0x00` bis `0xFF`. Als Antwort erhält man hinter dem kleingeschriebenen Kommando eine Zeichenkette bestehend aus 32 Hexadezimalzahlen, also **16** Bytes. Bei einer Adresse größer als `0xF0` gibt es einen Zählerüberlauf und es werden wieder die ersten Bytes zurück gegeben. Der gesamte RAM Speicher verfügt, ähnlich zu `RT` im EEPROM, über 16 Zeilen mit je 16 Bytes, also insgesamt **256** Bytes. Daraus ergibt sich, dass die kleinste Einheit, ein Byte, aus zwei Hexadezimalzahlen besteht.
Abbildung A.1(b) zeigt das Speicherschema des RAM.

3.2 Libraries und Frameworks

Diese Arbeit nutzt „libraries“ und „frameworks“. „Libraries“ sind thematisch gebündelte Funktionen und Routinen. Sie bieten eine Programmierschnittstelle (engl. Application Programming Interface) (API), die das eigene Programm ansprechen kann.

[3] Man ist daran aber nicht gebunden und kann wie im Folgenden beim RAM erklärt die Adressen von `0x00` bis `0xFF` nutzen. Dies wird in dieser Arbeit auf den EEPROM aber **nicht angewendet**.
[4] Ein Byte besteht aus 8 Bits mit einem Wertebereich von $0_{10} - 255_{10} = 00_{16} - \mathrm{FF}_{16} = 0\mathrm{x}00 - 0\mathrm{xFF}$.
[5] Zwei Bytes bestehen aus 16 Bits mit einem Wertebereich von $0_{10} - 65\,535_{10} = 00_{16} - \mathrm{FFFF}_{16}$.

„Frameworks“ bieten daneben ganze Programmiergerüste und Routinen. Sie rufen, wenn nötig, selbst vorgesehene Funktionen auf und folgen damit dem Paradigma des „Inversion of Control“.

Auf dem aktuellen Stand bieten beide nicht nur leicht zugängliche Aufrufe, sondern sind auch sicher und robust in ihrem Einsatzgebiet.

3.2.1 Serielle Kommunikation

Bei dem Versuch die Grätedatei direkt anzusprechen kam es zu Problemen, die in Abschnitt 6.1.2 erörtert werden. Für die zuverlässige serielle Kommunikation kommt daher eine geeignete Library zum Einsatz.

libserial

Das selbst entwickelte C++ Programm nutzt „liberial“[6] um sicher und zuverlässig über das Gerätelaufwerk mit dem Arduino (und letztlich dem Kaffeevollautomaten) zu kommunizieren. Diese Bibliothek bietet eine objektorientierte Schnittstelle für alle Portable Operating System Interface (POSIX) Systeme und ist unter Linux über die Paketverwaltung installierbar.

3.2.2 Speicher- und Austauschformat

Sowohl bei der Untersuchung des Speichers, als auch am Ende für das aufbereitete Ergebnis, werden Informationen gesammelt, verglichen und öffentlich zugänglich gemacht. Als beliebtes und flexibles Speicherformat wird in dieser Arbeit JavaScript Object Notation (JSON) verwendet. Es bietet viele Datenformate und ist unbegrenzt verschachtelbar. In dieser Arbeit werden hauptsächlich Zeichenketten, Zahlen, Unter-Objekte und -Arrays verwendet. Darüber hinaus kann es leicht vom C++ Programm, der Webseite, oder auch für viele weitere Projekte genutzt werden.

A. Kamilaris, A. Pitsillides und V. Trifa nennen in ihrem Paper [KPT11] alternativ noch Extensible Markup Language (XML) als oft verwendetes, aber recht langatmiges maschinenlesbares Dateiformat.

libjsoncpp

Die „libjsoncpp“[7] bietet dem C++ Programm eine Lese- und Syntaxanalyse-Funktion zum Aufnehmen eines JSON Ausdrucks und eine Möglichkeit, ein JSON Objekt kompakt zusammengefasst oder leserfreundlich aufgefächert in einen Ausgabestrom zu schreiben. Als

[6] https://github.com/crayzeewulf/libserial
[7] https://en.wikibooks.org/wiki/JsonCpp

Ausgabestrom sind reine JSON Dateien nach einer Speicherauszugs-Aufnahme oder die Standardausgabe im Gebrauch als API vorstellbar.

3.2.3 Webseite

Eine kleine Webseite soll am Ende die ausgelesenen Werte visuell anschaulich präsentieren. Die JavaScript-Bibliothek jQuery und das Framework Bootstrap helfen dabei, dies zügig und ansprechend umzusetzen.

Bootstrap

Bootstrap[8] ist ein von den Twitter Entwicklern begonnenes Projekt, um ursprünglich intern die Verwaltungswerkzeuge zu vereinheitlichen. Daraus ist ein ganzes System an Design-Elementen geworden, welches heute sehr populär ist. Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript (JS) werden zusammen eingesetzt und bieten Entwicklern eine Gitteranordnung für eine Mobil- / Desktop-Ansicht, Inhaltselemente wie Tabellen und Abbildungen oder auch Komponenten wie Steckkarten und Knöpfe.

jQuery

jQuery[9] ist eine freie JS-Bibliothek, die in der „slim“-Variante bereits über Bootstrap eingebunden ist. Um später wirklich mit dem Kaffeevollautomaten interagieren zu können, wird unter anderem Asynchronous JavaScript and XML (AJAX) aus dem vollständigen jQuery Paket benötigt (auch wenn daraus später ein synchrones JavaScript mit JSON wird).

Über das Content Delivery Network (CDN) eingebundene Paket stehen ab jetzt einfache Programmierschnittstellen zum Modifizieren der HTML Elemente, ein erweitertes Event-System, sowie AJAX-Funktionalitäten bereit.

JavaScript Cookie

JavaScript Cookie[10] ist eine kleine JavaScript-API. Sie vereinfacht es, Webbrowser-Kekse (Cookies) zum Ablegen persönlicher Einstellungen zu erzeugen, zu bearbeiten und zu löschen.

Intro.js

Zu guter Letzt folgt mit Into.js[11] noch eine Schritt-für-Schritt-Anleitung, die dem Seitenbenutzer eine kurze Einführung in den Aufbau und die Bedienung der Webseite geben soll.

[8] https://getbootstrap.com/
[9] https://jquery.com/
[10] https://github.com/js-cookie/js-cookie/
[11] https://introjs.com/

Kapitel 4

Methodik und Implementierung

Dieses Kapitel führt das Vorgehen und die Implementierung eines C++ Programms aus. Dadurch wird es möglich, Aussagen über den Speicher des Kaffeevollautomaten zu treffen. Immer wiederkehrende Arbeitsabläufe werden von diesem Programm übernommen und auf den unteren Ebenen automatisiert.

4.1 Vorgehen

Die Speicher des Kaffeevollautomaten, der EEPROM und der RAM, können zeilenweise ausgelesen werden. Der nötige Zugriff zum Auslesen kann auf zwei Arten erfolgen: direkt am Speicherstein auf der Hauptplatine oder seriell über die vorhandene UART Schnittstelle. Abschnitt 6.1.1 diskutiert die Vor- und Nachteile, die dazu geführt haben im Folgenden die Kommunikation über die UART Schnittstelle und mithilfe der *libserial*-Library erfolgen zu lassen. Dadurch kann die Speicherabfrage als Blackbox Modell, ähnlich zum Blackbox Testing in [Sal09], betrachtet werden, wo nur mit Ein- und Ausgaben des Kaffeevollautomaten gearbeitet wird, ohne dessen Programm zu kennen.

Unabhängig voneinander wurden beide Speicher untersucht. Ein eigens entwickeltes C++ Programm setzt die 16 Zeilen einer Speicherabfrage zu einem gesamten Speicherauszug zusammen. Intern werden dabei je zwei Hexadezimalzahlen in eine ganzzahlige Dezimalzahl umgerechnet und in einem Vektor abgelegt. Dadurch sind Position und Wert bekannt.

Abbildung 4.1 visualisiert den Arbeitsablauf, der zur Bestimmung der Speicherstellen in dieser Arbeit angewandt wurde. Nach dem ersten Schritt ganz links wird nun von Hand eine möglichst elementare Veränderung an dem Kaffeevollautomaten vorgenommen, um gezielten Aktionen die Werteänderungen bestimmter Speicherzellen zuordnen zu können. Die angewandten Veränderungen werden in Abschnitt 4.1.1 beschrieben.

Nach einer erneuten Aufnahme eines Speicherauszugs können diese beiden nun verglichen werden. Das Programm iteriert durch den Vektor und vergleicht die Speicherwerte an der glei-

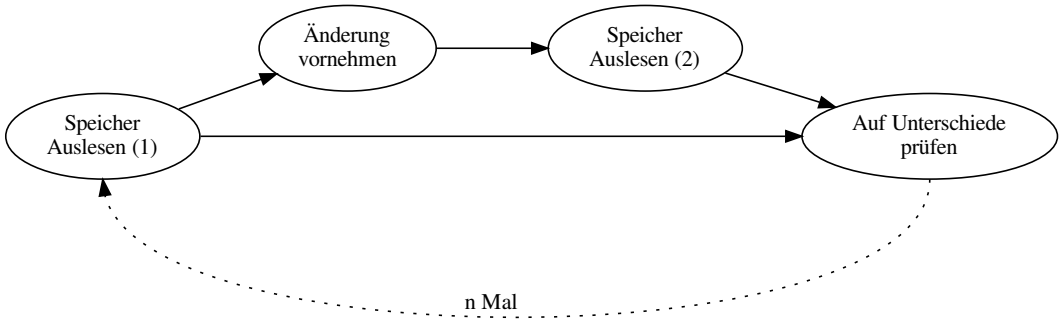


Abbildung 4.1: Arbeitsablauf zur systematischen Untersuchung des EEPROMs und RAMs

chen Position. Bei Ungleichheit existiert ein Unterschied, der festgehalten wird. Dieser Ablauf wurde auf alle Einstellmöglichkeiten und Funktionen des Kaffeevollautomaten angewandt.

Bei dem RAM, im Gegensatz zum EEPROM, fiel jedoch auf, dass sich mehrere Speicherwerte auch ohne eine vorgenommene Veränderung änderten. Deshalb wurden viele Speicherauszüge im Ruhezustand aufgenommen und die sich regelmäßig ändernden Bytes ausgeschlossen. Zu Beginn der Ergebnisse über den RAM in Abschnitt 5.2 sind diese aufgeführt. Dennoch gab es gelegentliche Unregelmäßigkeiten, sodass das Vorgehen für den RAM um eine geschlossene Rückrichtung ergänzt wurde. In der Regel wurden pro RAM Funktion $n = 3$ Durchläufe vorgenommen und die gemeinsame Schnittmenge der Veränderungen bestimmt.

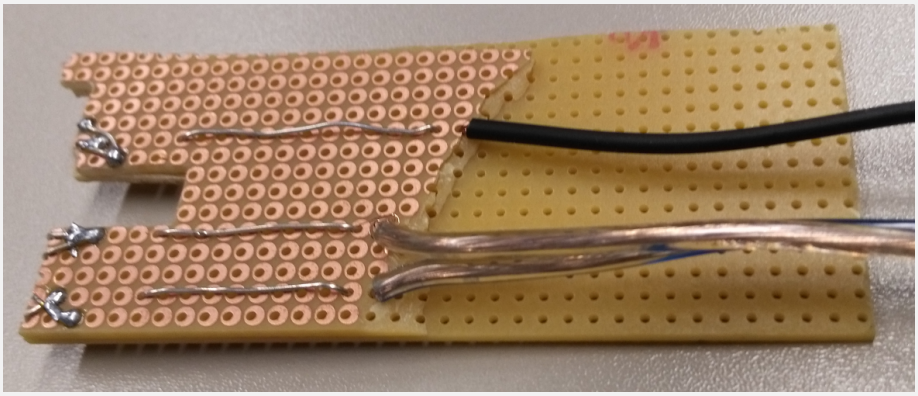
Bei einem zweiten Vorgehen wurde gezielt der EEPROM an bekannten Speicherstellen beschrieben. Hintergrund war die unbekannte Position bzw. später die Zusammensetzung des Bezüge-Zählers im Einstellungsmenü des Kaffeevollautomaten. Abschnitt 4.1.2 führt dies im Folgenden aus.

4.1.1 Veränderungen zur Bestimmung der Speicherstellen

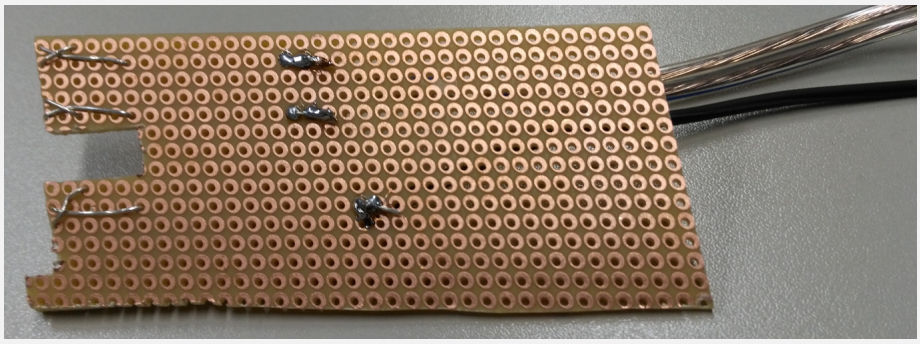
Für den EEPROM war es hilfreich, systematisch durch das Handbuch der Maschine zu gehen und zum Beispiel immer eine Menüoption zu variieren. Bei einer Skala (wie z.B. beim Kaffeepulver) hat es ausgereicht, den niedrigsten, den höchsten und den Standard-Wert im Speicher zu bestimmen; deckte sich die einstellbare Anzahl an Abstufungen mit der Differenz der Werte im Speicher, ließ sich auf die verbleibenden Werte schließen. Auch im normalen Gebrauch fiel unter anderem ein Einschaltzähler im EEPROM auf.

Für den RAM ging es mit den gezielt auslösbaren Statusmeldungen los. Das Anheben des Wassertanks oder das Abziehen der unteren Schale löste eine Warnmeldung auf dem Display aus und wurde auch im RAM vermerkt. Ein zu hoher Wasserstand in der Schale konnte ohne Wasser mit einer eigenen Platine vorgegaukelt werden. Abbildung 4.2(a) zeigt die Fassung an der Rückwand innerhalb des Kaffeevollautomaten für die Schale. Abbildung 4.2(b) und

(a) Dargestellt ist die Fassung im Kaffeevollautomaten.



(b) Zu sehen ist die Oberseite des Ersatzes für die Schale.



(c) Zu sehen ist die Unterseite des Ersatzes für die Schale.

Abbildung 4.2: Ersatz der Schale des Kaffeevollautomaten

4.2(c) visualisieren die Platine, an der leicht von außen mittels Krokodilklemmen die Kontakte im Inneren kurzgeschlossen werden konnten. Am schwierigsten sind komplexe Abläufe, wie die Zubereitung eines Kaffees. Hier konnte nur mehrfach zu gezielt gleichen oder ungleichen Zeitabständen der zweite Speicherauszug angestoßen werden.

Abschließend war auch für das Display ein kleines Programm nötig, um die volle Funktionsvielfalt festzustellen. Abschnitt 4.3 führt dies später aus.

In Abschnitt 6.2 wird die Aussagekraft dieser Ergebnisse noch thematisiert.

4.1.2 EEPROM gezielt beschreiben

Ziel war es die unbekannte Position bzw. später die Zusammensetzung des Bezüge-Zählers im Einstellungsmenü des Kaffeevollautomaten zu entziffern. Dafür wurde die angezeigt Zahl in eine Hexadezimalzahl umgerechnet und im Speicherauszug des EEPROM des Kaffeevollautomaten gesucht. Da die Zahl eine Stromunterbrechung überstand und eine Änderung des einzigen übereinstimmenden Wertependants in Wort 15_{16} keine Änderung der Ausgabe hervorrief, begann die Suche an weiteren bekannten Speicherstellen im EEPROM. Bekannte Zähler in Wörtern wie 00_{16}, 01_{16}, 02_{16}, $0D_{16}$, $0E_{16}$ oder mehreren weiteren wurden einzeln auf sehr hohe Werte geändert. Manche Änderungen veränderten den Bezüge-Zähler oder lösten Warnmeldungen an dem Kaffeevollautomaten aus.

Es folgte die Erkenntnis, dass sich der Bezüge-Zähler aus mehreren Zählern zusammen setzt. Der Trester Füllstand in der Schale wird nicht gemessen, sondern im Betrieb gezählt, um den Füllstand zu erfassen. Auch die Reinigungsankündigung beruht auf Zählerständen über Kaffeebezüge und Spülungen. Die Wasser-Durchflussmenge des Filters wird ebenfalls erfasst und ab einem festen Wert eine Warnmeldung zum Wechseln ausgegeben.

Das gezielte Eingreifen in die Zählerstände ließ es zu, diese Grenzwerte exakt zu bestimmen. Die Ergebnisse befinden sich in Abschnitt 5.1.1.

4.2 Das C++ Programm „./JuraCoffeeMemory“

Für diese Arbeit wurde ein C++ Programm entwickelt, das die Kommunikation und die Aufschlüsselung der Antworten übernommen hat.

4.2.1 Makefile

In dem Projekt Ordner befinden sich auf der Hauptebene mehrere `CPP`- und `HPP`-, eine `H`- sowie eine `Makefile`-Datei. Sind die in der `Readme.md` genannten Abhängigkeiten und Voraussetzungen erfüllt, kann das Projekt mit dem Befehl `make` ohne Zusatzangaben kompiliert werden. Am Ende sollte dann auf der Hauptebene eine ausführbare Datei namens `./JuraCoffeeMemory` herauskommen. Es werden neben Objekt-Dateien auch noch einige

weitere Tools im Unterordner `./tools/` erstellt, auf die später in Abschnitt 4.3 eingegangen wird.

Der Befehl `make clean` entfernt die Objekt-Dateien; mittels `make dist-clean` werden ebenfalls die ausführbaren Programme entfernt und der Projekt-Ordner wird in seinen Ursprungszustand zurück versetzt.

4.2.2 Interaktives Menü

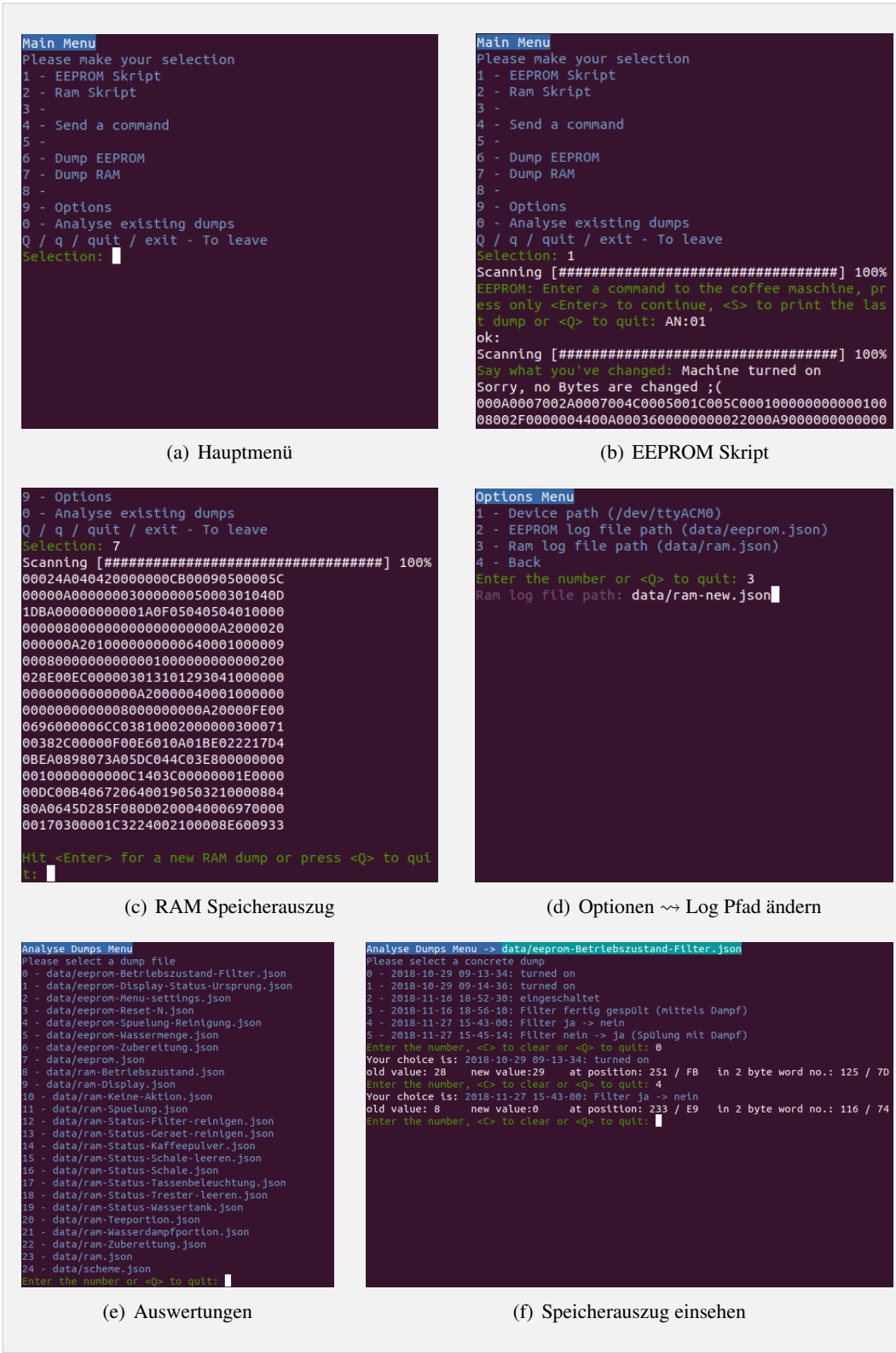
Startet man in einem Terminal das Hauptprogramm `./JuraCoffeeMemory` wird das Hauptmenü angezeigt. Abbildung 4.3(a) visualisiert diesen Zustand. Das Listing 4.1 zeigt schematisch alle Menüoptionen.

```
Main Menu
|-- 1: EEPROM Skript
|-- 2: Ram Skript
|-- 4: Send a command
|-- 6: Dump EEPROM
|-- 7: Dump RAM
|-- 9: Options
|   |-- 1: Device path (/dev/ttyACM0)
|   |-- 2: EEPROM log file path (data/eeprom.json)
|   |-- 3: Ram log file path (data/ram.json)
|   `-- 4: Back
|-- 0: Analyse existing dumps
|   |-- Analyse Dumps Menu
|   |   |-- data/xxx.json
|   |   |   `-- Dumps
|   |   |-- C: Clear window
|   |   `-- Q: Quit
|   `-- Q: Quit
`-- Q / q / quit / exit: To leave
```

Listing 4.1: Menü-Baum ./JuraCoffeeMemory

EEPROM / RAM Skript

Über die Nummer `1` startet aus dem Hauptmenü das *EEPROM Skript*, über die `2` das *RAM Skript*. Siehe hierfür Abbildung 4.3(b). In beiden Fällen wird das Vorgehen in Abschnitt 4.1 angewendet. Das Skript erzeugt initial einen Speicherauszug und merkt sich diesen. Der Anwender kann sich diesen über die Eingabe `S` ausgeben lassen oder an diesem Punkt über `Q` das Skript beenden. Um fortzufahren kann nun eine Aktion vorgenommen oder direkt ein Befehl abgesetzt werden. Das Skript erstellt im Anschluss einen weiteren Speicherauszug und fragt nach einem Kommentar zu den vorgenommenen Änderungen. Nach einer Texteingabe vergleicht das Programm die Speicherauszüge und hält Unterschiede in der Standardausgabe



(a) Hauptmenü

(b) EEPROM Skript

(c) RAM Speicherauszug

(d) Optionen ⇝ Log Pfad ändern

(e) Auswertungen

(f) Speicherauszug einsehen

Abbildung 4.3: Interaktives Menü des C++ Programms „./JuraCoffeeMemory“

und der JSON Ausgabedatei fest. Ebenso werden in der JSON Datei jedes Mal beide Speicherauszüge mit Zeitstempel und Kommentar abgelegt. Das Skript verwirft den vorletzten Speicherauszug und bietet die Fortführung mit dem Anwender Menü an.

Einen Befehl abschicken

Über die Nummer `4` startet aus dem Hauptmenü eine immer wiederkehrende Eingabeaufforderung, in der Befehle aus der Tabelle A.1 an den Kaffeevollautomaten verschickt werden können. Hierüber kann schnell ein Funktionstest der Verbindung zum Kaffeevollautomaten vorgenommen werden. Über `Q` kehrt der Anwender in das Hauptmenü zurück.

Speicherauszug

Über die Nummer `6` bzw. Nummer `7` kann ein einzelner Speicherauszug des EEPROMs bzw. des RAMs erstellt werden, der unverzüglich in der Standardausgabe ausgegeben wird. Abbildung 4.3(c) zeigt einen Speicherauszug des RAMs. Diese Funktion ist ein Teil des EEPROM bzw. RAM Skripts. Es fragt aber nicht nach einem Kommentar und fungiert an dieser Stelle rein lesend. Mithilfe des Hilfsprogramms zum Formatieren eines Speicherauszugs aus Abschnitt 4.3.1 kann man den Speicherauszug um Positionsangaben aufwerten und Speicherwerte dadurch kontrollieren.

Optionen

Über die Nummer `9` gelangt der Anwender in das Optionsmenü, wie in Abbildung 4.3(d) dargestellt. Es können für den aktuellen Prozess der Pfad zum Gerätelaufwerk des Arduinos und die Ausgabedateien für das EEPROM und RAM Skript angepasst werden. Nach der Eingabe eines neuen Pfades wird dieser sofort übernommen und angezeigt. Über die Nummer `4` kehrt der Anwender in das Hauptmenü zurück.

Analyse aufgenommener Speicherauszüge

In der JSON Datei werden im Knoten `"data"` die Wertänderungen hinter jedem entsprechenden Byte inklusive Kommentar festgehalten. Nach mehreren Durchläufen des EEPROM / RAM Skripts können hier schon direkt einige Zugehörigkeiten abgelesen werden.

Das C++ Programm bietet über die Nummer `0` auch die Option Unterschiede einzelner Aufnahmen über das Skript erneut auszugeben. Zuerst werden sämtliche JSON Dateien im Unterordner `data/` aufgelistet, siehe Abbildung 4.3(e). Nach der Wahl einer Datei wird eine Übersicht aller darin enthaltener Speicherauszüge ausgegeben. Durch die Wahl der entsprechenden Nummer werden der Kommentar und die veränderten Werte pro Byte aufgeschlüsselt.

In Abbildung 4.3(f) ist dies für zwei Speicherauszüge dargestellt, die jeweils nur eine Änderung beinhalten.

Angaben in der Aufschlüsselung der Veränderungen

Sowohl im EEPROM / RAM Skript als auch in der Analyse aufgenommener Speicherauszüge können Veränderungen eingesehen werden. Das erste Beispiel aus Abbildung 4.3(f) lautet:

```
Your choice is: 2018-10-29 09-13-34: turned on
old value: 28  new value:29  at position: 251 / FB  in 2 byte word no.: 125 / 7D
```

Listing 4.2: Beispiel einer Abweichung zweier Speicherauszüge

Diese Ausgabe ist für den EEPROM und RAM prinzipiell gleich, die relevanten Stellen sind aber leicht verschieden. Zu Beginn erscheint die Uhrzeit und der bei der Aufnahme eingegebene Kommentar. Pro Zeile folgt danach eine Änderung. In diesem Fall änderte sich der Wert von `28` auf `29`. Diese Aufnahme kommt aus dem EEPROM, für den die Befehle auf den 2 Byte *Wort* Adressen beruhen. Möchte man diesen Wert verändern, ist die letzte Spalte mit der hexadezimalen Zahl von Bedeutung, zum Beispiel `WE:7D,xxxx`.

Für den RAM ist die normale Position eines Bytes von Bedeutung, hier würde sich der dezimale Wert `29` an der Position `FB` befinden. Über den Befehl `RR:FB` bekäme man die `29` als anfängliche Hexadezimalzahl zurück gegeben.

Farbcodierungen

Die verwendeten Farben sollen dem Anwender Orientierung und Struktur bei der Benutzung verschaffen. Normale Ausgaben und Benutzereingaben erscheinen in weißer Schrift. Weiße Schrift auf blauem Grund stellt Menü-Überschriften dar. In blauer Schrift sind Menüs dargestellt. Grüne Schrift vor der blinkenden Eingabemarke fordert zur Eingabe auf. Der Text beschreibt valide Eingaben. Abschließend beschreibt roter Text einen Fehler. Die aufgetretene Position wird in weißer Schrift auf rotem Grund hervorgehoben.

4.2.3 Die API nach außen

Das Hauptprogramm kann auch über vier Parameter aufgerufen werden. Die Ausgaben sind dann im fehlerfreien Fall farblose JSON Zeichenketten oder einfacher Text in die Standardausgabe.

Mögliche Fehlermeldungen sind für Entwickler in der `Readme.md` dokumentiert. Die Fehlerausgabe erfolgt wie im interaktiven Menü mit Farben. Für Entwickler bietet sich daher der Rückgabewert des Programms an.

./JuraCoffeeMemory command Von der Standardeingabe wird eine Zeichenkette erwartet, die an den Kaffeevollautomaten weiter gegeben wird. Mögliche Eingaben sind die Befehle aus der Tabelle A.1. Die Antwort des Kaffeevollautomaten erscheint als einfacher Text.

./JuraCoffeeMemory eepromWrite Bei diesem Aufruf wird von der Standardeingabe ein JSON Objekt erwartet. Alle dort genannten Bezeichnungen werden abgearbeitet und die neuen Werte im Speicher des Kaffeevollautomaten hinterlegt. Auf der obersten Ebene müssen sich aus `EEPROM_Status::getEntriesEEPROM()` bekannte Bezeichnungen befinden, die ein Unterobjekt mit der Bezeichnung `value` sowie je einen ganzzahligen Wert beinhalten. Listing 4.3 veranschaulicht dies an einem Beispiel.

Treffen die Bedingungen zu, wird ein Schreib-Befehl an den Kaffeevollautomaten gesandt. Die Antworten werden in einer Zeichenkette mit der verwendeten Bezeichnung und dem Text „###“ als Abstandshalter festgehalten und auf der Standardausgabe ausgegeben. Die Antwort auf das Beispiel Listing 4.3 steht in Listing 4.4.

```
{
  "powder_quantity_special_coffee": {
    "value": 11
  },
  "water_quantity_special_coffee": {
    "value":  380
  }
}
```

Listing 4.3: Beispiel einer JSON Eingabe für./JuraCoffeeMemory eepromWrite

```
ok: powder_quantity_special_coffee###ok: water_quantity_special_coffee###
```

Listing 4.4: Antwort auf das Beispiel der JSON Eingabe

./JuraCoffeeMemory eeprom Nach ungefähr 5 Sekunden erhält man die aktuellen Einstellungen und Zählerstände des EEPROM in einem kompakten JSON Objekt. Im Gegensatz zum EEPROM Skript werden nur die nötigen Speicherstellen abgefragt. Die Bezeichnung der Speicherstellen, die auch der Name eines JSON Eintrags ist, befindet sich mit der zugehörigen Adresse in `EEPROM_Status::getEntriesEEPROM()`. Eine Adresse besteht aus der Position eines Wortes sowie einer oder beiden Bytes. Von Hand sind dort zum Teil weitere Hintergrundinformationen, wie Standardwerte über die `[N]`-Taste des Kaffeevollautomaten, minimale und maximale Schranken, der Wert für einen deaktivierten Zustand oder überhaupt zulässige Werte, hinterlegt.

./JuraCoffeeMemory ram Ähnlich zum Abfragen des EEPROM wird hier aber der RAM in ungefähr 3 Sekunden an den entscheidenden Stellen ausgelesen. Die Assoziation einer Bezeichnung mit ihrer Speicherposition geschieht in `RAM_Status::getEntriesRAM()`. Hier wird ein Byte und ggf. mehrere zusammenhängende Bits als ein Eintrag abgelegt. Als Ausgabe erfolgt ein kompaktes JSON Objekt.

4.2.4 Technische Umsetzung

Abbildung 4.4 zeigt schematisch den Zusammenhang der C++ Klassen. Die Hauptdatei ist die `JuraCoffeeMemory.cpp`, aus der später auch das ausführbare Programm `./JuraCoffeeMemory` erzeugt wird. Sie steht ganz oben in Abbildung 4.4. Die `main`-Funktion fängt die Argumente des Programm-Aufrufs ab und übergibt entsprechend an die `EEPROM_Status` bzw. `RAM_Status` Klasse für die API oder leitet den Anwender selber durch das interaktive Menü. Die *API*-Box in Abbildung 4.4 repräsentiert die API mit den beiden Klassen.

Beim Aufruf des EEPROM Skripts wird die `eeprom`-Funktion betreten, für den RAM analog die `ram`-Funktion. Diese Funktionen bauen zu Beginn eine serielle Verbindung zum Arduino auf und blockieren damit alle anderen Prozesse. Danach wird eine Instanz der gleichnamigen Klasse erzeugt. Die Klasse ist in der Box namens *memory* in Abbildung 4.4 aufgeführt. Der Konstruktor liest dabei den Speicher aus und füllt Zeile für Zeile den `raw` Vektor in der Elternklasse `Storage`. Abschließend veranlasst der Konstruktor die Umrechnung der hexadezimalen Zeichenketten in einen langen Vektor aus einzelnen Zahlen-Werten, die in `bytes` abgelegt werden. Die Instanz wird mit dem Präfix `old` in der Ursprungsfunktion abgespeichert.

Dem Anwender wird, wie in Abschnitt 4.2.2 beschrieben, ein Menü präsentiert. Im Falle einer Eingabe in Form eines Befehls, wird diese über die serielle Verbindung gesendet und blockierend eine Antwort abgewartet, die dem Anwender ausgegeben wird. Gemäß dem Vorgehen in Abschnitt 4.1 wird dann ein weiterer Speicherauszug erstellt und die neue Instanz mit dem Präfix `new` abgespeichert. Abbildung 4.4 veranschaulicht, dass `JuraCoffeeMemory.cpp` und die Speicherinstanz gemeinsam auf die serielle Verbindung über die `SerialConnection` Klasse zugreifen können.

Die Klasse `Storage` bietet zum byteweisen Abgleich eine Methode mit dem Namen `diffBytesWith()`. Argumente sind eine weitere Instanz mit der verglichen wird, ein Vektor mit ausgeschlossenen Bytes, ein Merker ob in die JSON Datei geschrieben werden soll und ob bereits ein Kommentar vorliegt. Das EEPROM / RAM Skript nutzt ausschließlich das erste obligatorische Argument, indem die `new` Instanz die Methode und im Argument die `old` Instanz aufruft. Die `ram`-Funktion nutzt an dieser Stelle zusätzlich das erste optionale Argument und exkludiert die im Vorgehen in Abschnitt 4.1 angesprochenen schwankenden Bytes,

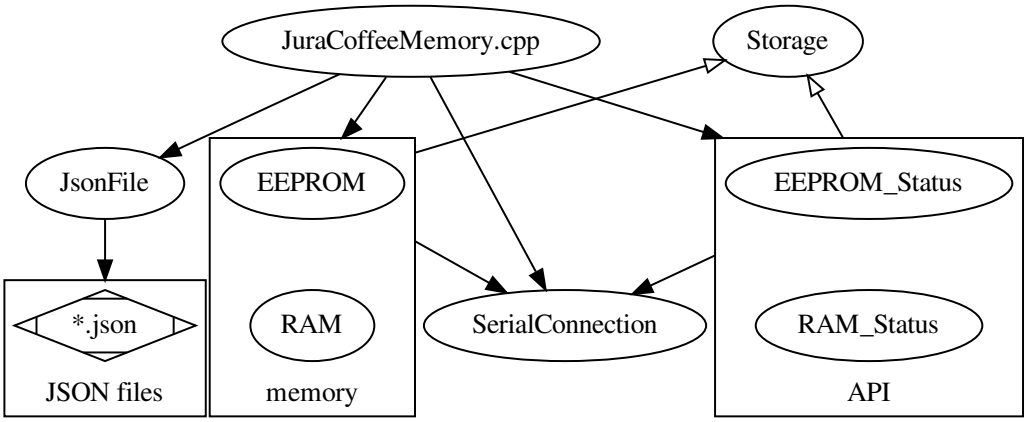


Abbildung 4.4: Schematischer Zusammenhang und Gruppierung der C++ Klassen

die in den Ergebnissen in Abschnitt 5.2 explizit genannt werden. Die nachträgliche Analyse wertet die Unterschiede erneut aus, nutzt aber die letzten beiden optionalen Argumente, um ein erneutes Beschreiben der JSON Datei mit den bereits vorliegenden Ergebnissen zu unterdrücken. Der Kommentar als letztes Argument wird in diesem Fall einfach nur ausgegeben, statt den Anwender nach einem Neuen zu fragen.

Zurück in der `eeprom-` / `ram`-Funktion der `JuraCoffeeMemory.cpp` wird die `old` Instanz freigegeben und die aktuelle `new` Instanz zur neuen `old` Instanz umbenannt, damit der Anwender das Verfahren wiederholen kann.

Die Klasse `JsonFile`, zu sehen in Abbildung 4.4, steht zum Einlesen von JSON-Dateien und -Zeichenketten, zum Bearbeiten des Objekts und der Ausgabe in einen Ausgabestrom bereit. Sie wird ebenfalls zur Aufbereitung der JSON-Speicherauszüge in eine geordnete Struktur zur Weiterverarbeitung verwendet.

Besonderheiten der technischen Umsetzung / Implementierung

Die `EEPROM`- und die `RAM`-Klasse unterscheiden sich durch die Größenangaben zum Speicher und dem zum Abfragen benötigten Kommando in der entsprechenden `hpp`-Datei. Vieles wurde in die gemeinsame Elternklasse `Storage` ausgegliedert.

Die `SerialConnection` Klasse setzt das *Singleton-Entwurfsmuster* um[1]. Es wird automatisch eine Instanz der Klasse erstellt und ist über eine statische Methode abrufbar. Dies stellt sicher, dass es nur eine serielle Verbindung zur Zeit gibt. Darüber hinaus wird beim `connect()` Methodenaufruf eine Dateisperre auf die serielle Gerätedatei erzeugt. Dies wiederum garantiert, dass nur eine Prozessinstanz des Programms zur Zeit die Verbindung hält und ungestört bleibt. Bis zum Aufruf von `disconnect()` bleibt die Sperre aufrecht erhalten.

[1] Die `JsonFile` Klasse folgt ebenfalls dem Singleton-Entwurfsmuster.

Zusätzlich prüft die `connect()` Methode die Übertragung nach dem Verbindungsaufbau mit dem Testkommando `TY:` und erwartet von dem Kaffeevollautomaten die eingestellte Antwort `ty:E1300 CAPU 3\r`. Beim Senden und Empfangen übernehmen die entsprechenden Methoden die Handhabung des Wagenrücklaufs und des Zeilenumbruchs am Ende der Kommandos. Der Anwender oder Entwickler kann dadurch die Kommandos aus Tabelle A.1 direkt anwenden.

Die Klassen `EEPROM_Status` und `RAM_Status` besitzen auch eine Besonderheit. Die gleich lautende `scan()` Methode fragt nicht nur die nötigen Speicherstellen ab, sondern vermerkt in dem Vektor `known_bytes` ebenfalls deren Position. Wenn die Methode `getEntriesEEPROM()` bzw. `getEntriesRAM()` um weitere Speicherstellen ergänzt wird, ist dadurch sichergestellt, dass auch nur tatsächlich abgefragte Speicherstellen und keine konstanten Nullen zurückgegeben werden. Andernfalls erhält der Entwickler eine Fehlermeldung. Weitere Einzelheiten sind in der `Readme.md` im Projektordner festgehalten.

4.3 Weitere kleine Tools

Im Unterordner `./tools/` befinden sich nach dem Aufruf von `make` drei kleine Programme.

4.3.1 Formatieren eines Speicherauszugs

Ruft man `./tools/formate-dump` auf, kann man dem Programm eine Zeichenkette aus 1024 oder 512 Hexadezimalzeichen übergeben. Das Programm erkennt an der Größe EEPROM und RAM Eingaben und wertet den Speicherauszug um Angaben wie die Speicherposition und Dezimal-/Hexadezimalzahlen Formate auf.

4.3.2 Display

Um den verfügbaren Zeichensatz des Displays in der oberen linken Ecke an der Front des Kaffeevollautomaten zu bestimmen, wurde ein kleines Extraprogramm entwickelt. Anlass waren Sonderzeichen und die deutschen Umlaute, die sich über die gegebenen Befehle (`?D0`, `?D1xxx`, `?D2xxx`) nicht einfach auf das Display bringen ließen. Das Programm `./tools/display` iterierte dafür systematisch über die Zahlen von 0 bis ungefähr 260 und gab einen Befehl zum Anzeigen der Dezimalzahl sowie des dazugehörigen ASCII-Zeichens an den Kaffeevollautomaten. Dies umfasst das einfache und erweiterte ASCII Alphabet sowie einige weitere Zahlen bzw. Zeichen. Dabei stellte sich nur ein mittlerer Block des einfachen ASCII-Alphabets als interessant heraus, der jetzt eingegrenzt von eben diesem Programm durchlaufen wird. Ein weiteres Programm namens `./tools/display-screen-saver` veranschaulicht die damit verbundenen Möglichkeiten.

Die gegebenen Befehle über die serielle Verbindung `?D0`, `?D1xxx` und `?D2xxx` verändern nur den Standardtext „Kaffee bereit“. Andere Texte, wie das Programmmenü oder Warnmeldungen, überlagern den Standardtext mit den einprogrammierten Texten aus den vorhandenen Sprachen.

Möchte man die Ausgaben des ersten Programms nachstellen, ist zu beachten, dass der Kaffeevollautomat eingeschaltet sein muss und auch an die erste Displayzeile ein Ausgabebefehl gegangen sein muss, damit die Ausgabe sichtbar wird. Das Programm verändert dann die zweite Zeile. Folgende Befehlseingaben werden empfohlen:

1. `cd JuraCoffeeMemory/`
2. `make`
3. `./JuraCoffeeMemory`
4. `4` (Senden eines Kommandos)
5. `AN:01` (Maschine einschalten)
6. `?D1xxx` (Etwas Text an die erste Displayzeile)
7. `<Strg>+D` (Verlassen des Programms)
8. `./tools/display`

4.4 Die Webseite

Die Webseite ist zu sehen in Abbildung 4.5. Nach ein paar Hinweisen kann im oberen Bereich der einzelne Kaffee sowie die gesamte Maschine konfiguriert werden. Für jede Kaffeeart ist es möglich, sich seine eigene Konfiguration in einem lokalen Webbrowser-Keks (Cookie) abzuspeichern und komfortabel in den Speicher des Kaffeevollautomaten zu schreiben um sich anschließend einen Kaffee zubereiten zu lassen. Der Keks kann auch wieder entfernt werden oder sich auf die Standardwerte zurücksetzen lassen. Prinzipiell kann hierüber der ganze bekannte EEPROM überschrieben werden.

Im nächsten Absatz sind Statusinformationen aus dem RAM einsehbar. Einige Meldungen werden am Bild des Kaffeevollautomaten visualisiert. Der Status kann über `Refresh` aktualisiert werden. Einfache „ja“ oder „nein“ Meldungen werden durch eine grünes „on“ oder rotes „off“ präsentiert. Informationen, die sich über mehrere Bits erstrecken, werden mit ihrem entsprechenden Zahlenwert dargestellt.

Im letzten Abschnitt befinden sich Zählerstände und Prozent-Anzeigen, die verdeutlichen wie weit es noch bis zur nächsten Warnmeldung, wie Trester leeren, Maschine reinigen oder Filter wechseln, ist.

Wenn am Computer der Mauszeiger auf einer Zahl oder Einstellung zum Finger wird, öffnet sich mit einem Klick ein Fenster, in dem der Wert verändert werden kann. Nach Möglichkeit werden der Standardwert, der Wert zum Deaktivieren und weitere Hinweis-Texte angeboten.

Unten rechts auf der Seite kann über `Command` ein Kommando aus der Tabelle A.1 an den Kaffeevollautomaten abgesetzt werden. Eine Liste bietet viele Vorschläge mit dazugehörigen Bezeichnungen. Rechts daneben aktualisiert ein Klick auf `Refresh` die Statusinformationen aus dem RAM. Ganz rechts in der Ecke kann eine Tour durch die Bedienung der Seite gestartet werden.

4.4.1 Technische Umsetzung

Die Webseite befindet sich im Projektunterordner `./website/`. Die Datei `data.php` nimmt Parameter entgegen und ruft damit das C++ Programm mit evtl. Datenübergaben auf. Die Antwort des C++ Programms wird in einem JSON-Objekt verpackt und der Webseite zurück gegeben. Bei Problemen wird auch die Fehlernummer (der Rückgabewert des Programms) einschließlich der Meldung leicht zugänglich in einem JSON-Objekt zurück gegeben.

Die für den Benutzer ersichtliche Webseite besteht primär aus den Dateien `index.html`, `script.js` für die Interaktion und `style.css` für das individuelle Layout. Weitere eingebundene Dateien sind das Favicon im Unterordner `favicon/`, Bilder im Unterordner `img/`, `intro.min.js` und `intro.min.css` für die Tour durch die Webseite. `offline_eeprom.json` und `offline_ram.json` bieten Beispielwerte für eine Offline-Funktion, in der die Webseite ersichtlich wird, ohne eine Verbindung zu dem Kaffeevollautomaten zu haben.

Das HTML Dokument der Webseite besteht aus vielen leeren Inhaltselementen, die im Attribut `id="..."` Einträge haben, die mit den Bezeichnungen der JSON Elemente der Ausgabe von der API des C++ Programms übereinstimmen. Beim Seitenaufbau wird der EEPROM abgefragt. Wenn die Antwort nach ungefähr 5 Sekunden vorliegt wird der RAM abgefragt und anschließend nach ungefähr weiteren 3 Sekunden für jeden JSON Eintrag der Wert in das gleichnamige HTML Inhaltselement eingetragen.

Eine Ausnahme bildet die Temperatur, die für alle drei Kaffeegrößen gilt, aber im Dokument drei verschiedene ID-Bezeichnungen besitzen muss. Ebenso ist etwas Handarbeit bei der Ausgabe der Prozent-Anzeigen vonnöten.

Das bei einem Klick aufgehende Fenster zum Bearbeiten von Werten aus dem EEPROM arbeitet ebenfalls über die ID-Bezeichnung und entnimmt weitere Informationen aus dem JSON Objekt welches zuletzt abgefragt und abgespeichert wurde.

Abbildung 4.5: Webseite zur Kontrolle und Steuerung des Kaffeevollautomaten als Visualisierung der API.

Kapitel 5

Ergebnisse

Die „Jura Impressa S9“ besitzt einen RAM für Status, Messwerte und Zwischenberechnungen im Betrieb und einen EEPROM für Einstellungen und Zählerstände, die auch nach einer Stromunterbrechung erhalten bleiben. Durch die Anwendung des Vorgehens in Abschnitt 4.1 können Aussagen zu 40 von 255 Wörtern im EEPROM und 43 von 255 Bytes im RAM getroffen werden. Für den RAM bedeutet dies 19 Statusinformationen und die Position mehrerer Zähler.

5.1 EEPROM

In Abschnitt 3.1.4 wurde der vorhandene Electrically Erasable Programmable Read-Only Memory (EEPROM) Speicher mit seinen 512 Bytes aufgeführt. Im Folgenden wird vom „Wort“, bzw. dessen ersten (XX) und zweiten (YY) Byte gesprochen.
Dabei gilt: „Wort“ $=$ 256$\times$„erstes Byte“ $+$ „zweites Byte“ oder auch eine Antwort des Kaffeevollautomaten in der Form `re:YYXX`.

Abbildung A.1(a) zeigt noch einmal das Speicherschema des EEPROM.

5.1.1 Kaffeezubereitung

Der Kaffeevollautomat unterscheidet bei der Zubereitung zwischen einem einmaligen Pulverkaffee und der normalen Zubereitung über das Bohnenfach. Nur im Falle eines *Pulverkaffees* wird der Zähler in Wort 06_{16} um einen inkrementiert. *1 großer Kaffee*, *2 große Kaffees*, *1 kleiner Kaffee*, *2 kleine Kaffees* und ein *Spezialkaffee* werden bei einer Zubereitung über das Bohnenfach einzeln in den Zählern der Wörter 00_{16} bis 04_{16} in eben dieser Reihenfolge erfasst. Der in den Einstellungen des Kaffeevollautomaten angezeigte *Bezüge-Zähler* gibt die Summe der bis hier genannten 6 Zähler aus.

Die Zähler in den Worten $0F_{16}$ und 15_{16} werden bei *jeder Zubereitung* jeweils um einen inkrementiert, egal ob ein einfacher oder ein doppelter Bezug erfolgt, ebenso egal ob als

Pulverkaffee oder über das Bohnenfach. Erreicht der Wert in $0F_{16}$ eine Anzahl von 220 Bezügen erscheint eine Reinigungsankündigung.

Darüber hinaus gibt es zwei *partielle Zähler* in den Worten $0D_{16}$ und 10_{16}, die ungefähr bei jeder dritten Zubereitung oder Spülung um einen inkrementiert werden. Der genaue Grund dafür ist bisher unbekannt.

Der *Trester*-Stand in der Schale dieses Kaffeevollautomaten wird nicht gemessen, sondern bei jeder Zubereitung in *zwei Zählern* vermerkt. In Wort $0E_{16}$ ist nicht genau bekannt, wann um einen inkrementiert wird. Ab einem Wert von 16 folgt dann aber die Meldung „Trester leeren“.
Ein zweiter Trester-Zähler befindet sich hinter Wort 34_{16}. Je nach eingestellter Pulvermenge wird der Zähler um mehrere Einheiten erhöht. Ab einem Wert von 960 folgt ebenfalls die Meldung „Trester leeren“.
Beide Werte werden bei der Entnahme der Schale für mindestens 8 Sekunden automatisch auf 0 zurück gesetzt. Dies geschieht auch bei einer Reinigung.

5.1.2 Kaffee Einstellungen

Die *Pulvermenge* kann in 29 Stufen auf einer Skala von 0 bis 28, für einen kleinen Kaffee in Wort 82_{16}, für einen großen Kaffee in Wort 83_{16} und für einen Spezialkaffee in Wort 84_{16}, jeweils im zweiten Byte eingestellt werden. Der Standardwert, über die `[N]`-Taste am Kaffeevollautomaten, beträgt 5 für einen kleinen Kaffee, 8 für einen großen Kaffee und 11 für einen Spezialkaffee. Die Menge wird automatisch für zwei kleine Kaffees und zwei große Kaffees hochgerechnet.

Die *Wassermenge* eines kleinen Kaffees wird in Wort 86_{16}, eines großen Kaffees in Wort 87_{16} und eines Spezialkaffees in Wort 88_{16} über beide Bytes hinweg gespeichert. Der Standardwert, über die `[N]`-Taste am Kaffeevollautomaten, beträgt 180 für einen kleinen Kaffee, 340 für einen großen Kaffee und 380 für einen Spezialkaffee. Auch hier wird die Menge automatisch auf zwei kleine Kaffees und zwei große Kaffees hochgerechnet.

Die *Temperatur* kann für einen kleinen Kaffee, für einen großen Kaffee und für einen Spezialkaffee jeweils normal oder hoch eingestellt werden. Der Standardwert liegt bei 0 im zweiten Byte von Wort 85_{16} und bedeutet eine normale Temperatur für alle Sorten. Möchte man einen kleinen Kaffee auf heiß stellen, addiert man 1. Für einen großen Kaffee addiert man 2 und für einen Spezialkaffee addiert man 4 auf den alten Wert. Der Wert 7 ist somit das Maximum und bedeutet eine heiße Temperatur für alle Sorten.

5.1.3 (Tee-) Wasser- und Dampfportionen

In Wort BB_{16} wird die Größe einer *Dampfportion* in Sekunden abgespeichert. Nach der eingestellten Zeit endet automatisch die Dampfausgabe am Kaffeevollautomaten beim Bezug

einer Dampfportion.

In Wort BC_{16} wird die Größe einer *Teeportion* abgespeichert. Die Einheit ist hier nicht bekannt, es könnte sich aber um die Größenordnung von 0.1 Sekunden handeln. Wird das Drehrad von der Position „Kaffeetasse“ auf „Teeauslass“ gestellt, wird eine Teeportion an heißem Wasser ausgegeben und automatisch beendet.

5.1.4 Weitere Maschinen Einstellungen

Wort 24_{16} speichert im ersten Byte die Information der *Wasserhärte*. Der Wert 0 deaktiviert die Wasserhärte-Funktion. Die Werte 1 bis 4 repräsentieren die vier Stufen, die im Benutzerhandbuch zur Maschine ausgeführt werden. In dem zweiten Byte wird der *Economy Mode* gesteuert. Aktiviert versetzt dieser den Kaffeevollautomaten in einen Sparmodus, in dem die Dampfbereitschaft und die Erwärmung der Tassenablage eingestellt wird. 1 aktiviert und 0 deaktiviert den Economy Mode.

In Wort 25_{16} wird die *automatische Einschaltzeit* des Kaffeevollautomaten bei eingestellter Uhrzeit gespeichert. Das erste Byte speichert die *Stunde* in einem Wertebereich von 0 bis 23 und das zweite Byte die *Minute* in einem Wertebereich von 0 bis 59. Der deaktivierte Zustand ist gegeben, wenn die Stunde auf 0 und die Minute auf 128 steht.

In Wort 26_{16} steht im zweiten Byte die *automatische Ausschaltzeit*. Möglich sind die Werte 0, 1, 2, 4, 6, 8, 10, 12, 14, 16 und 18. Wert 0 bedeutet deaktiviert, die übrigen Werte geben ein Intervall in halben Stunden an, nach dem sich der Kaffeevollautomat automatisch abschaltet. Das bedeutet, dass 0,5 Stunden das Minimum und 9 Stunden das Maximum ist.

In Wort 31_{16} befindet sich nach [Kö18] der *Maschinen Typ*. Dieser kann mittels `RE:31` gelesen und mittels `WE:31,xxxx` überschrieben werden.

In Wort $7D_{16}$ befindet sich ein Zähler, der bei jedem Einschaltvorgang inkrementiert wird und somit die totale *Anzahl an Einschaltvorgängen* erfasst. Ausschaltvorgänge werden nicht vermerkt.

In Wort 81_{16} befindet sich im zweiten Byte die *Spracheinstellung*. Die folgenden Werte repräsentieren folgende Sprachen: 0 Deutsch, 16 Italienisch, 32 Niederländisch, 48 Spanisch, 64 Englisch, 80 Französisch und 96 Portugiesisch.

In Wort $B0_{16}$ wird im zweiten Byte mit einem Wert von 16 der *Inkasso Modus* aktiviert und mit einem Wert von 0 deaktiviert. Bei eingeschaltetem Inkasso Modus erfolgt kein Bezug mehr über die Bezugstasten an der Vorderseite des Gehäuses, jedoch wird die gedrückte Taste über die UART Schnittstelle weitergeleitet. Mit dem entsprechenden Befehl kann über die serielle Schnittstelle dann die Zubereitung angestoßen werden.
Man erhält von dem Kaffeevollautomaten auf Knopfdruck `?PAE` für einen kleinen Kaffee, `?PAF` für zwei kleine Kaffees, `?PAA` für einen großen Kaffee, `?PAB` für zwei große Kaffees,

`?PAG` für einen Spezialkaffee, `?PAJ` für eine Dampfportion, `?PAI` für Wasserdampf und `?PAK` für heißes Teewasser.

5.1.5 Filter

In Wort $7D_{16}$ wird im zweiten Byte gespeichert, ob ein *Filter eingelegt* ist. Der Wert 0 steht für „nein“, 8 bedeutet „ja“. Dieser Wert hat Einfluss auf Zähler an anderen Speicherstellen, die zum Teil nur im Falle eines „ja“ weiter gezählt werden.

In Wort 05_{16} befindet sich ein *Filter-Zähler*, der für jeden neu eingesetzten Filter inkrementiert wird.

Bei einem Filterwechsel werden die Werte in den Worten $B1_{16}$, $B3_{16}$ und $C1_{16}$ (Bedeutung unbekannt) auf 0 zurück gesetzt. Erreicht der Wert in Wort $B3_{16}$ einen Stand von 500 sind 50 Liter Wasserdurchfluss erreicht und der Filter muss gewechselt werden. Die Einheit lautet entsprechend 0.1 Liter. Laut Handbuch sollte ebenso spätestens nach zwei Monaten der Filter gewechselt werden.

5.1.6 Spülung

In Wort 07_{16} befindet sich ein Gesamtzähler für die *Anzahl an Spülungen*.

Die partiellen Zähler hinter den Worten $0D_{16}$ und 10_{16} werden auch bei einer Spülung angestoßen. Der Wert in den Worten 11_{16}, bei eingelegtem Filter auch $B1_{16}$ und $B3_{16}$, werden bei einer Spülung um 1 inkrementiert. Erreicht der Wert in Wort $B3_{16}$ den Wert 500, sind 50 Liter Wasserdurchfluss erreicht und der Filter muss gewechselt werden.

5.1.7 Reinigung

Die Meldung wird ausgelöst, wenn der Wert in Wort $0F_{16}$ einen Wert von 220 erreicht hat oder der Wert in Wort 11_{16} einen Wert von 180 erreicht hat. Im ersten Fall sind 220 Bezüge erfolgt, im zweiten Fall waren es 180 Spülungen.

In den Worten 07_{16} (Anzahl an Spülungen), 08_{16} (Bedeutung unbekannt), $7C_{16}$ (Bedeutung unbekannt) und am Ende nach der abschließenden Spülung bei eingelegtem Filter in $B1_{16}$ und $B3_{16}$ wird der Wert um 1 inkrementiert. In den Worten 10_{16} und $B3_{16}$ wird der Wert um 10 inkrementiert. Die Werte in den Worten $0E_{16}$, $0F_{16}$, 11_{16} und 34_{16} werden zurück auf 0 gesetzt. Wort 11_{16} wird jedoch anschließend an die abschließende Spülung gleich wieder um 1 inkrementiert.

Wort 29_{16} Byte 1 wird zu Beginn einer Reinigung auf 65 gesetzt und am Ende auf 0 zurückgesetzt.

5.2 RAM

In Abschnitt 3.1.4 wurde der vorhandene Ramdom-Access Memory (RAM) Speicher mit seinen 256 Bytes aufgeführt. Jedes Byte besitzt 8 Bits, die im Folgenden mit den Nummern 0 ($= 1 \times 2^0$) bis 7 ($= 1 \times 2^7$) bezeichnet werden. Abbildung A.1(b) zeigt noch einmal das Speicherschema des RAM.

Aufgrund starker, aber zum Teil regelmäßiger Schwankungen, wurden die folgenden Bytes über mehrere Speicherauszüge ohne zwischenzeitliche Veränderungen ermittelt und *für diese Arbeit ausgeblendet*: 02_{16}, 09_{16}, $0C_{16}$, $1C_{16}$ bis 21_{16}, 27_{16}, 28_{16}, $2A_{16}$, $2C_{16}$, $2E_{16}$, 41_{16}, 66_{16}, 67_{16}, $6A_{16}$, $6C_{16}$, 70_{16}, $7C_{16}$, 94_{16}, 95_{16}, 97_{16}, $9C_{16}$, $9D_{16}$, $A2_{16}$, $F1_{16}$ und FC_{16} bis FF_{16}.

Die Aufschlüsselung und der Bezugspunkt für die im Folgenden genannten Rubriken sind die Spalten der Tabelle A.2 und der Tabelle A.3.

Unterstrichene und hervorgehobene Zellen sind „immer“ abfragbar. Die einzige Voraussetzung ist die in Abschnitt 3.1.1 beschriebene Verkabelung sowie eine angelegte Netzspannung an den Kaffeevollautomaten. Der Kaffeevollautomat **muss** sich dann **nicht** im eingeschalteten Zustand befinden.

Eingeklammerte Zellen sind nicht exklusiv für die genannte Rubrik, sie teilen sich den Wert mit weiteren Funktionen.

5.2.1 Maschine an

Sobald der Kaffeevollautomat eingeschaltet wird, ändern sich die Werte mehrerer Speicherstellen im RAM. Tabelle A.2 zeigt die regelmäßigen Änderungen beim Einschalten in der ersten Spalte „Maschine an“. Der Umschwung der 7 Bits erfolgt sofort nach der Betätigung des „Ein“-Schalters, selbst wenn sich der Kaffeevollautomat noch aufwärmt und „Bitte warten“ anzeigt. Eine Änderung nach der Aufwärmphase konnte nicht festgestellt werden.

Beim Ausschalten geschehen die Änderungen umgekehrt.

5.2.2 Schale fehlt

Die beiden linken Kontakte, die von der eingesetzten Schale verbunden werden, siehe Abbildung 4.2(a), sind voneinander getrennt. Ohne eine eingesetzte Schale schaltet der Kaffeevollautomat sicherheitshalber alle Tätigkeiten ab. Dies wird auch im ausgeschalteten Zustand vermerkt.

Die Änderungen sind in Tabelle A.2 in der zweiten Spalte dargestellt. In Byte 16_{16} an Bit 0 und in Byte 69_{16} an Bit 0 steht nur eine 0, wenn der Kaffeevollautomat eingeschaltet ist. Die anderen beiden Bits enthalten zu jeder Zeit den aktuellen Status.

5.2.3 Schale leeren

Die linken beiden und der rechte Kontakt sind kurzgeschlossen. Der Wasserstand in der Auffangschale ist dann gefährlich hoch und der Kaffeevollautomat schaltet sicherheitshalber alle Tätigkeiten ab. Dies wird auch im ausgeschalteten Zustand vermerkt.

Die Änderungen sind in Tabelle A.2 in der dritten Spalte dargestellt.

5.2.4 Trester leeren

Hat einer der beiden Trester-Zähler seine Grenze erreicht, erfolgt die Aufforderung, den Trester zu leeren. Indem die Schale für mindestens 8 Sekunden entnommen wird, werden beide Zähler auf 0 zurück gesetzt.

Eine wichtige Ergänzung zur Tabelle A.2 ist, dass nur der Trester-Zähler im EEPROM in Wort 34_{16} die Änderung im RAM in Byte 80_{16} an den Bits 1 und 0 auslöst. Um allgemein diese Meldung ausschließen zu können, muss sich der Kaffeevollautomat im eingeschalteten Zustand befinden.

5.2.5 Wasser füllen

Bei dieser Meldung schließt der magnetische Schwimmer im Wassertank nicht den Kontakt und die Meldung leuchtet auf. Der Wasserspiegel kann hierbei zu niedrig sein oder der Wassertank wurde zum Auffüllen oder zum Wechseln des Wassers entnommen. Dies wird auch im ausgeschalteten Zustand vermerkt.

Die Änderungen sind in Tabelle A.2 in der fünften Spalte dargestellt.

5.2.6 Gerät reinigen

Sind 220 Bezüge oder 180 Spülungen erreicht, wird eine Reinigung gefordert. In Byte 10_{16} an Bit 7 und in Byte 22_{16} an Bit 4 steht dann je eine 1, siehe Tabelle A.2 Spalte 6.

5.2.7 Maschine spült

Wenn der Kaffeevollautomat eine Spülung vornimmt, wird dies in mehreren Bits vermerkt. Die Änderungen sind in Tabelle A.2 in der siebten Spalte dargestellt.

5.2.8 Maschine vor dem Ausschalten spülen

Der Kaffeevollautomat erzwingt eine Spülung vor dem Ausschalten, wenn während der Betriebszeit ein Kaffeebezug erfolgte, ohne eine abschließende Spülung vorgenommen zu haben. In Byte $0D_{16}$ an den Bits 3 und 2 steht in diesem Fall mal an dem einen und mal an

dem anderen Bit eine 1. Dieser Merker befindet sich sehr wahrscheinlich hier, auch wenn die genaue Bedeutung der einzelnen Bits nicht sicher bekannt ist.

5.2.9 Tassenbeleuchtung

Bei einer Zubereitung, oder laut Benutzerhandbuch auch durch das Drücken einer beliebigen Taste im ausgeschalteten Zustand, wird die Tassenbeleuchtung eingeschaltet. Nach 3 Minuten schaltet die Tassenbeleuchtung dann wieder automatisch ab.

In Byte $0F_{16}$ an den Bits 2 bis 1 steht bei eingeschalteter Tassenbeleuchtung immer mindestens eine 1. Beim Betreten des Menüs oder auch beim Ausschalten wird mindestens eine 1 gesetzt. Eine genaue Aufschlüsselung oder Zuordnung der beiden Bits zu bestimmen Rahmenbedingungen konnte nicht hergestellt werden.

5.2.10 Filter wechseln

Ist ein Durchfluss von 50 Litern erfolgt, erscheint diese Meldung mit der Aufforderung den Wasserfilter auszuwechseln. Die Änderungen sind in Tabelle A.3 in der ersten Spalte dargestellt. Diese Meldung kann im ausgeschalteten Zustand des Kaffeevollautomaten abgefragt werden.

5.2.11 Hahn offen & Teeportion

Wird der Drehknauf von der „Kaffeetasse“ weggedreht, wird der *Hahn geöffnet*. Erreicht der Regler darüber hinaus die Stellung des „hinplätschernden Wasserhahns“, ist eine *Teeportion* aktiv.

Die Änderungen sind in Tabelle A.3 in der zweiten und dritten Spalte dargestellt. Auch diese Meldungen können im ausgeschalteten Zustand des Kaffeevollautomaten abgefragt werden.

5.2.12 Dampfbezug & Wasserdampfportion

Beim *Dampfbezug* wird bis zur erneuten Betätigung der Taste heißer Wasserdampf aus dem rechten Rohr ausgegeben. Eine *Wasserdampfportion* ist zeitlich begrenzt und kann in den Einstellungen dimensioniert werden.

Die Änderungen sind in Tabelle A.3 in der vierten und fünften Spalte dargestellt. Die einzige Position für eine sichere Unterscheidung ist in Byte 13_{16} an Bit 1, wo nur bei dem Dampfbezug eine 1 steht.

5.2.13 Pulver füllen

Hinter der Wartungsklappe befindet sich die Wahltaste für vorgemahlenes Kaffeepulver. Wird diese betätigt, wird der nächste Kaffeebezug einmalig über das Pulverfach zubereitet und

es werden keine Bohnen gemahlen. Dieser Modus kann ebenfalls über die serielle UART Schnittstelle mit dem Befehl `FA:03` ausgelöst werden.

In Byte 04_{16} an Bit 0 befindet sich dann eine 1, die nach dem nächsten Bezug automatisch wieder auf 0 zurückgesetzt wird.

5.2.14 Bohnen füllen

In Byte $0E_{16}$ an Bit 7 steht eine 1 im Falle fehlender oder verklemmter Bohnen. Die Meldung verschwindet mit ausreichend Bohnen im Bohnenfach beim nächsten Bezug. Zusätzlich darf dabei kein Stein das Mahlwerk blockieren.

5.2.15 Zubereitung

Für eine Kaffeezubereitung werden im *ersten Schritt* die Bohnen gemahlen. In Byte 03_{16} an den Bits 7 und 4 steht dann je eine 1. In Byte 03_{16} an Bit 7 steht nur während einer Kaffeezubereitung eine 1 und nicht im Falle des manuellen Starts des Mahlwerks mit dem Befehl `FN:07` oder `FN:09`.

Im *zweiten Schritt* wird der Kaffee aufgebrüht und ausgeschenkt. Die Änderungen sind in Tabelle A.3 in der letzten Spalte dargestellt.

Über die *gesamte Zeit* steht in Byte $0A_{16}$ an Bit 6 eine 1. Darüber hinaus steht in Byte $0A_{16}$ an den Bits 2 bis 0 die gewählte Portion kodiert. 1 steht für einen kleinen Kaffee, 2 steht für 2 kleine Kaffees, 3 steht für einen großen Kaffee, 4 steht für 2 große Kaffees und 5 steht für einen Spezialkaffee.

5.2.16 Geteilte Bits

In Tabelle A.2 und Tabelle A.3 sind mehrere Einträge eingeklammert. Im Folgenden sollen einige Zusammenhänge gegenübergestellt werden:

In Byte 03_{16} an den Bits 6 und 1 steht der Wert auf 1, wenn der Kaffeevollautomat eine Spülung vornimmt, eine Teeportion ausschenkt oder manchmal, wenn eine Zubereitung eines Kaffees erfolgt. In Byte 03_{16} an Bit 6 war gerade die Zubereitung unregelmäßig, während sich Byte 03_{16} an Bit 1 reproduzierbar verhielt. Auch in Byte 62_{16} an Bit 1 steht eine 1 und in Byte 68_{16} an Bit 5 steht eine 0 wenn die Maschine spült oder sich im zweiten Schritt der Zubereitung befindet. In Byte 68_{16} an Bit 6 steht eine 1, wenn eine Teeportion ausgegeben wird oder die Maschine spült.

In Byte 03_{16} an Bit 3 steht eine 1 bei einer Wasserdampfportion oder dem Einschalten direkt nach dem Ausschalten mit abschließender Spülung.

In Byte 04_{16} an Bit 3 steht eine 1, wenn die Schale fehlt, der Wassertank fehlt oder die Schale voll ist, aber auch beim offenen Hahn oder dem Dampfbezug.

In Byte 04_{16} an Bit 3 steht eine 0 bei einer Teeportion, beim Dampfbezug, bei einer Wasserdampfportion oder auch bei der Zubereitung eines Kaffees. In Byte $4C_{16}$ an Bit 0 steht eine 0 beim Wasser füllen oder offenem Hahn. Auch in Byte 91_{16} an Bit 0 steht eine 0 beim Wasser füllen aber wechselhaft bei der Zubereitung eines Kaffees.

In Byte 16_{16} an Bit 0 und Byte 69_{16} an Bit 0 steht eine 0, wenn die Schale fehlt oder die Maschine ausgeschaltet ist, andernfalls befindet sich der Kaffeevollautomat hier in einem betriebsbereiten Zustand.

5.2.17 Deutung und Zähler

Im RAM könnte Byte 13_{16} an den Bits 2 bis 0 für eine interne Wasserschaltung stehen.

In den Bytes 37_{16} bis $3C_{16}$, in Byte 43_{16} und in den Bytes 71_{16} bis 75_{16} befinden sich sehr wahrscheinlich interne Zählerstände, die während einer Kaffeezubereitung oder allgemeiner beim Pumpen von Wasser inkrementiert werden. In dem Byte $8B_{16}$ sowie bis in das davor liegende Byte $8A_{16}$ reicht eine Zähler für primär den Dampfbezug oder eine Wasserdampfportion, aber auch bei einer Kaffeezubereitung oder Spülung bewegt sich der Zähler.

5.3 Alleinstellungsmerkmal für die API und die Webseite

Tabelle A.2 und Tabelle A.3 machen deutlich, dass es mehrfach redundante Speicherstellen im RAM gibt, an denen potentiell die gleiche Information steht. Da die serielle Übertragung abgefragter Speicherstellen Zeit benötigt, versucht die API nur die wirklich nötigen Speicherstellen abzufragen und dadurch Zeit einzusparen.

Abbildung A.2 zeigt die abgefragten Speicherstellen für die API im EEPROM. Es werden vier Zeilen und drei einzelne Worte abgefragt. Da auch die Anfrage in der Übertragung Zeit benötigt, ist eine Zeile effizienter als 16 einzelne Wörter.

Abbildung A.3 zeigt die abgefragten Speicherstellen für die API im RAM. Hier können nur ganze Zeilen abgefragt werden, aber diese können an beliebiger Stelle beginnen. Es reichen für alle Funktionen vier Zeilen aus. Viele interne Zähler und Dopplungen können ausgelassen werden. Dennoch wird deutlich, dass die Bytes bestimmter Statusinformationen, die an verschiedenen Bits stehen, sehr nahe beieinander liegen und an anderer Stelle einzelner Bytes effizienter als ganze Reihen wären.

5.4 Display

Das Display der „Jura Impressa S9“ umfasst zwei Zeilen mit je 8 Zeichen, die wiederum mit je 5×5 Lichtpunkten dargestellt werden. Tabelle A.4 zeigt die verfügbaren Schriftzeichen auf. Es

handelt sich um arabische Zahlen, große lateinische Buchstaben sowie ein paar Sonderzeichen, die unter anderem eine Skala darstellen können.

0_{10} bis 31_{10} und alle ASCII-Zeichen über 97_{10}, auch aus der erweiterten ASCII-Tabelle von 128_{10} bis 255_{10} und darüber, stellt der Kaffeevollautomat nicht mehr als eindeutig bekannte ASCII-Zeichen dar. Es handelt sich höchstens um verpixeltes Kanji und wurde in der Tabelle ausgelassen.

Kapitel 6

Diskussion

Dieses Kapitel benennt aufgetretene Probleme während der Arbeit und hinterfragt die Ergebnisse aus dem vorherigen Kapitel. Auch der Begriff des *Reverse Engineering* wird hier wieder aufgegriffen.

6.1 Probleme

Nicht nur die Kommunikation mit dem Kaffeevollautomaten, sondern auch das Verständnis der Vorgänge in der Maschine erfordern Zeit und Arbeit.

6.1.1 Kommunikation mit dem Kaffeevollautomaten

Der direkte Zugang ist wahrscheinlich am schnellsten, im laufenden Betrieb aber schlecht möglich, da die Speicheradressen aktiv abgefragt werden müssten und damit Kurzschlüsse in der Elektronik produziert werden würden. Die serielle UART Schnittstelle ermöglicht das Abfragen im laufenden Betrieb, also auch im RAM. Das Auslesen benötigt jedoch seine Zeit, ungefähr 9 Sekunden für den gesamten RAM und ungefähr 15 Sekunden für den gesamten EEPROM. Gerade im RAM gibt es viele Veränderungen im Ruhezustand, sodass auf diesem Weg nie ein zusammenhängender Speicherauszug zu einem festen Zeitpunkt ausgelesen werden kann.

Ein weiterer Nachteil ist, dass während einer Übertragung das interne Bus System gehemmt ist. Eingaben an der Maschine, wie die Navigation durch das Menü, werden während der Erstellung eines Speicherauszugs verzögert umgesetzt. Aber auch in der Gegenrichtung sorgen wechselnde Displaytexte für Verzögerungen während der seriellen Übertragung.

Durch die Optimierung für die API benötigt das Auslesen nur noch ungefähr 3 Sekunden für den RAM und ungefähr 5 Sekunden für den EEPROM. Abbildung A.2 und Abbildung A.3 zeigen die dabei abgefragten Speicherstellen.

6.1.2 Serielle Kommunikation

Gerätedatei direkt ansprechen Um mit dem Arduino und letztlich dem Kaffeevollautomaten zu kommunizieren, war der erste Versuch den Dateideskriptor `/dev/ttyACM0` sowohl lesend als auch schreibend zu nutzen. Zunächst funktionierte dies sehr gut, bis nach wenigen Tagen kaum reproduzierbares Fehlverhalten auftrat. Gerade Antworten der Kaffeemaschine kamen nur noch in Bruchstücken an. Dies wurde bei der Einrichtung bereits im Projekt „CoffeeMachine“[Kö18] in der `Readme.md` für die Ausführung auf einem Raspberry Pi beschrieben.

Versuche, dies direkt zu lösen, indem eine funktionierende Umgebung nachgebaut wurde, schlugen fehl. Dabei konnte über `screen /dev/ttyACM0 9600` und `exit` die serielle Verbindung initiiert werden. Die Einstellungen können dann über `stty -F /dev/ttyACM0` eingesehen werden, jedoch hatte es nicht denselben Effekt, nur diese Einstellungen gezielt über den Befehl

```
stty -F /dev/ttyACM0 9600 raw \
ignbrk -brkint -icrnl -imaxbel \
-opost -onlcr \
-isig -icanon -iexten -echo -echoe -echok -echoctl -echoke
```

Listing 6.1: stty zum setzen der Verbindungseinstellungen

zu setzen.

C++ und die *libserial*-Library Für eine sichere Verbindung wird in dem C++ Programm auf die im Linux Repository erhältliche Library *libserial* zurückgegriffen. Beim Verbindungsaufbau über die *connect()* Methode der *SerialConnection* Klasse wird die Schnittstelle initialisiert. Hier werden der Gerätepfad, die Baudrate und eine erwartete Mindestlänge festgelegt.

Kommandos gehen zuverlässig an den Arduino raus und ausgelesene Antworten kommen nun im Ganzen an. Der *read* Befehl ist ein blockierender Aufruf, der das Programm anhält bis eine Antwort vorliegt. Dies stellt für die Speicherauslesung kein Problem dar, da die ordnungsgemäßen Befehle immer ein `ok:` oder `xx:0-F` Speicherauszug zurückgeben.

6.1.3 Unbekannte Speicherorte

Bis auf den Speicherort des veränderbaren Standardtextes konnten alle Einstellungen und Grenzwerte sowie ein paar Funktionsabläufe festen Speicherstellen zugeordnet werden. Da der Standardtext eine Stromunterbrechung nicht übersteht, liegt er sehr wahrscheinlich im RAM oder einem weiteren flüchtigen Speicher der Maschine. Selbst unter der Annahme, dass die Maschine vielleicht nur die halbe ASCII-Tabelle pro Zeichen benutzt und dadurch zwei

Displayzeichen pro Byte kodiert, konnten einfache und monotone Texte wie `AAAAAAAA` nicht in einem Speicherauszug wiedergefunden werden.

Um dies weiter zu ergründen, muss künftig ein Blick in die Maschine erfolgen, um die vorhandenen Speicher und deren Größen zu bestimmen. Nach dem Datenblatt des Mikrocontrollers [Hol17] gibt es vier Ausführungen, von der aber keine den 256 Byte RAM und 512 Byte (256 Wörter) EEPROM, die über die seriellen Befehle adressierbar sind, entspricht.

Es wäre vorstellbar, dass die gegebenen Befehle aus Tabelle A.1 nicht den insgesamt zur Verfügung stehenden Speicher adressieren und nach außen kommunizieren. Andererseits ist es ebenfalls möglich, dass viele unbekannte Speicherstellen gar keine Funktion besitzen; der vorhandene Speicher dieses Standard Mikrocontrollers würde von dem Programm dann nicht voll ausgenutzt werden.

6.1.4 Weitere Automatisierung durch das C++ Programm

Für den EEPROM ist es ein vertretbarer Aufwand, Einstellungen am Gerät vorzunehmen, das Skript anzustoßen und einen Kommentar zu hinterlegen. Die Ergebnisse können aus der JSON Datei abgelesen werden.

Die wechselhaften Sprünge im RAM erforderten ein wiederholtes Anwenden des Skripts, wie in Abschnitt 4.1 beschrieben. Hier könnte das C++ Programm dahingehend ausgebaut werden, $n = 3$ Durchläufe mit dem selben Kommentar zu versehen. Die Auswertung sollte aber von Hand erfolgen und es sollten Ergebnisse aus variierenden Umgebungsbedingungen einbezogen werden. Zum Beispiel den Wasserfilter einmal aktivieren und einmal deaktivieren. Es bedürfte einer sehr guten künstlichen Intelligenz, diese Umgebungen vorzuschlagen und die Ergebnisse zusammenzuführen.

Mit dem in Abschnitt 4.1 beschrieben ersten Vorgehen lassen sich Statusinformationen und Einstellmöglichkeiten sehr gut auslesen. Bedingte Abhängigkeiten, wie die Zusammensetzung eines Bezüge-Zählers aus mehreren Zählern, oder Abläufe, wie die zwei Schritte einer Kaffeezubereitung, wären durch ein starr hochgezogenes Computerprogramm unentdeckt geblieben.

Ausbaufähig ist die wieder bessere Trennung des EEPROMs und des RAMs. In den gefundenen Unterschieden des Skripts können weniger relevante Spalten ausgeblendet werden und es könnten die JSON Dateien mit einem Merker versehen werden, um der nachträglichen *Aufschlüsselung der Veränderungen* im C++ Programm eine bessere Differenzierung zu ermöglichen. Zur Zeit muss für den RAM von Hand die Zeile 192 der `JuraCoffeeMemory.cpp` einkommentiert werden.

6.2 Aussagekraft der Ergebnisse

Die EEPROM Einstellmöglichkeiten können recht sicher festen Speicherstellen zugeordnet werden. Bei einigen partiellen Zubereitungszählern ist aber nicht bekannt, wann sie zählen und wofür. Die Firmware fehlt an dieser Stelle.

Im RAM sind mehrere Status Bits einigermaßen aussagekräftig, vor allem aber die im ausgeschalteten Betriebszustand, die auch in Tabelle A.2 und Tabelle A.3 hervorgehoben sind. Durch viele unregelmäßige Veränderungen wurden aber mehrere Bytes außer Acht gelassen, die an bestimmten Bit Positionen wichtige Merker enthalten könnten.

Unsicherheit besteht durch mehrere Speicherpositionen für ein und dieselbe Funktion. Entweder verbirgt sich hinter einzelnen Positionen noch eine andere Bedeutung oder eine bisher unbekannte Abhängigkeit für weitere Funktionen. Zustände könnten im RAM tatsächlich mehrfach erfasst werden.

6.3 Einordnung zur Terminologie des Reverse Engineerings

Diese Arbeit besteht im Wesentlichen aus den zwei Teilen, den Speicher zu analysieren und zu verstehen, bzw. das Wissen darüber in einer neuen Oberfläche einfach zur Verfügung zu stellen. Ein kurzer Rückblick in Abschnitt 2.1 erinnert an die Begriffe *Forward Engineering*, *Reverse Engineering*, *Redocumentation*, *Design Recovery*, *Restructuring* und *Reengineering*.

Für den ersten Teil wurden eine gegebene serielle Schnittstelle und bereits implementierte Befehle ausgenutzt, um den Speicher auszulesen. Viele Aufnahmen, mit kleinen Veränderungen zwischendurch, ließen Rückschlüsse auf die Bedeutung der Werte an bestimmten Speicherstellen zu. Gerade mit den Überlegungen, was während einer Aktion am Kaffeevollautomaten alles im Speicher passiert, lässt sich diese Tätigkeit dem Begriff *Reverse Engineering* zuordnen. Aus dem fertig implementierten und gegebenen System werden Rückschlüsse auf das Design der Maschine gezogen. *Redocumentation* trifft aufgrund fehlender Spezifikationen und Quellcodeauszügen wenig zu. Betrachtet man aber das Benutzerhandbuch und Interneteinträge über Erfahrungen mit der Maschine als externe Informationen, sowie Schlussfolgerungen und Unschärfelogik über den RAM, kann dieser Part auch als *Design Recovery* über den Speicher angesehen werden.

Mit diesem Wissen des ersten Teils wurde dann im zweiten Teil etwas Neues geschaffen. Das Abfragen der aktuellen Einstellungen und Zustände bietet eine neue Oberfläche, die als *Redocumentation* oder *Restructuring* angesehen werden kann. Für die *Redocumentation* spricht das neue Ausgabeformat auf der Ebene der Implementierung. Ein eigenes Gerät ist zur Alternative (Ausnahmen siehe 7.1) für das kleine Display des Geräts geworden und gibt den aktuellen Zustand in ausführlicherer Weise aus. Ohne eine präventive Instandhaltung beabsichtigt zu haben ist *Restructuring* aber auch zutreffend, da Wissen aus dem ersten Teil

hier eingeflossen ist. Wenn die Bedeutung bestimmter Speicherstellen nicht bekannt gewesen wäre, hätten die eingestellten Werte keine Bedeutung gehabt, was für den Begriff *Restructuring* spricht.

Für eine individuelle Zubereitung wird nur die Datengrundlage kurz zuvor mit einem Impuls von außen verändert, der Kaffee wird dabei nach wie vor von den Standard Abläufen der Kaffeemaschine zubereitet. Die neue Oberfläche kann als *Restructuring* oder dem allgemeineren *Reengineering* angesehen werden. Da die neue Oberfläche über die serielle Schnittstelle in Verbindung mit einem eigenen Endgerät aber sehr von den Bordmitteln der Maschine abweicht, passt vielleicht das *Reengineering* etwas besser. Die Funktionalitäten der Maschine sind aber immer noch die Gleichen, das Programmmenü muss nun jedoch nicht mehr verwendet werden, um die allgemeinen Geräteeinstellungen für den eigenen Kaffee anzupassen.

Im Detail ist die genaue Zuordnung der Begrifflichkeiten immer eine persönliche Auslegung dieser Begriffe. Sehr zutreffend ist das Zitat aus Abschnitt 2.1, denn diese Arbeit hat den Speicher intensiv betrachtet und mittels der Ergebnisse aus dem zweiten Teil um neue Perspektiven bereichert, ohne das Ziel verfolgt zu haben, die Maschine zu klonen oder umzubauen.

Kapitel 7

Zusammenfassung

Im Rahmen dieser Arbeit wurde der Speicher des Kaffeevollautomaten „Jura Impressa S9“ reverse engineered. Für den Aufbau und die Verkabelung ist auf das Arduino Skript und die Dokumentation aus dem Projekt [Kö18] zurückgegriffen worden. Viele der dort gebündelten Informationen stammen wiederum aus der Community[1]. Darauf aufbauend folgte bereits in Kapitel 3 die Definition der *UART Gruppe* und des Speicheraufbaus mit der Definition des *EEPROM Wortes*.

Zur Untersuchung des Speichers und der darin enthaltenen Informationen zur Überwachung und Steuerung des Kaffeevollautomaten, wurde eine strukturierte Vorgehensweise in Abschnitt 4.1 entwickelt. Mit einem C++ Programm sind Speicherauszüge vor und nach einer Veränderung an der Maschine aufgenommen und verglichen worden. Die Ergebnisse mit den Speicherpositionen bestimmter Einstellungen oder Funktionen listet Kapitel 5 auf.

Neu gebündelt wurden diese über eine leicht zugängliche API nach außen offengelegt. EEPROM und RAM können dadurch schneller abgefragt werden. Die Erstellung eines vollständigen Speicherauszugs bräuchte wesentlich mehr Zeit. Abschließend wird die API auf einer kleinen Webseite visuell präsentiert. Im Abschnitt 6.2 dieser Arbeit sind die Vorgehensweise und die Aussagekraft der Ergebnisse kritisch hinterfragt.

Das Reverse Engineering besitzt mit seinen benachbarten Disziplinen aus Abschnitt 2.1 ein weites Spektrum an Möglichkeiten, alltägliche Gegenstände zu erhalten und weiterzuentwickeln. In dieser Bachelorarbeit wurde deutlich, dass die Erarbeitung an Wissen über den internen Speicher des Kaffeevollautomaten als *Reverse Engineering* oder *Design Recovery* bezeichnet werden kann. Die darauf aufbauende API kann wiederum als *Redocumentation* oder *Restructuring* bezeichnet werden. Die Webseite ist schlussendlich als neue Oberfläche der Ertrag des *Reengineering* und zeigt eine smatere Nutzbarkeit des Gerätes.

[1] http://protocoljura.wiki-site.com/

7.1 Ausblick

Die in dieser Arbeit aufgeführten Ergebnisse sind noch nicht vollständig, zum Beispiel nennt das Benutzerhandbuch weitere Meldungen wie: `Gerät verkalkt`, `Störung 2` oder `Störung 8` sowie etliche Menümeldungen während der internen Programmabläufe. Die Speicherpositionen der Displaymeldungen sind unbekannt, evtl. sogar für alle Sprachen fest im Programmablauf hinterlegt. Die RAM Position für die Wahl der Display Meldung ist ebenfalls unbekannt, siehe Abschnitt 6.1.3. Die verbleibenden drei Meldungen konnten während dieser Arbeit nicht gezielt ausgelöst werden.

Mit entsprechendem zeitlichen und finanziellen Budget könnte die Arbeit weiter fortgeführt werden. Man könnte eine Umgebung schaffen, in der die drei Störungsmeldungen ausgegeben werden, um deren Ursprung im RAM zu lokalisieren. Ebenso wäre es interessant, fehlende Einheiten zu bestimmen. Zeit-, Gewichts- oder Durchflusseinheiten würden dem Anwender die Konfiguration vereinfachen.

Es wäre ebenfalls lohnenswert, die Firmware aus dem Read Only Memory (ROM) der Maschine auszulesen und zu disassemblieren. Dadurch kann die Erkenntnis, welche Speicherstellen überhaupt vorgesehen und angesprochen werden, erlangt werden. Ebenso würden Vorgänge bei Funktionsabläufen verständlicher.

A. Kamilaris, A. Pitsillides und V. Trifa nennen in ihrem Paper [KPT11] als eine Grundanforderung für das intelligente Zuhause den echten Mehrbenutzer Betrieb. Die Aufbereitung der Webseite könnte durch Crossbar.io[2] mithilfe von AutobahnC++[3] erweitert werden. Über das Web Application Messaging Protocol (WAMP) könnte regelmäßig der Zustand des Kaffeevollautomaten an alle verbundenen Endgeräte weitergegeben werden, ohne dass ein Endgerät tätig werden muss und während der Kommunikation die Ressource bindet.

[2] https://crossbar.io/
[3] https://github.com/crossbario/autobahn-cpp

Literaturverzeichnis

[CC90] CHIKOFSKY, E. J. ; CROSS, J. H.: Reverse engineering and design recovery: a taxonomy. In: *IEEE Software* 7 (1990), Jan, Nr. 1, S. 13–17. http://dx.doi.org/10.1109/52.43044. – DOI 10.1109/52.43044. – ISSN 0740–7459 3, 4

[Eil05] EILAM, Eldad: *Reversing : secrets of reverse engineering*. Indianapolis, Ind. : Wiley, 2005 http://www.gbv.de/dms/ilmenau/toc/480966761eilam.PDF. – ISBN 0764574817 9780764574818 5

[Hol17] HOLTEK SEMICONDUCTOR INC. (Hrsg.): *HT46F46E/HT46F47E/HT46F48E/HT46F49E Cost-Effective A/D Flash MCU with EEPROM*. Rev. 1.50. Hsinchu, Taiwan: Holtek Semiconductor Inc., April 2017. http://www.holtek.com.tw/documents/10179/11842/HT46f4xev150.pdf. – http://www.holtek.com/productdetail/-/vg/46f4xe 41

[Ing94] INGLE, Kathryn A.: *Reverse engineering*. New York u.a. : McGraw-Hill, 1994 http://www.loc.gov/catdir/toc/mh022/94018447.html. – ISBN 0070316937 9780070316935 4, 5

[KPT11] KAMILARIS, Andreas ; PITSILLIDES, Andreas ; TRIFA, Vlad: The Smart Home meets the Web of Things. In: *International Journal of Ad Hoc and Ubiquitous Computing* 7 (2011), Nr. 3, 145. http://dx.doi.org/10.1504/ijahuc.2011.040115. – DOI 10.1504/ijahuc.2011.040115 1, 11, 46

[Kö18] KÖSTLER, Maximilian: *CoffeeMachine*. Git Repository. https://collaborating.tuhh.de/e-17/General/CoffeeMachine. Version: Januar 2018. – nicht öffentlich 8, 9, 31, 40, 45, 57

[Mad15] MADAKAM, Somayya: Internet of Things: Smart Things. In: *International Journal of Future Computer and Communication* 4 (2015), Nr. 4, 250–253. http://dx.doi.org/10.7763/ijfcc.2015.v4.395. – DOI 10.7763/ijfcc.2015.v4.395 1

[Off18] OFF, Fabian: *Reverse-engineering a De'Longhi Coffee Maker to precisely bill Coffee Consumption*, Universität Magdeburg, Diplomarbeit, Februar 2018 5

[Sal09] SALEH, Kassem A.: *Software engineering*. Boca Raton, Fla. : Ross, 2009. – ISBN 1932159940 9781932159943. – Literaturangaben 3, 13

[SRT00] SOUCEK, Stefan ; RUSS, Gerhard ; TAMARIT, Clara: *The smart kitchen project-an application of fieldbus technology to domotics*. Citeseer, 2000 5

Anhang A

Große Abbildungen

Im Folgenden sind größere Abbildungen dargestellt, die eine ganze Seite oder eine Doppelseite einnehmen.

Tabelle A.1 führt eine Auswahl der möglichen Befehle für den Kaffeevollautomaten auf. Die vollständige Liste befindet sich unter `JuraCoffeeMemory/result/commands.txt`.

Abbildung A.1 zeigt schematisch den Speicher des Kaffeevollautomaten. Der EEPROM und der RAM sind in jeweils 16 Zeilen zeilenweise dargestellt.

Tabelle A.2 und Tabelle A.3 stellen spaltenweise die Bit-Änderungen für je eine Meldung oder Funktion im RAM dar.

Abbildung A.2 und Abbildung A.3 zeigen die abgefragten Wörter und Bytes für die Ausgabe der API.

Tabelle A.4 stellt den verfügbare Zeichensatz für das Display dar. Gleichzeitig werden die dafür nötigen ASCII Kommando-Eingaben aufgeführt.

Kommando	Beschreibung	Rückgabewert
Betriebszustand (AN:<id>)		
AN:01	Einschalten	ok:
AN:02	Ausschalten	ok:
AN:03	Display Test	ok:
AN:<id>	u.v.m.	ok:
Bezugstaste (FA:<id>)		
FA:02	Gerät spülen	ok:
FA:0C	Spezialkaffee	ok:
FA:<id>	u.v.m.	ok:
Steuerungskomponenten (FN:<id>)		
FN:<id>	Pumpen, Heizung, u.v.m	ok:
Eingabestatus*		
IC:	Eingaben auslesen*	
Spiele Musik (easter egg)*		
PM:	Play music*	
Speicherzugriff		
RE:<address>	Liest 2 Byte EEPROM Speicher	re:[0000-FFFF]
WE:<address>,<value>	Schreibt 2 Byte in EEPROM	ok:
RT:<address>	Liest eine Zeile EEPROM	rt:[0-F 64x]
RR:<address>	Liest eine Zeile RAM	rr:[0-F 32x]
Sperrung		
?M3	Aktiviert den Inkassomodus	?ok
?M1	Deaktiviert den Inkassomodus	?ok
Display		
?D0	Standard Displaytext zurück setzen	?ok
?D1[A-Z 8-11x]	Displaytext Zeile 1	?ok
?D2[A-Z 8-11x]	Displaytext Zeile 2	?ok
Aktion an der Kaffeemaschine		
	1 kleiner Kaffee (Produkt 1)	?PAE
	2 kleine Kaffees (Produkt 2)	?PAF
	1 großer Kaffee (Produkt 3)	?PAA
	2 große Kaffees (Produkt 4)	?PAB
	Spezialkaffe (Produkt 7)	?PAG
	Dampf (Produkt 6)	?PAI
	1 Dampfportion (Produkt 5)	?PAJ
	1 Tasse Tee (Produkt 8)	?PAK
*** Nicht alle Befehle sind in der Jura Impressa S9 implementiert, siehe weitere Geräte der S-Reihe**		

Tabelle A.1: Befehlsübersicht der Jura Kaffeevollautomaten (S-Reihe)

Adresse DEC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HEX	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0 0x00																
16 0x10	RE:00 (2 Byte Wort lesen,0x00-0xFF)															
32 0x20	re:0000 (2 Byte = 4 HEX Antwort)															
48 0x30																
64 0x40	RT:00 (eine Reihe = 32 Byte lesen, 0x00-0xF0) (32 Byte = 64 HEX Antwort)															
80 0x50	rt:00010001000600010001000000004000F000000000000000000001000F0000000E															
96 0x60																
112 0x70	WE:00,0001 (Schreibe an Adresse, den Wert)															
128 0x80	ok:															
144 0x90																
160 0xA0																
176 0xB0																
192 0xC0																
208 0xD0	1 Kästchen ≡ 1 Byte															
224 0xE0	je 0x00-0xFF = 00-255															
240 0xF0																

∑ 512 Byte

(a) EEPROM

Adresse DEC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HEX	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0 0x00																
16 0x10	RR:00 (eine Reihe = 16 Byte lesen, 0x00-0xF0)															
32 0x20	rr:000204000000000006A000102000058 (32 HEX Antwort)															
48 0x30																
64 0x40																
80 0x50																
96 0x60																
112 0x70																
128 0x80																
144 0x90																
160 0xA0																
176 0xB0																
192 0xC0																
208 0xD0	1 Kästchen ≡ 1 Byte															
224 0xE0	je 0x00-0xFF = 00-255															
240 0xF0																

∑ 256 Byte

(b) RAM

Abbildung A.1: Schematische Darstellung des Speichers vom Kaffeevollautomaten

Funktion	Maschine an	Schale fehlt	Schale leeren	Tresterleeren	Wasser füllen	Gerät reinigen	Maschine spült	Abschließend spülen	Tassenbeleuchtung
Byte	03_{16}	$0E_{16}$	04_{16}	04_{16}	04_{16}	10_{16}	03_{16}	$0D_{16}$	$0F_{16}$
Bit	2: 0 → 1	2: 0 → 1	(3: 0 → 1)	5: 0 → 1	(3: 0 → 1)	7: 0 → 1	(6,3,1: 0 → 1)	3 oder 2: 0 → 1	2 oder 1: 0 → 1
Byte	05_{16}	16_{16}	$0E_{16}$	$0E_{16}$	$0E_{16}$	22_{16}	13_{16}		
Bit	5: 0 → 1	0: 1 → 0	4: 0 → 1	5: 0 → 1	6: 0 → 1	4: 0 → 1	3,1: 0 → 1		
Byte	16_{16}	29_{16}	$1B_{16}$	80_{16}	$0F_{16}$		17_{16}		
Bit	1: 0 → 1	2: 1 → 0	1: 0 → 1	1,0: 0 → 1	4: 1 → 0		1: 0 → 1		
Byte	44_{16}	69_{16}	29_{16}		$4C_{16}$		62_{16}		
Bit	0: 0 → 1	0: 1 → 0	6-3: 0 → 1		(0: 1 → 0)		(1: 0 → 1)		
Byte			$2B_{16}$		91_{16}				
Bit			6-3: 0 → 1		(1: 1 → 0)				
Byte	$1A_{16}$						68_{16}		
Bit	2: 1 → 0						(6: 0 → 1)		
Bit	1,0: 0 → 1						(5,4: 1 → 0)		

Tabelle A.2: Speicherpositionen im RAM (1)

Funktion	Filter wechseln	Hahn offen	Teeportion	Dampfbezug	Wasserdampfportion	Pulver füllen	Bohnen füllen	Zubereitung		
								immer	1. Schritt	2. Schritt
Byte	10_{16}	04_{16}	03_{16}	04_{16}	03_{16}	04_{16}	$0E_{16}$			03_{16}
Bit	5: $0 \rightarrow 1$	(3: $0 \rightarrow 1$)	(6: $0 \rightarrow 1$)	(3: $0 \rightarrow 1$)	(3: $0 \rightarrow 1$)	0: $0 \rightarrow 1$	7: $0 \rightarrow 1$			1: $0 \rightarrow 1$
Byte	22_{16}	$0F_{16}$	$0B_{16}$	$0B_{16}$	$0B_{16}$					$0B_{16}$
Bit	3: $0 \rightarrow 1$	6: $1 \rightarrow 0$	(3: $1 \rightarrow 0$)	(3: $1 \rightarrow 0$)	(3: $1 \rightarrow 0$)					(3: $1 \rightarrow 0$)
Byte	$F8_{16}$	$4C_{16}$	$0F_{16}$	13_{16}	13_{16}					17_{16}
Bit	0: $0 \rightarrow 1$	(0: $1 \rightarrow 0$)	5: $0 \rightarrow 1$	2-0: $0 \rightarrow 1$	2,0: $0 \rightarrow 1$					3: $0 \rightarrow 1$
Byte		$4D_{16}$	$4C_{16}$	49_{16}	49_{16}					62_{16}
Bit		3,1: $0 \rightarrow 1$	(0: $1 \rightarrow 0$)	(0: $0 \rightarrow 1$)	(0: $0 \rightarrow 1$)					(1: $0 \rightarrow 1$)
Byte			68_{16}	$4C_{16}$	$4C_{16}$					68_{16}
Bit			(6: $0 \rightarrow 1$)	(0: $1 \rightarrow 0$)	(0: $1 \rightarrow 0$)					(5: $1 \rightarrow 0$)
Byte	$F9_{16}$		$4D_{16}$					$0A_{16}$	03_{16}	
Bit	7,6,5,4,2: $0 \rightarrow 1$		3,1: $1 \rightarrow 0$					6: $0 \rightarrow 1$	7,4: $0 \rightarrow 1$	
	($0_{10} \rightarrow 244_{10}$)		0: $0 \rightarrow 1$					(2-0: „Kaffee“)	(7 nicht manuell)	

Tabelle A.3: Speicherpositionen im RAM (2)

Abbildung A.2: Abgefragte Speicherstellen für die API im EEPROM (1 Kästchen ≡ 1 Wort)

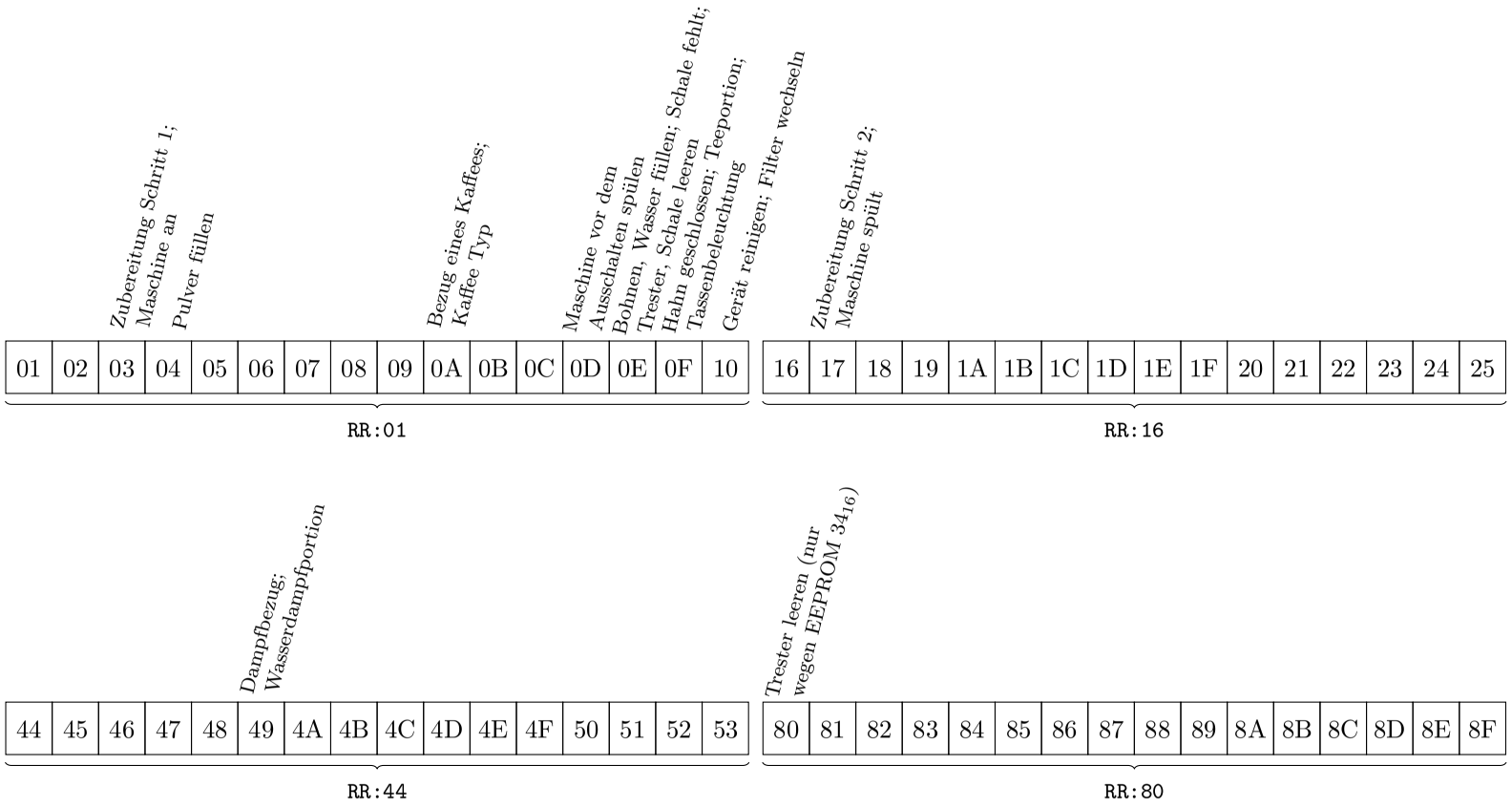


Abbildung A.3: Abgefragte Speicherstellen für die API im RAM (1 Kästchen ≡ 1 Byte)

ASCII			Kaffeevollautomat	ASCII			Kaffeevollautomat
hex	dez	Zeichen	Display	hex	dez	Zeichen	Display
	0 – 31	…	…	0x41	65	A	A
0x20	32	Leerzeichen	Leerzeichen	0x42	66	B	B
0x21	33	!	_	0x43	67	C	C
0x22	34	"		0x44	68	D	D
0x23	35	#		0x45	69	E	E
0x24	36	$		0x46	70	F	F
0x25	37	%		0x47	71	G	G
0x26	38	&		0x48	72	H	H
0x27	39	’		0x49	73	I	I
0x28	40	(		0x4A	74	J	J
0x29	41	)		0x4B	75	K	K
0x2A	42	*		0x4C	76	L	L
0x2B	43	+	+	0x4D	77	M	M
0x2C	44	,	,	0x4E	78	N	N
0x2D	45	-	-	0x4F	79	O	O
0x2E	46	.	.	0x50	80	P	P
0x2F	47	/	/	0x51	81	Q	Q
0x30	48	0	0	0x52	82	R	R
0x31	49	1	1	0x53	83	S	S
0x32	50	2	2	0x54	84	T	T
0x33	51	3	3	0x55	85	U	U
0x34	52	4	4	0x56	86	V	V
0x35	53	5	5	0x57	87	W	W
0x36	54	6	6	0x58	88	X	X
0x37	55	7	7	0x59	89	Y	Y
0x38	56	8	8	0x5A	90	Z	Z
0x39	57	9	9	0x5B	91	[	
0x3A	58	:	:	0x5C	92	\	
0x3B	59	;	;	0x5D	93	]	
0x3C	60	<	<	0x5E	94	^	
0x3D	61	=	=	0x5F	95	_	
0x3E	62	>	>	0x60	96	‘	
0x3F	63	?	?		97-127	a-z { | } ~	…
0x40	64	@					

Tabelle A.4: Verfügbarer Zeichensatz des Displays

Anhang B

Inhalt der CD

Inhalt	Beschreibung
thesis.pdf	Diese Arbeit in digitaler Form
JuraCoffeeThesis/	Die LaTeX Quelldateien
JuraCoffeeMemory/	Das entwickelte Projekt
JuraCoffeeMemory/arduino/arduino.ino	Das Arduino Skript aus [Kö18]
JuraCoffeeMemory/data/	Die aufgenommenen Speicherauszüge im JSON Format
JuraCoffeeMemory/doxygen/	Die Doxygen Dokumentation
JuraCoffeeMemory/result/	Ergebnisse der Auswertung
JuraCoffeeMemory/tools/	Kleine Helfer-Programme
JuraCoffeeMemory/website/	Die Webseite zur Visualisierung

Tabelle B.1: Inhalt der CD