HS PF

Hochschule Pforzheim

Fakultät für Technik

Studiengang Technische Informatik

Bachelorarbeit zur Erlangung des akademischen Grades

Bachelor of Engineering

Konzept und Implementierung einer SPS-Kleinsteuerung auf einem Raspberry Pi mit graphischer Programmierung und IFTTT-Funktionalität

Julian Wiche
306675

Datum: 19. Mai 2019
Erstprüfer: Prof. Dr. Karlheinz Blankenbach
Zweitprüfer: Prof. Dr.-Ing. Thomas Greiner

Zusammenfassung

In dieser Bachelorthesis soll eine speicherprogrammierbare Steuerung mittels eines Raspberry Pi nachempfunden werden. Der Fokus liegt dabei darauf, eine möglichst günstige Lösung zu schaffen, um auch Einsteigern die Möglichkeit zu bieten, einfache Steuerungsprojekte zu realisieren. Das Steuerungsprogramm hierfür soll mittels Zeichnung intuitiv in einer Weboberfläche erstellt werden können.

Schlagwörter: SPS, Kleinsteuerung, Raspberry Pi

Abstract

Concept and implementation of a PLC-minicontroller using a Raspberry Pi with a graphical programming interface and IFTTT-functionality

Goal of this bachelor thesis is to adapt a programmable logic controller using a Raspberry Pi. The main focus is to achieve an affordable solution enabling even beginners to implement trivial controlling projects. The controlling programm for this shall be creatable intuitively using a drawing tool within a web-gui.

Keywords: PLC, minicontroller, Rasperry Pi

Eidesstattliche Erklärung

Ich, Julian Wiche, Matrikel-Nr. 306675, versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema

> *Konzept und Implementierung einer SPS-Kleinsteuerung auf einem Raspberry Pi mit graphischer Programmierung und IFTTT-Funktionalität*

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Pforzheim, den 19. Mai 2019

Julian Wiche

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

Speicherprogrammierbare Steuerungen oder kurz SPS tauchen überall dort auf, wo große elektrische Maschinen eingesetzt werden. Dies ist vor allem in der Industrie der Fall. Der kleine Bruder der SPS ist die Kleinsteuerung. Sie bietet die gleichen Kernfunktionen, hat jedoch eine deutlich kleinere Anzahl an Ein- und Ausgängen. Sie werden häufig von Elektroinstallateuren eingesetzt, wenn eine klassische verbindungsprogrammierte Steuerung zum Beispiel durch Drahtbruch oder defekte Spulen in eingesetzten Relais nicht mehr korrekt funktioniert. Der Hersteller Eaton hat mit seinem Produkt *»Easy«* (siehe Abbildung 1.1) genau diese Zielgruppe im Blick. Die Programmierung erfolgt hier, als würde man klassische Schütz-Kontakte in Reihe schalten. Die Einstiegsgeräte sind relativ preiswert, doch kauft man sich in eine proprietäre Produktwelt ein, welche aufgrund von inkompatiblen Bauteilen und Bussystemen schwer wieder zu verlassen ist. So gestaltet sich die Erweiterung einer bestehenden Steuerung um die Möglichkeit einer Fernabfrage über das Internet als nahezu unmöglich, oder setzt den Austausch der kompletten Steuerung voraus. Dabei sind Ein- und Ausgänge doch eigentlich das Gleiche wie an jedem Raspberry Pi vorhandene GPIOs. Auf Basis dieser Überlegung und günstigen Preisen entstand die Idee, eine Lösung mittels Raspberry Pi zu erarbeiten.

Abbildung 1.1: Eaton Easy Kleinsteuerung[1]

1.1 Ziel der Arbeit

Ziel dieser Arbeit ist es, eine möglichst günstige Möglichkeit zu schaffen, um eine Steuerung zu realisieren, welche intuitiv programmiert werden kann und die grundsätzliche Funktion der vorher eingesetzten Easy Steuerung um die Möglichkeit zur Fernabfrage über das Internet und weitere Funktionen erweitert. Dabei soll das Projekt mittels Git[2] versioniert werden, um es anderen Entwicklern auf GitHub als Open-Source Software zur Verfügung zu stellen. Dabei wird das Projekt als MIT lizenziert, was eine Modifikation sowie private und gewerbliche Nutzung und Verbreitung ausdrücklich gestattet.

1.2 Verwandte Arbeiten

Bei der Recherche nach weiteren Projekten, welche auf Basis eines Raspberry Pis eine speicherprogrammierbare Steuerung realisieren, fand sich zum einen das kommerzielle Projekt Codesys[3]. Zum anderen ist auch das Projekt Open-PLC zu

nennen[4].

1.3 Aufbau der Arbeit

In dieser Arbeit werden zunächst einige grundsätzliche Fragen der Steuerungstechnik thematisiert. Hierbei wird besprochen, weshalb eine speicherprogrammierbare Steuerung überhaupt benötigt wird, wie man sie einsetzt und wie die gängige Praxis zum Erstellen von Steuerungsprogrammen ist. Weiterhin werden einige eingesetzte Technologien in Hard- und Software beschrieben. Zuletzt wird auch auf einige organisatorische Dinge, wie die Versionsverwaltung oder die Entwicklungsumgebung, eingegangen. Kapitel *»3 Umsetzung«* beschreibt dann, wie die Arbeit praktisch umgesetzt wurde und geht dabei auf die logische Unterteilung der Komponenten sowie deren Zusammenspiel und die darin eingesetzten Algorithmen ein. Das letzte Kapitel beschreibt bis wohin die Entwicklung gelangt ist und welche Fehler noch bestehen. Außerdem wird hier darauf eingegangen, wie die ordnungsgemäße Funktion überprüft werden kann und es weist auf mögliche Szenarien hin, die zu Fehlern führen könnten. Im Anhang bietet sich ein Überblick über die beigelegte DVD. Außerdem lassen sich die Softwareversionen der eingesetzten Tools hier nachlesen.

2 Grundlagen

In diesem Kapitel wird zunächst erörtert, was genau eine speicherprogrammierbare Steuerung eigentlich ist, wo sie eingesetzt wird und wieso man sie benötigt. Des Weiteren wird am Beispiel einer Easy Kleinsteuerung beschrieben, wie die Erstellung eines Logikprogramms vonstattengeht. Im weiteren Verlauf wird ein Logikprogramm vorgestellt und es wird erklärt, in welche Teile die Funktion aufgeteilt ist. Zuletzt bietet dieses Kapitel auch Einblick in die Rahmenbedingungen der Arbeit, wie die Versionsverwaltung, die Entwicklungsumgebung sowie eingesetzte Softwarebibliotheken.

2.1 Prinzip speicherprogrammierbare Steuerung

Die grundsätzliche Funktion einer speicherprogrammierbaren Steuerung oder `SPS` ist die Ermittlung der Ausgangswerte durch eine logische Verknüpfung der Eingangswerte[5]. Wie im Beispiel der Abbildung 2.1 zu sehen, könnte ein an einen Eingang angeschlossener Schalter als Sensor dienen. Als Aktor könnte eine Leuchte Verwendung finden. Der Benutzer der Steuerung muss nun durch eine Logik für jeden Ausgang festlegen, in welchen Fällen dieser aktiv sein soll. Dies geschieht in Form eines Steuerungs- bzw. Logikprogramms, das der Benutzer der Steuerung erstellt. Doch wieso schließt man dann nicht einfach die Leuchte direkt an den Schalter an? Dies wäre bei einer einfachen Lampensteuerung sicherlich zu bevorzugen, jedoch handelt es sich bei den Szenarien, die mit einer solchen Steuerung realisiert werden, für gewöhnlich um deutlich komplexere Verschaltungen. Bei der klassischen Installation für eine Torsteuerung beispielsweise wären mehrere elektromechanische Relais, auch Schütze genannt, nötig. Zudem bedürfte ein automatisches Schließen des Tores ein Zeitrelais. Der Verdrahtungsaufwand und Platzbedarf wären relativ hoch. Führt man stattdessen jedoch alle benötigten Sensoren auf eine speicherprogrammierbare Steuerung, wird der Verdrahtungsaufwand erheblich reduziert, was zu einer höheren Übersichtlichkeit führt und weniger Potential für Fehler birgt. Auch zieht eine Änderung im logischen Verhalten der Steuerung dann für gewöhnlich keinerlei Änderungen an der Verdrahtung mehr nach sich. Zuletzt sind auch die Kosten für speicherprogrammierbare Steuerungen inzwischen auf einem

Niveau, das klassische Steuerungen unwirtschaftlich macht.

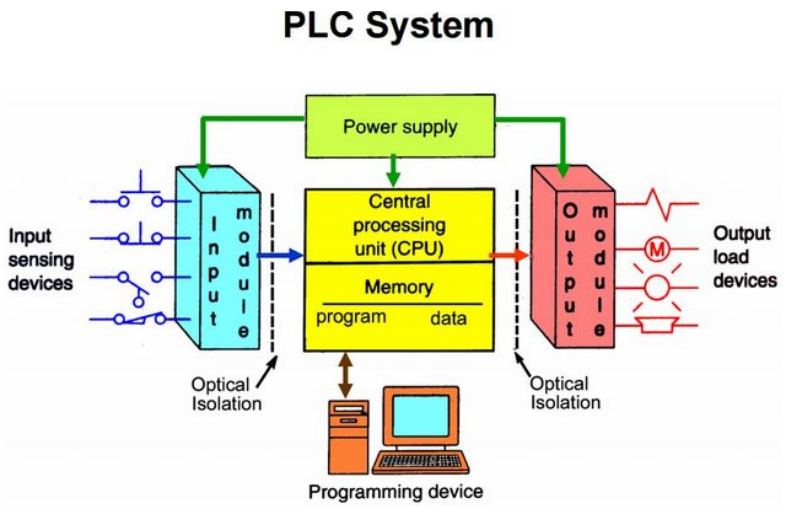

Abbildung 2.1: Blockdiagramm SPS[6]

2.2 Ausgangssituation

Als Vorbild für dieses Projekt dient die Kleinsteuerung Easy vom Hersteller Eaton. Das Einstiegsmodell bietet acht Ein- und vier Ausgänge. Das Logikprogramm, welches die Eingänge der Steuerung logisch mit den Ausgängen verbindet, wird auf einem kleinen Display direkt am Gerät erstellt[7]. Dabei stehen neben den physikalischen Ein- und Ausgängen auch Zeitfunktionen oder Zählerbausteine zur Verfügung. Im Programmiermodus, welcher in Abbildung 2.2 dargestellt ist, wird links ein Pluspol und rechts ein Minuspol symbolisiert. Der anzusteuernde Ausgang steht dabei stets ganz rechts. Der Strompfad kann nunmehr bis zum Pluspol durch gezeichnet, oder aber durch Sensoren unterbrochen und verzweigt werden. Aus diesem Schaltplan werden dann die booleschen Gleichungen gewonnen, welche die Steuerung im Betrieb durchläuft, um die Werte der Ausgänge zu bestimmen.

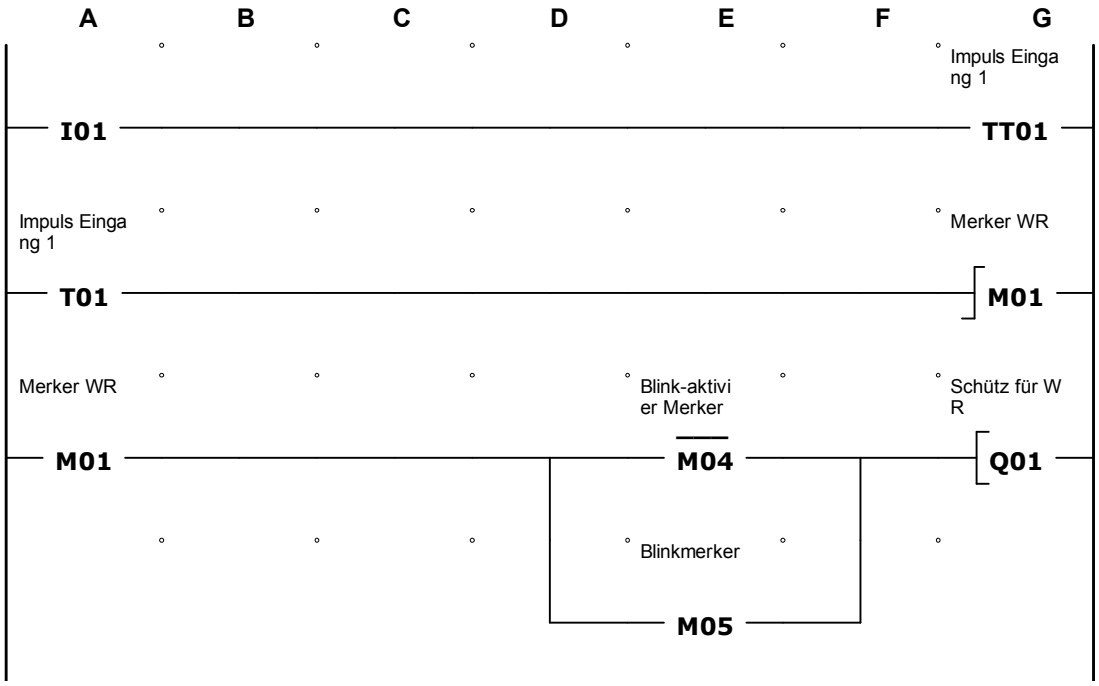

Abbildung 2.2: Programm einer Easy Kleinsteuerung

2.2.1 Benutzeroberfläche

Eine ähnliche Vorgehensweise bei der Erstellung des Logikprogramms (siehe Unterabschnitt 2.2) ist auch in diesem Projekt geplant. Da der Raspberry Pi netzwerkfähig ist, wurde jedoch statt eines Displays am Gerät eine Benutzeroberfläche gewählt, welche im Internetbrowser bedienbar ist. Als Basis für die Programmieroberfläche wurde das Projekt CircuitVerse[8] (siehe Unterunterabschnitt 3.9.3) herangezogen. Hierbei handelt es sich um einen Logiksimulator, in welchem komplexe Logikschaltungen durch Drag&Drop erstellt werden können. Das quelloffene Projekt ist auf GitHub[9] verfügbar und dank der MIT Lizenz zur Erweiterung und Modifikation freigegeben. Dabei musste das Projekt vor allem durch eine Funktion ergänzt werden, um die erstellte Logik in einem Format zu exportieren, welche vom Backend verstanden wird. Weiterhin mussten die zur Verfügung stehenden Bauteile dahingehend modifiziert werden, dass nur Schaltungen erstellt werden können, die auch im Backend implementiert sind. Im Vorbild der Easy Kleinsteuerung kann der Schaltplan auch Laufzeitinformationen wiedergeben. So wird ein (symbolisch)

unter Spannung stehender Zweig als breite Linie dargestellt, während unbestromte Zweige schmal gezeichnet werden. Diese Laufzeitinformationen sollen in dieser Bachelorarbeit ebenfalls dargestellt werden. Dazu wird für jeden Ein- und Ausgang eine Grafik eingeblendet, welche erkennen lässt, ob der Eingang bzw. der Ausgang an oder aus ist.

2.2.2 Backend

Als Schnittstelle zwischen Hardware und Bedienoberfläche wird eine Software eingesetzt, die das vom Benutzer erstellte Logikprogramm kontinuierlich durchläuft und somit sicherstellt, dass eine Änderung an einem Eingang, der Ablauf eines Timers etc. die Werte der davon abhängigen Ausgänge entsprechend verändert. Wie dies im Vorbild der Easy Kleinsteuerung gelöst wird, besteht kein Einblick.

2.2.3 Hardware

Wie schon vorab beschrieben, bildet ein Raspberry Pi die Grundlage für dieses Projekt. Dieser bietet von Haus aus einige GPIOs, welche dazu verwendet werden können, um Sensoren abzufragen oder um Aktoren anzusteuern. Jedoch fiel die Entscheidung darauf, eine Erweiterungskarte (HAT) zu diesem Zweck einzusetzen. Dies dient zunächst zum Schutz des Raspberry Pi, zudem bietet die eingesetzte Erweiterungskarte auch Leuchtdioden an den Ausgängen sowie Taster an den Eingängen, was das Testen erheblich vereinfacht. Da der Anschluss per SPI erfolgt, können theoretisch mehrere solcher Boards parallel betrieben werden. Dies wird jedoch dadurch verhindert, dass die Erweiterungskarte die verwendeten Anschlüsse abdeckt. Ein Adaptermodul, welches die Anschlüsse vervielfacht, findet sich zwar im Internet, ist jedoch nicht mehr erhältlich.

2.3 Versionsverwaltung

Da eine Versionsverwaltung für ein Projekt dieser Größe als notwendig erachtet wurde, fiel der Gedanke auf GIT.[10] Git ist eine dezentrale Versionsverwaltung, die Notwendigkeit eines Versionierungsservers entfällt hierdurch. Jedoch ist es in Git

möglich, ein oder mehrere sogenannte Remotes hinzuzufügen. Das sind entfernte Git-Repositorys, die mit dem lokalen Repository synchronisiert werden können. In der Praxis wird Git häufig eingesetzt, weshalb sich diese Bachelorarbeit als Einarbeitung anbot. Zunächst wurde ein lokales Git Repository erstellt, welches in Folge mit einem Remote-Repository auf GitHub verbunden wurde. Angedacht war ein Development-Branch und jeweils ein Feature-Branch, welches nach Vollendung des entsprechenden Features in den Development-Branch zurückgeführt werden soll. Darüber hinaus soll ein Tagging erfolgen. Dabei soll für jeden Zeitbalken des Gantt Diagramms, welches in der vorhergehenden Projektplanung erstellt wurde, ein Tag gesetzt werden.

2.4 Entwicklungsumgebung

Ein Problem, das dieses Projekt bietet, ist, dass nicht direkt auf dem Zielsystem entwickelt wird. Das hat zur Folge, dass das Programm entweder auf dem Entwicklungssystem crosskompiliert werden muss, oder die Quelldateien auf das Zielsystem kopiert werden müssen, um sie dort zu kompilieren. Da das Aufsetzen eines Crosskompilers inklusive der dazu nötigen Toolchain als zu aufwändig erachtet wurde, blieb nur der Weg, das Projekt direkt auf dem Zielsystem zu übersetzen. Um sich die Arbeit zu erleichtern, wurde nach einer Entwicklungsumgebung recherchiert, die ein Übersetzen über eine SSH Verbindung zulässt. NetBeans[11] bietet dies an, dabei wird entweder das gesamte Projekt beim Übersetzen per SSH auf das Zielsystem kopiert und dort übersetzt, oder es kann ein Ordner angegeben werden, an den eine Systemfreigabe eingehängt ist (z.B. über eine Samba-Freigabe). Im Laufe des Projektes wurde eigens dafür ein Samba-Server auf dem Zielsystem installiert. Von nun an konnte also in der Entwicklungsumgebung direkt auf den Dateien des Raspberry Pi gearbeitet werden, wobei per Tastendruck eine SSH Verbindung aufgebaut wurde, um auf dem Zielsystem den Befehl *»make«* auszuführen. Dabei werden sämtliche Ausgaben des Compilers in einem Fenster innerhalb der Entwicklungsumgebung angezeigt. Auch Debugging ist auf diesem Wege möglich.

2.5 C++ Library LibPiFace

Wie im Abschnitt 2.2.3 beschrieben, bildet die Erweiterungskarte PiFace Digital 2[12] die Grundlage für diese Bachelorarbeit. Im Lieferumfang befindet sich eine in C geschriebene Bibliothek inklusive eines lauffähigen Tests, welcher ebenfalls auf GitHub[13] zu finden ist. Diese Bibliothek stützt sich wiederum auf die Bibliothek libmcp23s17[14], welche den verbauten SoC über SPI anspricht. Im Laufe der Arbeiten fiel jedoch auf, dass die C Bibliothek nicht alle benötigten Funktionen enthielt. Da der Quellcode vorlag und die Lizenzierung Veränderungen am Quellcode zulässt, lag die Überlegung nahe, die benötigten Funktionen direkt in der Bibliothek zu ergänzen anstatt sie im eigentlichen Projekt unterzubringen. Weiterhin schien es auch ein erstrebenswertes Lernziel zu sein, das Erstellen und Übersetzen von statisch bzw. dynamisch gelinkten Bibliotheken kennenzulernen. Zuletzt wurde es schlichtweg als die sauberste Lösung erachtet. Zunächst wurde angenommen, dass sich auch mehrere Hardwaremodule per SPI mit einem einzelnen Raspberry Pi verbinden lassen. Dies ist technisch auch möglich, so bieten die eingesetzten Boards die Möglichkeit, über einen Jumper eine Hardwareadresse einzustellen. Der Hersteller bot auch die passende Hardware an, um mehrere Boards mit einem Raspberry Pi zu verbinden. Jedoch wurden diese scheinbar mangels Nachfrage aus dem Sortiment genommen. Obwohl eine Bastellösung es immer noch ermöglichen würde, ist der Aufwand hierfür sehr hoch und scheint unwirtschaftlich. Leider wurde bis zu dieser Erkenntnis schon einiges an Energie hinein investiert, mehrere Boards gleichzeitig zu unterstützen. Dies ist auch der ursprüngliche Grund, wieso ein objektorientierter Ansatz in C++ gewählt wurde - eine Instanz für jedes Hardware-Modul. Ein weiterer Grund war, dass die Anwendung ohne Caching nicht performant genug war. Das heißt, die zeitliche Lücke zwischen einer Änderung an einem Eingang bis zu dessen Auswirkung am Ausgang war deutlich spürbar und damit inakzeptabel. Dafür wurden Methoden vorgesehen, um das Caching ein- und auszuschalten - bei eingeschaltetem Caching verändern die Methoden, Bytes und Bits zu schreiben, lediglich den Wert einer Instanzvariable. Derzeit muss das Leeren des Caches explizit mit einem Aufruf der Methode *flush()* erfolgen. Über ein automatisches Verfahren wurde nachgedacht, jedoch erwies sich der manuelle Aufruf als einfacher.

2.6 WebSocket

Damit die in 2.2.1 beschriebene Benutzeroberfläche Statusinformationen von der Steuerung erhalten kann, muss ein Kommunikationskanal zwischen dem Internetbrowser des Anwenders und der Steuerung geschaffen werden. Eine Kommunikation per HTTP ist dabei jedoch wenig flexibel, da dieses Protokoll zustandslos ist. Das bedeutet, dass eine Verbindung nach Bearbeitung einer Anfrage sofort wieder abgebaut wird. Es ist also per se nicht möglich, dass der Server den Client beim Eintreffen neuer Daten benachrichtigt. Eine Lösung hierfür wäre, die Anwendung im Browser so zu schreiben, dass sie den Server zyklisch anfragt. Dabei muss ein Kompromiss aus Serverbelastung, also kleinen Abfrageintervallen, und einer annehmbaren Verzögerung gefunden werden. Eine Verbesserung bringt `Long-Polling`. Hierbei wird auf Clientseite auf die gleiche Weise vorgegangen, der Server blockiert jedoch Anfragen bis zur maximalen Zeit eines Timeouts. Erhält der Client bis zum Timeout keine Daten, so stellt er nach dem Timeout erneut eine Anfrage. Bei diesen Methoden ist die Serverbelastung relativ hoch und beinhaltet eine gewisse Latenz, wodurch Anwendungen weniger schnell auf Änderungen reagieren können. WebSockets[15] hingegen ermöglichen eine bidirektionale Kommunikation zwischen Server und Client, wobei die Latenz minimal ist. Dies begründet sich unter anderem dadurch, dass eine WebSocket-Verbindung persistent ist. Das heißt, sie wird einmalig aufgebaut und bleibt dann bestehen bis eine Seite die Verbindung beendet. Dadurch sind nur zum Aufbau der Verbindung Header nötig, was die zu übermittelnde Datenmenge drastisch reduziert. In der Praxis besteht ein HTTP Header, welcher jeder Anfrage mitgegeben werden muss, oft aus mehr als 100 Bytes. Eine WebSocket-Anfrage benötigt einen ebenso großen Header, jedoch nur um die Verbindung aufzubauen. Ist diese hergestellt, so werden lediglich zwei Byte je Übertragung fällig, um die Verbindung zu steuern. Das steigert nicht nur die Verbindungsgeschwindigkeit, sondern entlastet gleichzeitig den Server.[16]

2.7 Installation von Abhängigkeiten

Wie sich im Laufe des Projektes herausstellte, benötigten einige Programmteile Funktionen aus der Boost-Bibliothek.[17] Auf dem Desktop-Computer, auf dem

die Lösung getestet wurde, konnte das Projekt dank installiertem Boost-Paket ohne Probleme übersetzt werden. Da es sich bei dem Zielsystem jedoch um eine ARM Architektur handelt, musste der Programmcode dort Übersetzt werden. In den Paketquellen des dort installieren Raspbian ist jedoch eine ältere Version des Boost-Pakets hinterlegt, was dazu führt, dass Boost manuell heruntergeladen und gebaut werden muss. Obwohl sich dieses Problem im Laufe der Zeit durch Aktualisierung des Paketes in den Paketquellen von Raspbian von selbst lösen wird, muss dennoch geprüft werden, ob die installierte Version den Ansprüchen genügt. Hierfür wurde ein Bash-Script erstellt, welches auch alle weiteren Abhängigkeiten prüft und gegebenenfalls installiert. Dazu zählen auch die im Abschnitt 6 Anhang aufgeführten Bibliotheken, um die Hardware anzusprechen, und der verwendete Kompiler.

3 Umsetzung

In diesem Kapitel wird zunächst darauf eingegangen, wie das Logikprogramm für die Steuerung definiert wurde und wie dieses verarbeitet werden soll. Weiterhin wird erklärt, wie die verfügbaren Ein- und Ausgänge logisch unterteilt werden und mit welchem Namensschema sie adressiert werden sollen. Es wird erläutert, mit welcher Klassenstruktur diese Elemente im Backend implementiert wurden. Nachdem dann etwas genauer auf die einzelnen Ableitungen dieser Elemente eingegangen wurde, wird der Aufbau des Programms und dessen Kern beschrieben und anhand von Quelltextausschnitten erklärt. Den Übergang zum letzten Teil bildet die Erklärung des WebSocket-Servers, welcher die Schnittstelle zu der im letzten Unterabschnitt beschriebenen Benutzeroberfläche bildet. Hier wird auch die Statusanzeige, den Logik-Editor und die jeweilige Anbindung ans Backend beschrieben.

3.1 Aufbau und Parsing der Logikgleichungen

Das Ziel dieses Projektes ist es, dass die Ausgänge der Steuerung in Abhängigkeit von Eingängen wie zum Beispiel physikalischen Eingängen oder Timer-Bausteinen ein- bzw. ausgeschaltet werden. Dafür ist in einer Textdatei für jeden Ausgang eine Zeile vorgesehen. Diese Zeile kann auch als boolesche Gleichung betrachtet werden, denn jede Zeile beginnt mit dem zu definierenden Ausgang, also zum Beispiel `Ho1`, worauf ein Gleichheitszeichen zu folgen hat. Der gesamte Ausdruck hinter dem Gleichheitszeichen wird zur Laufzeit des Programms durchlaufen, wobei jedes vorkommende Paar von [und] durch eine Null oder eine Eins ersetzt wird. Innerhalb der Klammern findet sich genau wie bei dem Bezeichner vor dem Gleichheitszeichen die jeweilige Bezeichnung der Abhängigkeit. Lautet die Zeile also etwa Ho0 = [Hi0] & [Hi1], so wird die komplette erste Klammer durch den Wert von Hi0 ersetzt, während die zweite Klammer durch den Wert von Hi1 ersetzt wird. Daraus ergibt sich dann, vorausgesetzt Hi0 und Hi1 sind im Zustand *»Ein«*, Ho0 = 1 & 1. Das für den Leser offensichtliche Ergebnis dieser Gleichung ist 1 oder *»true«*. Jedoch gestaltet sich eine programmatische Lösung des Problems als deutlich komplexer. Denn sobald mehr als drei Ausdrücke im Spiel sind, müssen wie bei der klassischen Mathematik Rechenregeln befolgt werden. Punkt vor Strich ist in der booleschen

Mathematik mit einem ∧(und) vor ∨(oder) gleichzusetzen. Außerdem müssen Klammern erkannt und richtig aufgelöst werden. Dabei kann ein Ausdruck beliebig komplex sein. Eine Recherche nach Ansätzen führte zu Stackoverflow.[18] Dieser Ansatz löste genau das Problem und wurde somit in das Projekt übernommen.

3.2 Benamungsschnema

Nachdem die Auswertung beziehungsweise das Parsing der Logikausdrücke umgesetzt wurde, sollte auch die Auswertung der Bezeichner automatisiert werden. Ein Benamsungsschema wurde dabei schon vorher erdacht. Es besteht aus einem führenden Großbuchstaben, gefolgt von einem Kleinbuchstaben und einer Zahl. Dabei wird der führende Buchstabe als Kanal bezeichnet, der zweite als Entität und die Ziffer als Pin-Nummer. So könnte zum Beispiel der Buchstabe *»H«* den Kanal Hardware beschreiben, welcher wiederum die Entitäten *»i«* für Input und *»o«* für Output besitzt, welche jeweils ein Byte, also acht Bits oder Pins, umfassen.

3.3 Klassenstrukrur und Kanäle

Im Programm werden die zuvor beschriebenen Kanäle von einem Abkömmling der Basisklasse *»IO_Channel«* (siehe Abb. 3.2) repräsentiert, wobei jeder Kanal als Eigenschaft eine Map mit Entitäten führt, welche durch Objekte des Typs *»Channel_Entity«* (siehe Abb. 3.3) repräsentiert werden. Eine Entität wiederum weiß, wie groß ihre Breite ist, also wie viele Bits sie hat, und ob sie nur lesbar oder auch beschreibbar ist. Eine weitere Klasse *»IO_Channel_AccessWrapper«* (siehe Abb. 3.1) bündelt alle vorhandenen Kanäle inklusive der jeweiligen Entitäten. Zudem erleichtert sie mittels Überladung der Array-Operatoren den Zugriff. Ein Zugriff ist dann wie in Quellcode 3.2 gezeigt möglich. Dazu müssen die entsprechenden Kanäle vorher wie in Quellcode 3.1 zu sehen initialisiert und einer Instanz der Klasse *»IO_Channel_AccessWrapper«* übergeben werden. Der für den späteren Zugriff auf den Kanal nötige Buchstabe wird in diesem Zuge festgelegt, während die Buchstaben der untergeordneten Entitäten Bestandteil des jeweiligen Kanals sind und dementsprechend dort definiert werden.

```
    chnl.insert(std::make_pair('T', IOChannelPtr(new IO_Channel_Virtual_Timer())));
    chnl.insert(std::make_pair('P', IOChannelPtr(new IO_Channel_Virtual_Pipe("r7123d97a3", 0x07))));


                                // chnl is copied here!
    commandProcessor cp(isg, chnl, webSocketSessions);
```

Quellcode 3.1: Initialisieren der Kanäle und Entitäten

```
    // Initially read the timers config
    std::string fn_timers = "timers.conf";
```

Quellcode 3.2: Zugriff auf Kanal und Entität exemplarisch

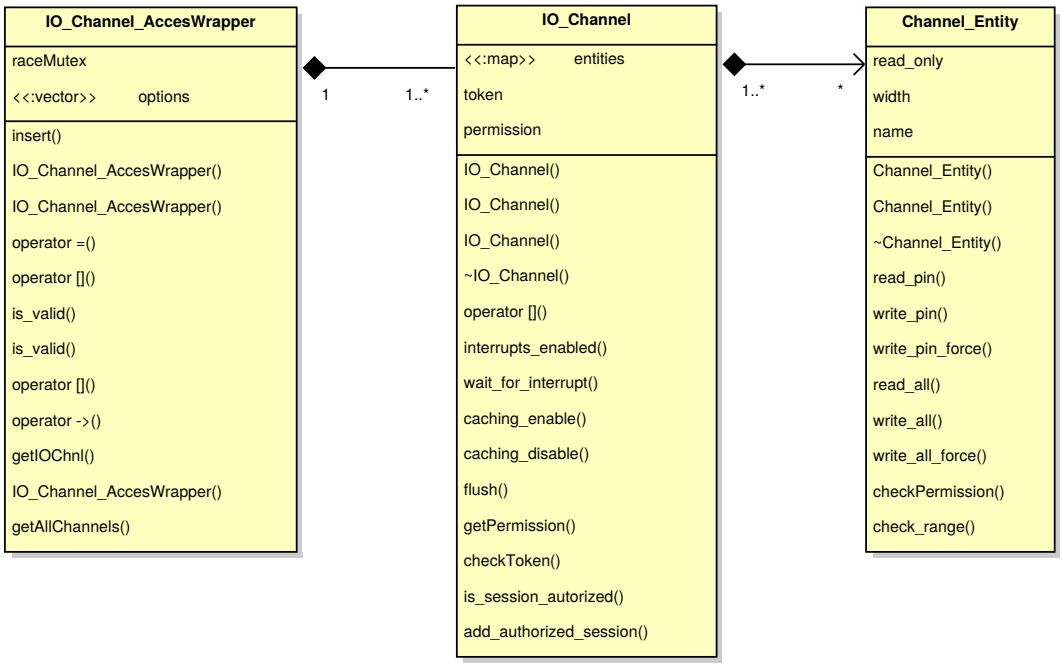

Abbildung 3.1: Klassendiagramm Aggregation der Klassen

Zu sehen ist die Aggregation zwischen dem *»IO_Channel_AccessWrapper«*, den Kanälen *»IO_Channel«* und den Entitäten *»Channel_Entity«*. Alle Kanäle werden von dem *»IO_Channel_AccessWrapper«* zusammengefasst. Ein Zugriff erfolgt ausschließlich über diesen Weg.

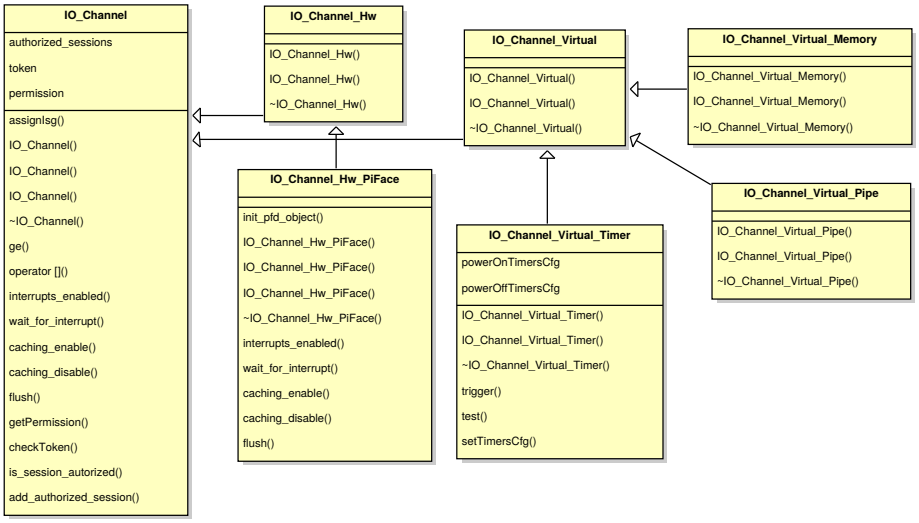

Abbildung 3.2: Vererbungshierarchie der Basisklasse IOChannel

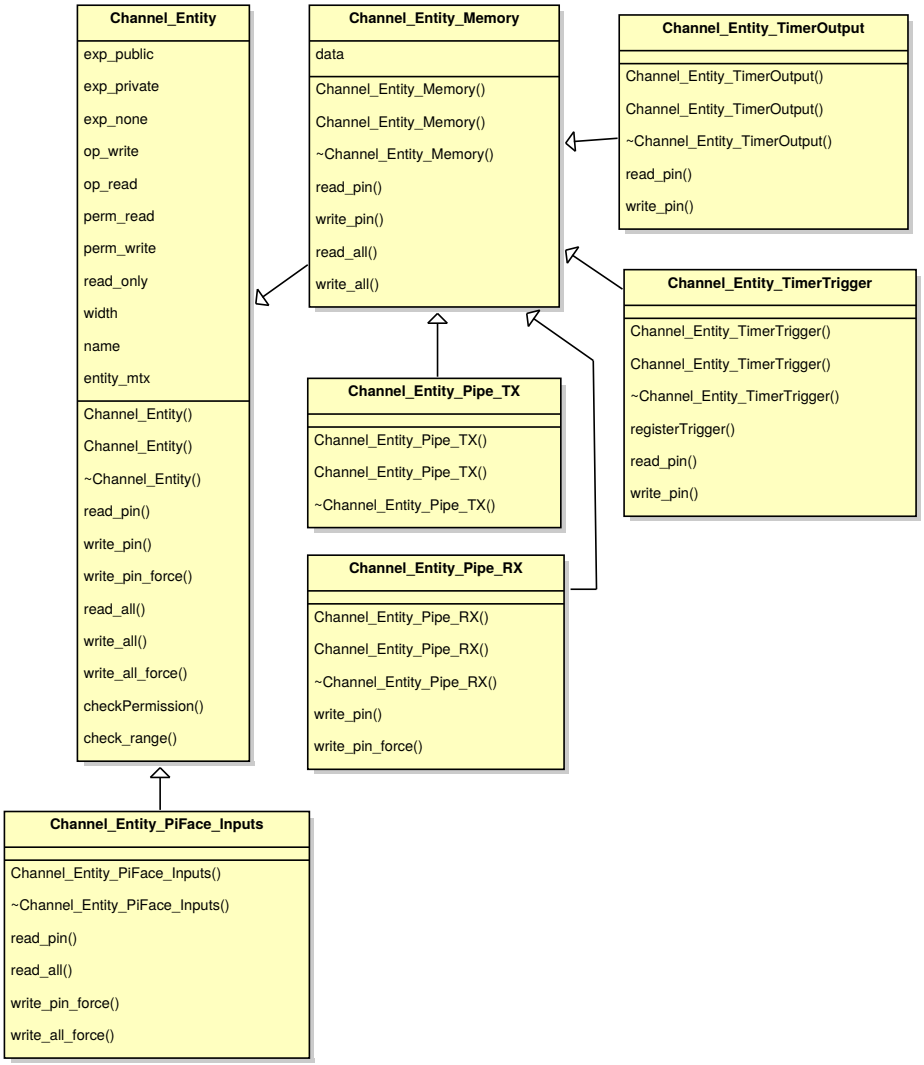

Abbildung 3.3: Vererbungshierarchie der Basisklasse ChannelEntity

Die Entitäten erben von der Basisklasse Entity. Sie überschreiben Methoden, um entweder lesend oder schreibend, und um entweder auf einmal oder bitweise auf den Inhalt zuzugreifen. Dafür sind die Methoden read_pin, write_pin bzw. read_all und write_all vorgesehen.

3.3.1 Hardware Kanäle

Der wohl wichtigste Kanal ist der Hardware Kanal. Die Klassenstruktur in Abb. 3.2 zeigt, dass alle Kanäle eine gemeinsame Basisklasse haben. Der Hardware Kanal ist eine Spezialisierung der Basisklasse, welche spezifische Funktionen für die entsprechende Hardware beherbergt. So finden sich hier Methoden zur Steuerung des Caching sowie ein blockierender Aufruf, welcher auf einen Interrput an der Hardware reagiert. Die Funktionen für den schreibenden und lesenden Zugriff auf die Ein- und Ausgänge der Hardware sind in Entitäten gekapselt. Für eben jenen Zweck existieren zwei Spezialisierungen der *»Channel_Entity«* (siehe Abb. 3.3) Basisklasse, welche für den Zugriff mit *»i«* für den Eingang (input) beziehungsweise *»o«* für den Ausgang (output) referenziert werden.

3.3.2 Merker Bausteine

Die erste Erweiterung, welche wie in Abb. 3.1 zu sehen mit dem Buchstaben M referenziert wird, sind Memory Bausteine. Da ein Memory Baustein nicht richtungsorientiert ist, das heißt, es existiert keine Eingangs- oder Ausgangsgröße, wurde hier vom Benamsungsschema für Entitäten abgewichen. Eine Referenzierung ist hier mittels eines oder mehrerer Buchstaben geplant. Somit bietet ein Merker Baustein Platz für acht Zustandswerte pro verwendeter Entität. Bislang ist lediglich die Entität *»a«* angelegt, wobei weitere Buchstaben in der Konfigurationsdatei angelegt werden können.

3.3.3 Timer Bausteine

Timer Bausteine waren die wohl komplexesten Bausteine. Eine Verzögerung im Programmablauf bedeutet entweder ein Blockieren, was aber dazu führen würde, dass das Programm in dieser Zeit auch nichts anderes tun und somit nicht auf Ereignisse reagieren kann. Die andere Alternative ist Multithreading, also das Einführen von parallelen Handlungssträngen, welche weitere Probleme mit sich bringen. So muss zum Beispiel beim Zugriff auf gemeinsam genutzte Ressourcen dafür Sorge getragen werden, dass nicht gleichzeitig gelesen wird, während gerade geschrieben wird. Des Weiteren musste die Hauptschleife überarbeitet werden. Bislang wurde

die Programmlogik bei jedem Eintreten eines Hardware-Ereignisses genau einmal durchlaufen. Jedoch muss die Programmlogik nun auch durch Ablauf eines Timers erneut durchlaufen werden. Gelöst wurde dies mithilfe einer binären Variable als Schalter, welche zusammen mit einem Mutex zur Vermeidung von gleichzeitigem Zugriff und einer Condition-Variable zur Benachrichtigung des Hauptprozesses in die Klasse *»iterationSwitchGuard«* gekapselt wurde. Ein Timer Baustein (mehrere sind theoretisch möglich) bietet acht konfigurierbare Timer. Dabei gibt es eine Entität, um einen Timer auszulösen *»t«* (Trigger), und eine Entität, welcher als Ausgang fungiert *»o«* (Output). So ergibt sich in der Programmlogik die gleiche Syntax, welche sich auch bei Hardwarekanälen bietet: Ein exemplarisches Triggern des Timers 3 würde erfolgen, indem eine Zeile der Logikdatei mit *»[Tt3=]«* beginnt. Eine Verwendung des Ausgangswertes des gleichen Timers, würde durch Integrieren des Bezeichners *»[To3]«* in die Logikzuweisung eines anderen Bezeichners erfolgen. Die Parametrisierung des Timers erfolgt durch Einbinden einer Konfigurationsdatei (siehe Quellcode 3.3). Hierbei lassen sich für jeden Timer eine Einschaltverzögerung sowie eine Ausschaltverzögerung definieren. Jeder Wert ungleich Null bewirkt eine Verzögerung, eine gleichzeitige Nutzung von Ein- und Ausschaltverzögerungen ist möglich. Werte gleich Null bewirken eine sofortige Änderung am Ausgang, sobald sich der Eingangswert verändert. Beim Einlesen der Konfigurationsdatei werden sämtliche Leerzeichen entfernt. Zudem kann mit einem Semikolon *»;«* oder einer Raute *»#«* ein Kommentar eingeleitet werden, welches alle restlichen Zeichen der Zeile beim Einlesen entfernt. Die Verzögerungszeiten sind in Millisekunden anzugeben, wobei die Reihenfolge nicht sukzessive erfolgen muss. Der Maximalwert beläuft sich auf 4.294.967.295, was die größte in unsigned long darstellbare Zahl ist und umgerechnet etwa 49 Tagen entspricht.

```
timer =T 0
powerOnDelay=5000
powerOffDelay=0

timer=T7
powerOnDelay=1000
powerOffDelay=1000

timer=T1
powerOnDelay=10000
powerOffDelay=4294967295
```

Quellcode 3.3: Beispiel der Timer Konfigurationsdatei

3.3.4 Virtuelle Kanäle

Virtuelle Kanäle sind ein Konzept, das die Kommunikation der Steuerung mit entfernten Endpunkten ermöglicht. Ein virtueller Kanal ist hierbei lediglich ein veränderter Memory Baustein, welcher entgegen eines normalen Memory Bausteins wieder richtungsorientiert ist. Das heißt, es gibt eine Eingangsentität und eine Ausgangsentität. Hierbei muss beachtet werden, dass die Perspektive so gesetzt ist, dass ein Eingang die extern empfangenen Daten beinhaltet, während in den Ausgang geschrieben wird. Die Gegenseite bildet der in Unterabschnitt 3.8 beschriebene WebSocket-Server beziehungsweise was auch immer die Gegenseite bildet. Das kann die Benutzeroberfläche dieses Projektes sein (siehe Unterunterabschnitt 3.9.2) oder auch jeder andere WebSocket-fähige Client. Angedacht ist auch die Implementierung eines WebSocket-Clients in das Backend, womit sich zwei Einheiten verbinden ließen.

3.4 Laden der Programmlogik aus Datei

Das Logikprogramm wurde im nächsten Schritt in eine Textdatei ausgelagert. Sie soll künftig von der grafischen Benutzeroberfläche automatisch erstellt werden können, kann aber nach wie vor auch von Hand erstellt werden. Die Datei enthält einen Ausgang pro Zeile, gefolgt von einem Gleichheitszeichen und den Abhängigkeiten. Zeichen, welche auf ein Semikolon folgen, werden dabei als Kommentar gewertet und ausgelassen. Sie wird bei Programmstart eingelesen und verbleibt dann im Speicher. Zum Speichern der Daten wird ein *»Vector«* benutzt, welcher für jede

Zeile der Logikdatei einen weiteren Eintrag erhält. Da vor einem Gleichheitszeichen lediglich ein Ausgang stehen darf (z.B. **??** Zeile 1: *»Ho0«*), ist somit jeder Eintrag des Vektors die Definition genau einer abgeschlossenen Zuweisung. Bei mehrfacher Definition eines Ausgangs, überschreibt die spätere Definition die vorangegangene.

```
Ho0 = [Hi0] | [Hi1] |  [Hi2]  | [Hi3] | [Hi4] | [Hi5] | [Hi6] | [Hi7]
Ho1 = [Hi0] | [Hi1] |  [Hi2]  | [Hi3] | [Hi4] | [Hi5] | [Hi6] | [Hi7]
Ho2 = [Hi0] | [Hi1] |  [Hi2]  | [Hi3] | [Hi4] | [Hi5] | [Hi6] | [Hi7]
Ho3 = [Hi0] | [Hi1] |  [Hi2]  | [Hi3] | [Hi4] | [Hi5] | [Hi6] | [Hi7]
Ho4 = [Hi0] | [Hi1] |  [Hi2]  | [Hi3] | [Hi4] | [Hi5] | [Hi6] | [Hi7]
Ho5 = [Hi0] | [Hi1] |  [Hi2]  | [Hi3] | [Hi4] | [Hi5] | [Hi6] | [Hi7]
Ho6 = [Hi0] | [Hi1] |  [Hi2]  | [Hi3] | [Hi4] | [Hi5] | [Hi6] | [Hi7]
Ho7 = ( [Hi0] & [Hi1] ) |  ( [Hi2]  & [Hi3] )  | ( [Hi4] & [Hi5] ) | ( [Hi6] & [Hi7] )
```

Quellcode 3.4: Beispiel der Programmlogik-Datei

3.4.1 Erneutes Laden der Logik zur Laufzeit

Zum erneuten Einlesen der Logik wurde ein Signalhandler vorgesehen, welcher auf das Signal SIGUSR1 hört. Die entsprechende Funktion ließt dann die Logikdaten erneut ein und überschreibt damit die vorherige Version im Speicher. Gleichzeitig wird hier auch die Konfigurationsdatei der Timer neu eingelesen. Damit muss die Steuerung nicht neu gestartet werden und ermöglicht es, das Steuerungsprogramm zur Laufzeit zu verändern. Das ist vor allem in Hinblick auf eine Integration in eine grafische Benutzeroberfläche deutlich komfortabler als den Systemdienst jedes Mal manuell neu starten zu müssen.

3.5 Globale Konfigurationsdatei

Im Laufe der Arbeit wurde die Anzahl der Konfigurationsparameter im Programm immer länger. Daher war ein Export der Konfiguration unumgänglich. Die Datei gliedert sich in Kontexte, wobei für jeden Kontext ein oder mehrere instanceKeys definiert werden können. Pro instanceKey wird im Programm eine Instanz angelegt. Somit könnten zum Beispiel durch ein zweiten instanceKey im Kontext Timer weitere acht Timer hinzugefügt werden. Dieser instanceKey findet sich dann als erster Buchstabe in den Bezeichnern, z.B. `H`. Das gleiche Vorgehen bietet sich für

die input-entity-keys und die output-entity-keys an. Hiermit können die Buchstaben für die Adressierung der Eingangs- bzw. Ausgangsentität festgelegt werden, z.B. `i` für Eingänge oder `o` für Ausgänge. Der Kontext `Memory` weicht dahingehend von diesem Standard ab, als dass hier duplexEntityKeys anstatt von input- oder output-EntityKeys zugewiesen werden. Weiterhin kann für jede Entität eine Zugriffsberechtigung (siehe Unterunterabschnitt 3.8.3) und ein private token definiert werden. Darüber hinaus gibt es kontexteigene Parameter. So ist im Kontext Hardware eine Eigenschaft `use_module` vorgesehen, mit welcher zukünftig das zu verwendende Hardware-Modul ausgewählt werden kann. Im Kontext `Network` kann festgelegt werden, auf welcher IP Adresse und auf welchem Port der WebSocket lauscht, und ob dieser überhaupt aktiv sein soll.

3.6 Hauptschleife und Iterationslogik

Das Logikprogramm liegt nun im Speicher und das Programm betritt die Hauptschleife. Doch treibt ein einfaches Polling die Auslastung des Prozessors unnötig nach oben. Nun könnte nach jedem Durchlauf eine gewisse Zeit gewartet werden, bevor eine neue Iteration beginnt. Doch das verringert die Reaktionszeit der Steuerung. Die beste Lösung findet sich direkt im Treiber des PiFaceDigital. Hier bietet sich eine Funktion, die bis zum Eintreten einer Veränderung der Eingangswerte blockiert. Intern eingesetzt wird hier ein Epoll, welcher einen File-Descriptor auf einen der GPIOs des Raspberry Pis überwacht und diesen somit als Interrupt-Kanal nutzt.

3.7 Logikprozessor

Den Kern des Projektes bildet der in Quellcode 3.5 gezeigte Logikprozessor. Es ist eine Methode, welche den in Unterabschnitt 3.4 geladenen Logikvektor zeilenweise durchläuft. Im ersten Schritt wird die Zeile nun auf Vorhandensein eines Gleichheitszeichens hin überprüft (Quellcode 3.5 Zeile 194). Eine Zeile ohne Gleichheitszeichen wird hierbei schlicht verworfen. Andernfalls wird zunächst der Teil vor dem Gleichheitszeichen näher untersucht. Dabei wird sichergestellt, dass der Bezeichner aus drei Zeichen besteht, wobei die ersten beiden alphabetisch und der

letzte numerisch sein muss. Diese drei Teile werden dann (Zeile 200-202) für die weitere Verarbeitung gespeichert. Der fertig ausgewertete Teil nach dem Gleichheitszeichen, auf dessen Verarbeitung nachfolgend noch etwas genauer eingegangen wird, und der nunmehr entweder `0` oder `1` ist, wird dem entsprechenden Kanal nun zugewiesen. Wie in Quellcode 3.5 Zeile 222 zu sehen, wird nun eine Instanz `chnl` der Klasse `IO_Channel_AccessWrapper` für den eigentlichen Zugriff auf die Ressource verwendet. In die eckigen Klammern werden dazu die Bezeichner eingetragen, die bereits in den Zeilen 200-202 ermittelt wurden. (Siehe Unterabschnitt 3.3*» Klassenstrukrur und Kanäle«*). Zudem wird dem Bezeichner in diesem Zuge sein Wert zugewiesen. Um diesen zu erhalten, muss nun die Gleichung gelöst werden. Diese beinhaltet ebenfalls Bezeichner, welche jeweils vom Funktor `replaceIdentifier` durch dessen aktuellen Wert ersetzt werden. Zurückgegeben wird ein String, welcher nun nur noch aus Binärzahlen, arithmetischen Operatoren (& für und bzw. | für oder) und Klammern besteht. Die eigentliche Lösung der Gleichung erfolgt dann mit einem Funktionsaufruf auf einen in Boost-Spirit geschriebenen Parser (siehe Unterabschnitt Unterabschnitt *»3.1 Aufbau und Parsing der Logikgleichungen«*).

```
186 void logicEngine(IO_Channel_AccesWrapper& chnl, std::vector<std::string>& softLogic){
187     bool             dbg           =  true;
188     std::string  delimiter   = "=";
189     size_t           found;
190 
191     // Iterates over Vercor row by row.
192     for (std::string softLogicRow: softLogic) {
193         // Splits the output string by the '=' sign in two parts, if found.
194         if ((found = softLogicRow.find('=')) != string::npos){
195             // The part before the '=' is the "output" to which the result will be asigned to
196             string asignedEntityStr         = softLogicRow.substr(0,found);
197             if(asignedEntityStr.size() != 5){
198                 throw std::invalid_argument(errMsg);
199             }
200             char    c_io_channel        = asignedEntityStr.at(1);
201             char    c_channel_entity    = asignedEntityStr.at(2);
202             int     c_pin_num           = asignedEntityStr.at(3) - '0';
203 
204             // String after '=' is equation including variables called identifiers (e.g. [Ho0] ),
205             // arithmetic operators ( &, | ) and round brakets.  Also allowed are literals (0 or 1)
206             string equationStringVariables  = softLogicRow.substr(found+1, string::npos);
207 
208             // Instantiate the replace-identifier Functor/Class thats used to
209             // inject the IO_Channel_AccessWrapper instance into the actual replace-function.
210             replaceIdentifier rplacIds(chnl);
211 
212             // Replaces every occurance of a Identifier by its current state. (e.g. [Hi0] => 1 / 0 )
213             string equationStringLiterals  = regex_replace(  equationStringVariables,
214                                                              regex("\\[([A-Z][a-z][0-8])\\]"),
215                                                              rplacIds
216                                                            );
217 
218             // Eventually the logic string will be evaluated (e.g. "!0 & 1 | 1" => 1)
219             bool logic_equation_res = evaluateLogicString(equationStringLiterals);
220 
221             // And is then assigned to the Identifier before the '='
222             chnl [c_io_channel] [c_channel_entity] -> write_pin(logic_equation_res, c_pin_num);
```

Quellcode 3.5: Logic engine

3.8 WebSocket-Server

3.8.1 Basis

Um die Steuerung mit der Außenwelt zu verbinden, wurde ein WebSocket-Server (siehe 2.6) in das Backend integriert. Als Basis hierfür diente das Beispiel, welches von Vinnie Falco im Rahmen eines Vortrages auf der `CppCon 2018` erstellt wurde. [19] Die Funktion dieses Beispiels ist das Bereitstellen eines WebSocket-Chatservers. Dieser baut auf einem rudimentären HTTP Server auf und ist deshalb in der

Lage, auch normale HTTP Anfragen zu beantworten. Auf diesem Wege erhält der Webbrowser des Clients die HTML Webseite, in welcher gleichzeitig auch der JavaScript Chat-Client enthalten ist, der letztendlich die direkte Verbindung zum WebSocket-Server aufbaut. Nachdem die Verbindung erfolgreich hergestellt wurde, kann der Benutzer Texteingaben tätigen und diese an den Server senden, welcher diese dann an alle verbunden Clients verteilt. Da es nun relativ einfach umsetzbar ist, anstatt Text auch andere Formate wie etwa JSON über diesen Server an die Clients zu senden, schien dieses Beispiel eine ideale Grundlage zu sein. So könnte anstatt Text ein JSON mit dem Status der Ein- und Ausgänge der Steuerung an die verbundenen Clients verteilt werden, welche diese dann mit einer JavaScript Anwendung so umsetzen, dass der Status der Ein- und Ausgänge visuell dargestellt wird. Zudem ist über diesen Kommunikationskanal auch ein Schreibzugriff denkbar. So könnte ein symbolisch dargestellter Schalter im Frontend beim Anklicken per JavaScript Funktion einen Befehl an das Backend senden, welches diesen dann ausführt.

3.8.2 Erweiterung

Das vorher erwähnte Beispiel wurde in einem ersten Schritt in einen Unterordner des Projektes kopiert und das CMake-File dahingehend geändert, das anstatt eines eigenen Executables eine Library gebaut wird, welche dann vom Grundprojekt verwendet wird. Das Unterprojekt wurde zudem als Abhängigkeit deklariert, sodass es beim Bauen des Hauptprojektes automatisch neu gebaut wird, sofern sich Änderungen im Unterprojekt ergeben haben. Die einfachste Erweiterung war es, die schon vorhandene `broadcast`-Funktion zu verwenden, um bei einer Änderung an den Ein- und Ausgängen eine Nachricht an alle verbundenen Clients zu senden. Da jedoch immer nur eine Nachricht gleichzeitig gesendet werden kann, wurde eine `Queue` verwendet, welche die Nachrichten sammelt, und diese dann nacheinander zustellt. Im nächsten Schritt wurden eingehende Nachrichten nun nicht mehr bloß eins zu eins an die `broadcast` Methode übergeben und damit an alle Clients weitergeleitet. Stattdessen mündeten diese nun in eine weitere `Queue`: *»Command-Queue«*. Eingehende Nachrichten sollen damit gesammelt werden, um die darin enthaltenen Befehle auszuführen.

3.8.3 Zugriffsbeschränkung

Um zu verhindern, dass jeder Client nun unkontrolliert vollen Zugriff auf alle Kanäle der Steuerung erhält, wurde daraufhin ein Konzept für die Zugriffsbeschränkung erarbeitet. Dafür wurden zwei Benutzerrollen eingeführt, `public` und `private`. Sie gelten für jeden Kanal und jeden Client individuell. Dafür besitzt jeder Kanal ein `private_token`. Authentifiziert sich ein Benutzer mittels `auth:<Kanal>:<Token>` für den Kanal, so wechselt er für diesen Kanal in die Rolle `private`. Da jeder Kanal aus mehreren Entitäten besteht, besitzt nun jede dieser Entitäten die Eigenschaften `expose_read` und `expose_write`. Zulässige Werte hierfür sind jeweils `public` `private` oder `none`. Bei Zugriff auf den Kanal bzw. die Entität wird nun darauf geprüft, ob der Benutzter **mindestens** die eingetragene Berechtigung hat. Nehmen wir zum Beispiel den Kanal Hardware *»H«*: Dessen Entitäten `i` für die Eingänge und `o` für die Ausgänge haben beide `expose_read` auf `private` gesetzt. Das `private_token` ist mit `top-secret-123` definiert. Ein verbundener Client müsste sich nun erst mit dem Befehl `auth:H:top-secret-123` für den Kanal authentifizieren, bevor er Statusinformationen über die Entitäten dieses Kanals erhält. Wäre `expose_read` auf `public` gesetzt, würde er diese Informationen ohne vorherige Authentifikation erhalten. Andersherum würde der Wert `none` bedeuten, dass niemand - nicht einmal ein authentifizierter Client - Statusinformationen zu dieser Entität erhält. Quellcode 3.6 stellt ein typisches Statusupdate der privaten Entitäten `Qo` und `Qi` dar, darauf folgt das Update aller Entitäten, die als `public` definiert wurden.

```
1 {"Qo" : 4, "Qi" : 1}
2 {"Po" : 170, "Pi" : 4, "Ho" : 0, "Hi" : 6}
```

Quellcode 3.6: JSON Statusupdate von privaten und öffentlichen Entitäten

3.9 Benutzeroberfläche

Die Benutzeroberfläche ist bei diesem Projekt in Form von einer Webseite umgesetzt, welche über einen Internetbrowser aufgerufen werden kann. Wie in Abbildung 3.4

zu sehen, dient hierfür die IP Adresse oder der Hostname der Steuerung solange man sich im selben Netzwerk befindet. Andernfalls muss der Internetrouter für einen ein externen zugriff eingerichtet werden.

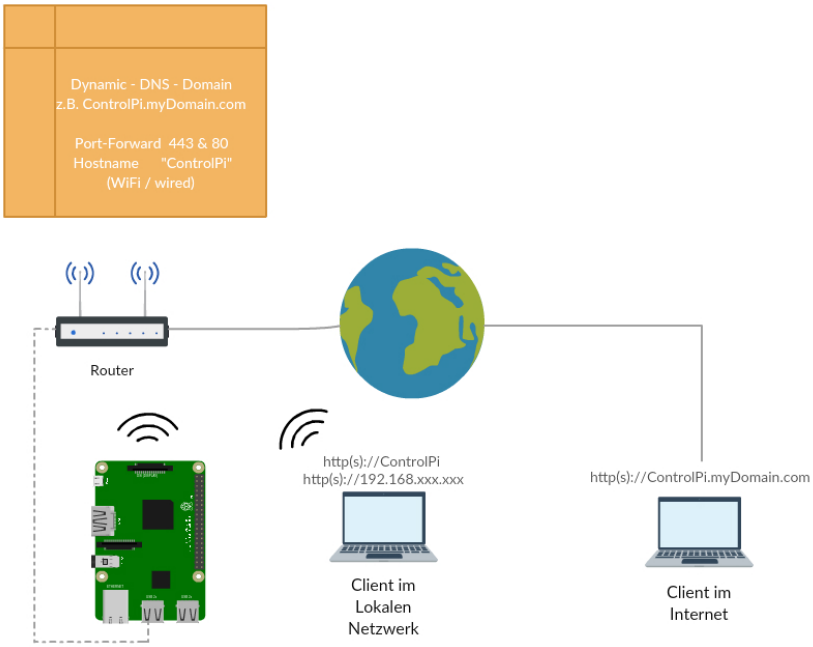

Abbildung 3.4: Zugriff auf die Weboberfläche

3.9.1 Übersicht Komponenten

Die eigentliche Anwendung ist hauptsächlich in HTML und JavaScript geschrieben und wird im Browser des Benutzers ausgeführt. Um die Benutzeroberfläche mit dem Rest der Anwendung kommunizieren zu lassen, bedarf es mehrerer Komponenten. Abbildung 3.5 zeigt diese und illustriert die Zusammenhänge. Obwohl der Hauptteil der Anwendung über JavaScript WebSockets eine direkte Verbindung vom Browser zum Backend herstellen könnte, wird ein Apache2 Webserver zwischengeschaltet. Dieser wird um das Proxy-Modul `modproxy` erweitert, was es ermöglicht, an das Backend gerichtete Anfragen mit dem WebSocket-Header auf die lokale loopback Adresse `127.0.0.1` weiterzuleiten, während alle sonstigen Anfragen vom Webserver direkt beantwortet werden können. Zum einen wird hierdurch die Konfiguration von

verschlüsselten Verbindungen über SSL und der Abruf der dazu nötigen Zertifikate erheblich vereinfacht, denn einige Zertifizierungsanbieter wie zum Beispiel Let's Encrypt[20] bieten extra auf Apache zugeschnittene Skripte, die ebendies fast vollständig automatisieren. Zum anderen ist es so möglich, die Auslieferung der HTML und JavaScript Quellen an den Webserver zu delegieren, anstatt bei einer eigenen Implementierung Sicherheitslücken und Performance-Probleme zu riskieren. Weiterhin ermöglicht es dieser Ansatz auch PHP Skripte zu verarbeiten, dies wird zum Beispiel eingesetzt, um Konfigurationsdateien vom Browser auf dem Server abzulegen, oder um dem Backend Signale zu senden, wenn es die Logikdatei erneut einlesen soll.

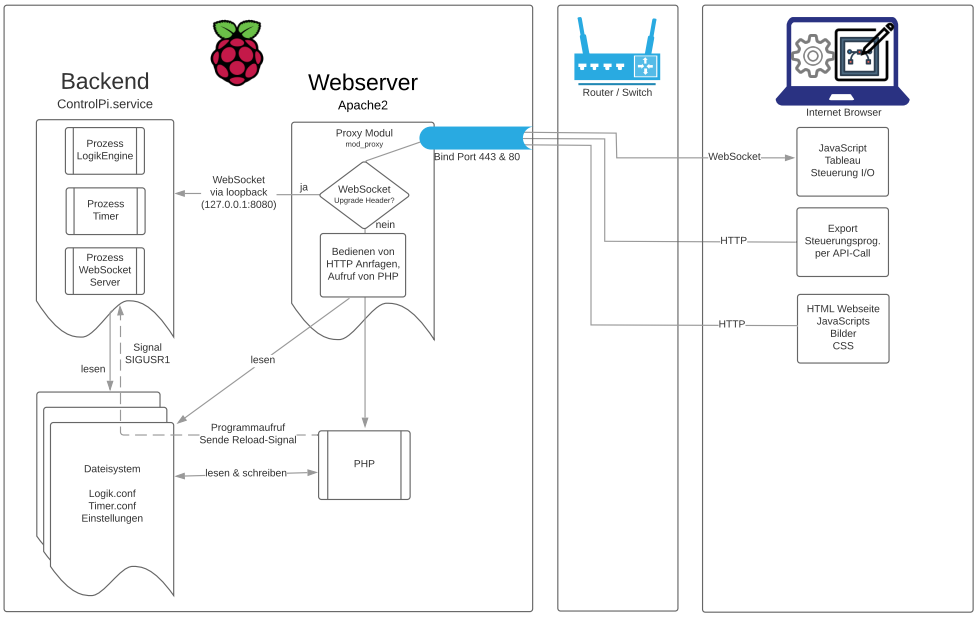

Abbildung 3.5: Networking Übersicht

3.9.2 JavaScript Anwendung

Um den Status der Steuerung auch dann sehen zu können, wenn man sich nicht in Sichtweite befindet, wurde hierzu eine Übersicht in die Benutzeroberfläche implementiert. Die in Abbildung 3.6 gezeigte Ansicht beinhaltet vier Spalten von Entitäten, wobei sich oberhalb der jeweiligen Spalte durch ein Drop-Down Menü

einstellen lässt, welche Entität angezeigt werden soll. Das Standardvorgehen ist, dass die erste Spalte die erste Entität anzeigt, die zweite die zweite und so weiter. Die JavaScript Anwendung baut gleich zu Anfang eine Verbindung zum Backend auf, die dann gehalten wird bis die Webseite wieder geschlossen wird. Sobald die Steuerung durch ein beliebiges Ereignis über das Logikprogramm iteriert, erhalten alle (autorisierten) WebSocket-Sitzungen ein Update aller Entitäten und stellen den Status der Ein- und Ausgänge grafisch dar. Dazu wird lediglich die angezeigte Grafik verändert. Es liegen dementsprechend jeweils eine Grafik für den Status `an` und eine für den Status `aus` vor.

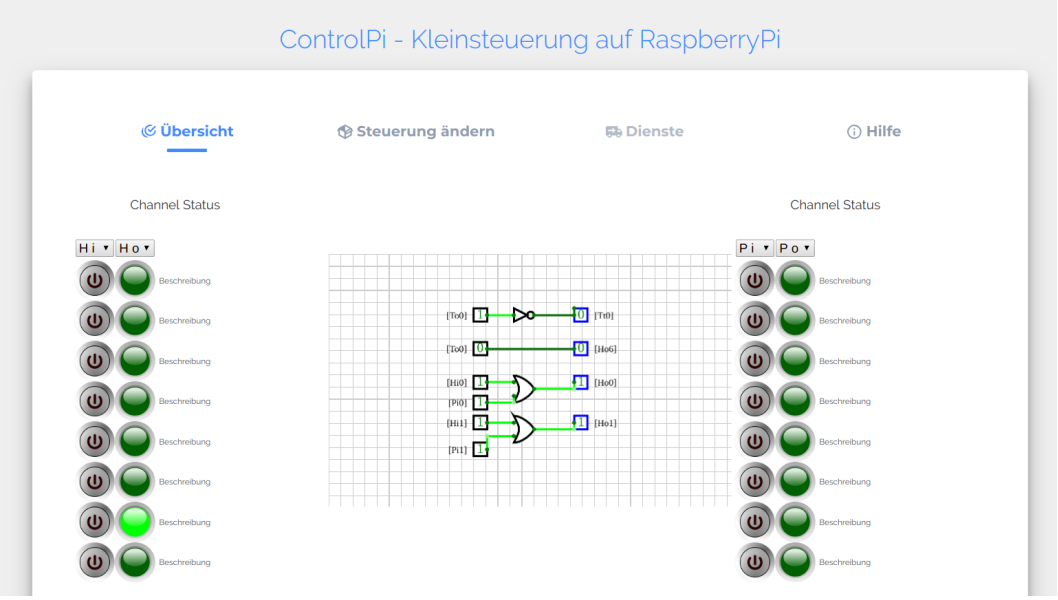

Abbildung 3.6: Statusinformationen der Steuerung

3.9.3 Logikeditor CircuitVerse

Die Wahl für einen graphischen Logikeditor fiel auf den mit der MIT Lizenz gekennzeichneten Editor CircuitVerse.[8] Die Lizenzierung gestattet es, das Programm zu verändern und weiter zu verbreiten. Der in Abbildung 3.7 gezeigte Screenshot zeigt den Aufbau des Editors. Im linken Abschnitt kann dabei zwischen verschiedenen Bauteilen gewählt werden, welche dann per Drag & Drop auf die rechts daneben befindliche Zeichenfläche gezogen werden können. Grüne Punkte kennzeichnen da-

bei die Anschlussknoten, welche dann wiederum durch Ziehen mit gehaltener linker Maustaste miteinander verbunden werden können. CircuitVerse bietet überdies die Möglichkeit, die erstellte Zeichnung in einem eigenen Format zu speichern sowie sie als Grafik zu exportieren. Des Weiteren gibt es die Möglichkeit, eine Zeichnung anhand einer Logiktabelle automatisch erstellen zu lassen.

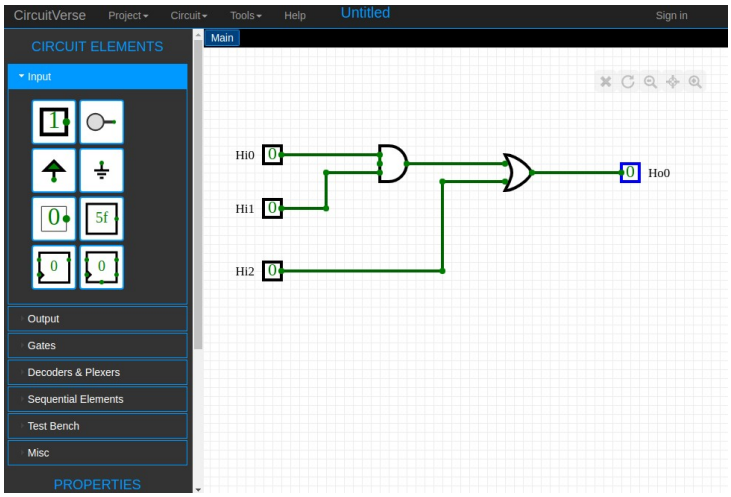

Abbildung 3.7: Screenshot einer Logikschaltung in CircuitVerse

Erweiterung für den Export

Eine Erweiterung von CircuitVerse ist nötig, um die gezeichnete Logikschaltung in ein Format umwandeln zu können, welches vom Backend verstanden wird. Weiterhin muss das Benamsungsschema (siehe Unterabschnitt *»3.2 Benamungsschnema«*) eingebracht werden. Letztlich muss auch die Bauteile-Auswahl auf jene Bauteile reduziert werden, welche auch tatsächlich in der Steuerung verfügbar sind. Unterabschnitt *»3.4 Laden der Programmlogik aus Datei«* beschreibt das Format der Datei, die letztlich aus einer gezeichneten Logikschaltung gewonnen werden soll. CircuitVerse verwendet für die Speicherung der Schaltung eine JSON Struktur. Dabei wird in Bauteile und Knoten unterschieden. Ein Bauteil hat einen oder mehrere Knoten, wobei Knoten miteinander verbunden werden können und dann jeweils Informationen über angeschlossene Knoten enthalten. Abbildung 3.9 verdeutlicht, wie CircuitVerse die Knoten verwendet. Grundlage für die Abbildung ist ein Grafik-

export von CircuitVerse, wobei die Knotennummern zum besseren Verständnis der Funktionsweise besonders gekennzeichnet wurden. In *» Abbildung 3.8 Darstellung der JSON Datenstruktur«* ist die zugehörige JSON Datenstruktur abgebildet.


```
▼{name: "ControlPi", timePeriod: 500, clockEnabled: true, projectId: "dIZeQVPjuofBL5XFCmXS",…}
   clockEnabled: true
   focussedCircuit: 28983776459
   name: "ControlPi"
   projectId: "dIZeQVPjuofBL5XFCmXS"
  ▼scopes: [{layout: {width: 100, height: 80, title_x: 50, title_y: 13, titleEnabled: true},…}]
   ▼0: {layout: {width: 100, height: 80, title_x: 50, title_y: 13, titleEnabled: true},…}
     ▶AndGate: [{x: 320, y: 110, objectType: "AndGate", label: "", direction: "RIGHT", labelDirection: "LEFT",…}]
     ▶Input: [{x: 140, y: 110, objectType: "Input", label: "[Hi0]", direction: "RIGHT", labelDirection: "LEFT",…},…]
     ▶NotGate: [{x: 230, y: 150, objectType: "NotGate", label: "", direction: "RIGHT", labelDirection: "LEFT",…}]
     ▶Output: [,…]
     ▼allNodes: [{x: 10, y: 0, type: 1, bitWidth: 1, label: "", connections: [5]},…]
       ▶0: {x: 10, y: 0, type: 1, bitWidth: 1, label: "", connections: [5]}
       ▶1: {x: 10, y: 0, type: 0, bitWidth: 1, label: "", connections: [10]}
       ▶2: {x: -10, y: -10, type: 0, bitWidth: 1, label: "", connections: [5]}
       ▶3: {x: -10, y: 10, type: 0, bitWidth: 1, label: "", connections: [6]}
       ▶4: {x: 20, y: 0, type: 1, bitWidth: 1, label: "", connections: [8]}
       ▶5: {x: 150, y: 100, type: 2, bitWidth: 1, label: "", connections: [0, 2]}
       ▶6: {x: 310, y: 150, type: 2, bitWidth: 1, label: "", connections: [3, 13]}
       ▶7: {x: 10, y: 0, type: 1, bitWidth: 1, label: "", connections: [12]}
       ▶8: {x: 10, y: 0, type: 0, bitWidth: 1, label: "", connections: [4, 11]}
       ▶9: {x: 10, y: 0, type: 1, bitWidth: 1, label: "", connections: [10]}
       ▶10: {x: 440, y: 250, type: 2, bitWidth: 1, label: "", connections: [1, 9]}
       ▶11: {x: 430, y: 120, type: 2, bitWidth: 1, label: "", connections: [8]}
       ▶12: {x: -10, y: 0, type: 0, bitWidth: 1, label: "", connections: [7]}
       ▶13: {x: 20, y: 0, type: 1, bitWidth: 1, label: "", connections: [6]}
      id: 28983776459
     ▶layout: {width: 100, height: 80, title_x: 50, title_y: 13, titleEnabled: true}
      name: "Main"
     ▶nodes: [5, 6, 10, 11]
   timePeriod: 500
```


Abbildung 3.8: Darstellung der JSON Datenstruktur

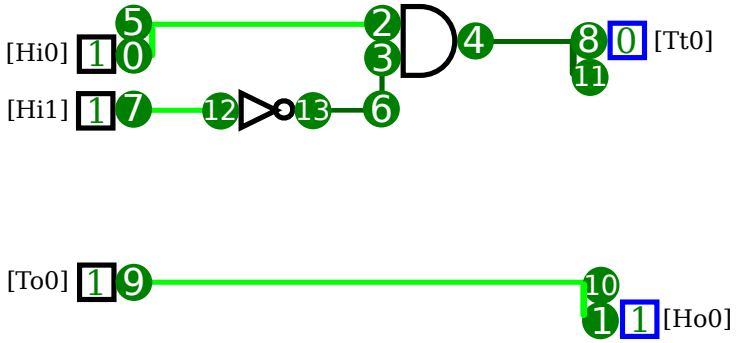

Abbildung 3.9: Logikschaltung mit gekennzeichneten Knoten

In der vorangehenden Abbildung 3.9 ist zu sehen, dass jedes Bauteil einen oder mehrere Knoten besitzt. In Abbildung 3.8 ist zu sehen, dass jeder dieser Knoten eine eigene Zeile unterhalb von `allNodes` besitzt. Hieraus geht hervor, mit welchen

anderen Knoten er verbunden ist. Außerdem verbirgt sich im Feld `Typ` die Information, ob es sich um einen Ausgangsknoten, einen Eingangsknoten oder einen Verbindungsknoten handelt. Ist der Knoten vom Typ Eingangknoten oder vom Typ Ausgangsknoten, so gehört er einem Bauteil wie einem Logikgatter, einem Eingangsbausteinen oder einem Ausgangsbausteinen. Die Anordnung der Knoten ist logischerweise andersherum wie die der Bauteile, so besitzt ein Ausgangsbaustein genau einen Eingangsknoten (z.B. 3.9 Knoten 1) sowie ein Eingangsbaustein genau einen Ausgangsknoten (z.B. Abbildung 3.9 Knoten 9). Ein Gatter hingegen hat einen Ausgangsknoten sowie einen oder mehrere Eingangsknoten (z.B. Abbildung 3.9 Eingangsknoten 3,4 und Ausgangsknoten 4). Für die Umwandlung in Textform schien es am Sinnvollsten, an einem Ausgang anzusetzen und eine Wegfindung durchzuführen, welche beim Erreichen eines Eingangsbausteins einen Endpunkt hat. Dazu durchläuft der in Abbildung 3.11 gezeigte rekursive Algorithmus den Weg von einem Ausgangsknoten bis zu einem Eingangsknoten. Ist ein Eingangsknoten erreicht, so wird geprüft, ob es sich um den Ausgang eines Eingangsbausteins handelt oder um den Ausgang eines Logikgatters. Da CircuitVerse die Knoten jedoch nicht nach der Art des zugehörigen Bauteils, sondern nur in Eingangsknoten, Ausgangsknoten oder Leitungsknoten unterteilt, muss die in 3.8 gezeigte Struktur oberhalb der Eigenschaft `allNodes` vorab durchlaufen werden. Hierbei wird ein Lookup-Array angelegt (siehe Abbildung 3.10), welches die Bauteile mit der Knotennummer ihres Ausgangs indiziert. Bezogen auf das in Abbildung 3.9 gezeigte Beispiel, würde für das AND-Gate ein Eintrag mit dem Index 4 angelegt. Abbildung 3.10 zeigt den Eintrag für dieses AND-Gate in der aufgeklappten Zeile mit dem Index `4`. Das Array `nodesIn` beinhaltet die Knoten 2 und 3. An diesem Punkt ruft die Funktion zur Wegfindung sich selbst auf. Auf diese Weise werden sämtliche Logikgatter durchlaufen. In diesem Beispiel (siehe Abbildung 3.9) würde die Wegfindung von Knoten 2 über Knoten 5 auf Knoten 0 treffen. Dieser ist jedoch kein Gateknoten sondern der Knoten eines Eingangsbausteines: Ein Endpunkt ist erreicht, das heißt, es wird nun die Zeichenkette `[Hi0]` zurückgegeben. Zurück im vorher besprochenen AND-Gate, wird diese Zeichenkette nun mit dem Rückgabewert des Stranges an Knoten 3 und einem kaufmännischen Und (&) verkettet. Da das AND-Gate genau zwei Eingangsknoten hat, sind nunmehr alle Knoten

abgearbeitet und der Baustein gibt die Zeichenkette `[Hi0] & ! [Hi1]` zurück. Im Eingangsknoten wird diese Zeichenkette nun um den Bezeichner ergänzt. Somit ergibt sich `[Tt0] = [Hi0] & ! [Hi1]` als Gesamtergebnis.

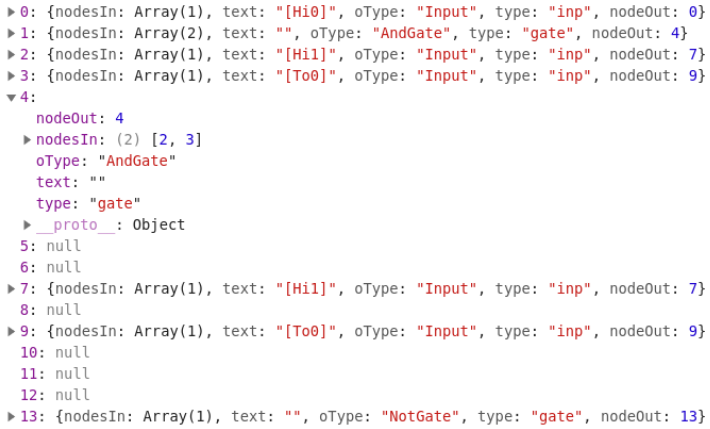

Abbildung 3.10: Lookup Array

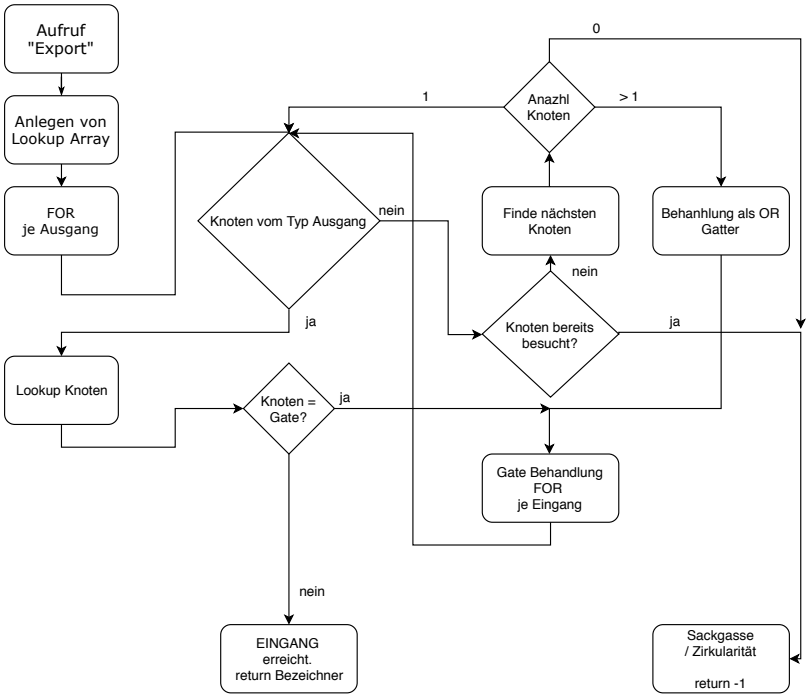

Abbildung 3.11: Programm Ablaufplan

Anpassung der Eingangsbausteine

Wie in *»3.9.3 Erweiterung für den Export«* besprochen, wurde CircuitVerse um einen Algorithmus zum Export der gezeichneten Schaltung erweitert. Dieser wird über einen neuen Menüpunkt (siehe 3.12) gestartet und zeigt dann wie in Abbildung 3.13 zu sehen das Exportergebnis zur Überprüfung an. Nach der Bestätigung des Ergebnisses wird dieses per Ajax Aufruf an ein in Unterunterabschnitt 3.9.4 näher beschriebenes PHP Skript übergeben, welches dieses dann als Textdatei speichert. Der Menüpunkt `Save Online` schreibt das bisherig verwendete JSON Format über die selbe Schnittstelle in eine weitere Textdatei auf dem Server. Dabei wird zudem eine Abbildung der Schaltung erzeugt und ebenfalls auf den Server übertragen. Letztlich dient die Schnittstelle auch dem lesenden Zugriff auf die JSON Datei.

So wird beim ersten Aufruf des Logikeditors die JSON Datei vom Server an die JavaScript Anwendung CircuitVerse übermittelt. Damit kann die bereits vorhandene Steuerung erweitert oder verändert werden. Um zu vermeiden, dass sich die Logik in der JSON Datei von der Logik in der vom Backend lesbaren Logikdatei unterscheidet, soll die Speicherung in einem Menüpunkt zusammengeführt werden. Dies dient vor allem auch dem einfacheren Verständnis durch den Endbenutzer. Es schränkt jedoch die Möglichkeit ein, neben der Steuerung, die gerade in Betrieb ist, weitere alternative Steuerungsprogramme abzuspeichern. Da der Logikeditor in ein helles Template eingebettet werden soll, wurden außerdem das Farbschema und die Schriftart modifiziert.

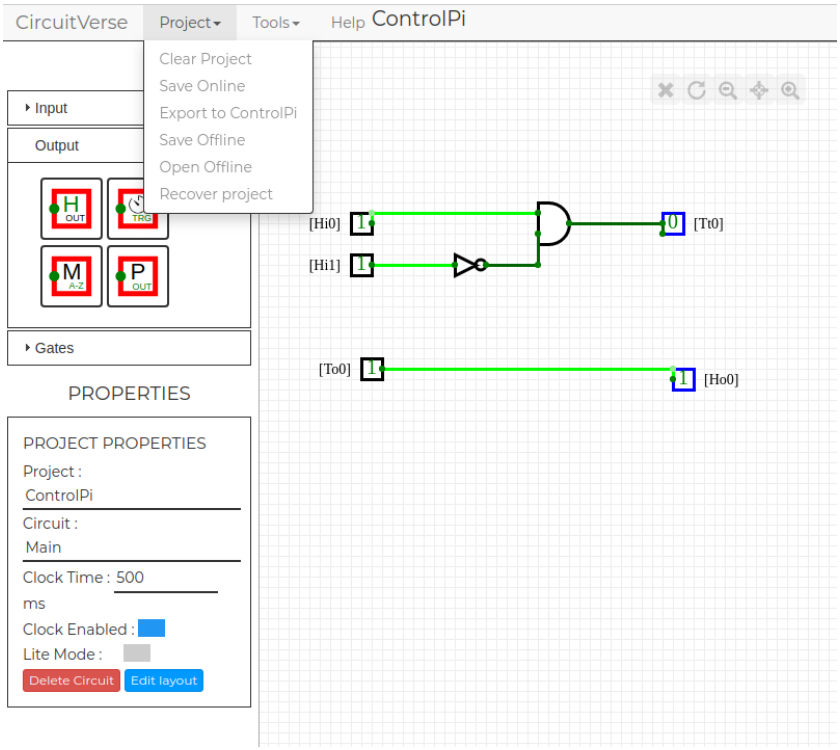

Abbildung 3.12: Umgestatletes CircuitVerse

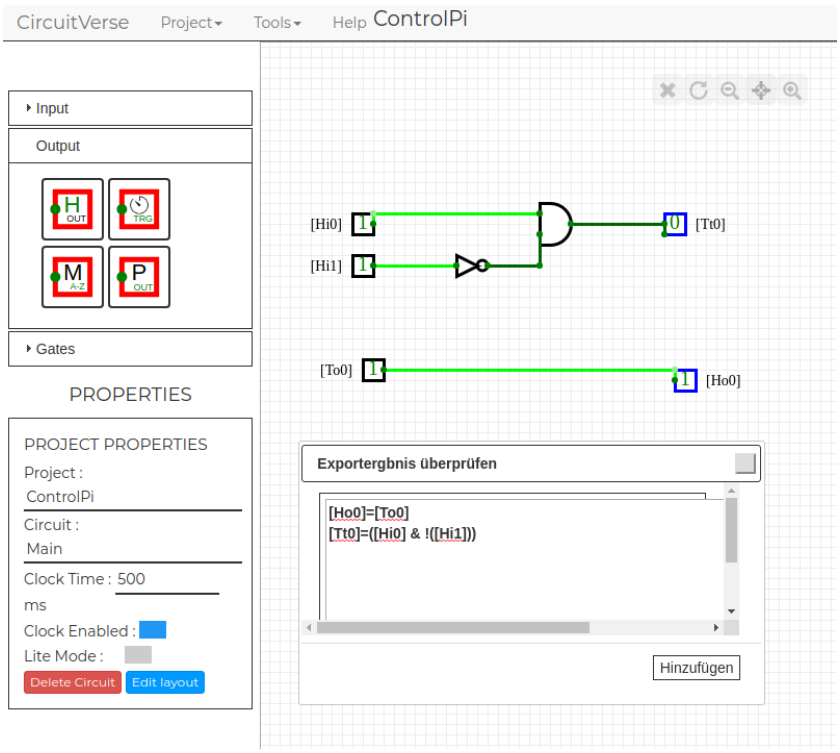

Abbildung 3.13: Dialog mit Exportergebnis

3.9.4 PHP Adapterprogramme

Schnittstelle zum Speichern als Datei

Da CircuitVerse als JavaScript Anwendung im Browser ausgeführt wird und somit keinen direkten Zugriff auf das Dateisystem des Servers hat, ruft es beim Speichern oder beim Laden per Ajax eine URL auf. Um Dateien nun persistent in das Dateisystem des Servers ablegen zu können, wird eine API Schnittstelle benötigt. Dies wurde durch ein rudimentäres PHP Skript `logicUpdateApi` realisiert. Auch ein lesender Zugriff ist somit möglich, dies geschieht zum Beispiel wenn der Logikeditor die bereits abgespeicherte Version des Steuerungsprogramms anzeigen soll. Zwischen lesendem und schreibendem Zugriff wird dabei über die verwendete Zugriffsmethode unterschieden, wobei `POST` einen schreibenden Zugriff gewährt und `GET` den Inhalt der Datei liest.

WebSocket-Adapter

Das im letzten Abschnitt 3.9.4 beschriebene Verfahren zum Ablegen von Konfigurationsdaten ist ein einmaliges Ereignis, dementsprechend wird hier auch eine normale HTTP Anfrage eingesetzt, welche nach vollständiger Übertragung abgebaut wird. Im Unterschied dazu wird für die Ansteuerung der virtuellen Kanäle sowie für die Statusabfrage der physikalischen Ein- und Ausgänge eine persistente Verbindung aufgebaut. Dabei baut der Client eine Verbindung zu dem WebSocket-Server auf, welcher in Kapitel 3.8 näher beschrieben wurde. Um einmalige Ereignisse von externen Diensten per HTTP zu ermöglichen, baut ein PHP Skript nun on-demand eine WebSocket-Verbindung zu dem im Backend 2.2.2 integrierten Server auf, führt den gewünschten Befehl aus und baut die Verbindung daraufhin wieder ab.

3.9.5 IFTTT

Um eine Anbindung an IFTTT zu ermöglichen, bieten sich Webhooks an. Hierbei wird beim Auslösen eines Ereignisses eine vorher hinterlegte URL aufgerufen. Im Falle dieses Projekts wird als Ziel hierfür die URL eines in Abschnitt 3.9.4 näher beschrieben PHP Skripts verwendet, welches dann kurzzeitig eine WebSocket-Verbindung zum Backend herstellt und diese nach der vollständigen Übermittlung der Daten wieder abbaut. Dabei dient ein Array in der Parameterliste dazu, beliebig viele Befehle mitzugeben, welche dann sequentiell über den WebSocket-Kanal gesendet werden. Ein Beispiel wäre, dass nach dem Herstellen der Verbindung durch die Übermittlung des Authentifizierungs-Token der Schreibzugriff für eine Ressource erworben wird, um durch einen weiteren Befehl schreibend auf die Ressource zuzugreifen. Als Auslöser kann ein beliebiger anderer IFTTT Service dienen. Im einfachsten Fall wäre dies ein Button für den Homescreen eines Smartphones.

4 Übersicht Gesamtsystem

4.1 Inbetriebnahme

4.1.1 Hardware Voraussetzungen

Voraussetzung für den Betrieb ist ein Raspberry Pi 2 oder 3. Außerdem ist eine Erweiterungsplatine *»PiFace Digital2«*[12] erforderlich. Diese muss vor dem ersten Start auf die GPIO-Kontakte des Raspberry Pis aufgesteckt werden. Eine Funktion ohne das Hardwaremodul ist grundsätzlich möglich, jedoch scheint ein Betrieb ohne Hardwareschnittstelle ohne Sinn. Die Software ist so konzipiert, dass auch andere Hardware-Module denkbar sind, eine Implementierung hierfür fehlt jedoch bislang.

4.1.2 Software Voraussetzungen

Für den Betrieb wird ein Linux Betriebssystem vorausgesetzt. Hierfür kann das angepasstes Raspbian[21] Image genutzt werden, welches auf der beigelegten DVD zu finden ist. Falls auf der SD Karte bereits ein Linux Betriebssystem installiert ist, kann das Arbeitsverzeichnis des Projekts auch einfach auf den Raspberry Pi übertragen werden. Ein Git Auszug hiervon ist ebenfalls auf der beigelegten DVD. Sollte die Wahl der Distribution auf eine andere als Raspbian Stretch[21] fallen, so muss das Projekt aus den Quellen übersetzt werden. In jedem Fall ist es nötig, dass SPI aktiviert wird. Dies geschieht am Einfachsten über das Raspberry Pi Konfigurationsprogramm `sudo raspi-config`.[22]

4.1.3 Kompilieren des Projekts

Ein Kompilieren des Projekts ist nur erforderlich, wenn das Projekt auf einem anderen Betriebssystem als Raspbian Stretch[21] verwendet werden soll, oder wenn Änderungen am Quellcode vorgenommen werden sollen. Dafür muss zunächst eine SSH Verbindung zum System hergestellt werden. Das Projekt sollte daraufhin auf das System übertragen werden. Die empfohlene Vorgehensweise hierfür ist, das GIT Repository über eine aktive Internetverbindung auf den Rasperry Pi

zu klonen `git clone https://github.com/dajuly20/ControlPi`. Um dann alle Abhängigkeiten automatisch zu installieren, kann das nachfolgend genannte Installationsskript verwendet werden: `./start_pull_and_build.sh`. Sollte keine Internetverbindung bestehen, können die auf der DVD mitgelieferten Bibliotheken auch manuell installiert werden. Eine Liste der benötigten Abhängigkeiten findet sich im Anhang in Abschnitt 6.

4.1.4 Inbetriebnahme unter Raspbian Stretch

Sollte auf dem Raspberry Pi bereits eine Version von Raspbian in der Version Stretch installiert sein, so genügt es, den Projektordner auf den Raspberry Pi zu übertragen. Hierfür dient entweder der Ordner `ControlPi` auf der beigelegten DVD oder das GIT Repository des Projekts, welches mittels `git clone https://github.com/dajuly20/ControlPi` auf den Raspberry Pi übertragen werden kann. Um sicherzustellen, dass es sich um die aktuellste Version handelt, sollte das Projekt mit `git pull` auf den neusten Stand gebracht werden. Daraufhin kann mit `./start_manual.sh` die Funktion überprüft werden. Dabei sollte der Benutzer in den Gruppen `spi` sowie `gpio` sein. Letztlich sollte das Projekt als Systemservice eingerichtet werden. Hierfür ist das Skript `./start_as_service.sh` dienlich. Es wird hier neben dem eigenen Systemservice auch ein Apache2 Webserver mit in Betrieb genommen.

4.2 Inbetriebnahme mit Systemabbild

Auch befindet sich ein vollständiges Systemabbild[23] auf der beigelegten DVD. Dieses sollte zunächst mit `tar -xvzf ControlPi-Raspbian-Strech.tar.gz` entpackt werden. Die enthaltene `.img` Datei belegt ca. 7,3 GB. Eine Speicherkarte sollte also mindestens 8 GB, besser jedoch 16 GB Platz bieten. Das Systemabbild kann mit `sudo dd if=ControlPi-Raspbian-Strech.img of=/dev/<XXX> bs=4M status=progress` oder unter Windows mit einem entsprechenden Tool[24] auf eine Speicherkarte übertragen werden. Nach erfolgreichem Start sollte eine der LEDs auf der PiFace Erweiterungskarte dauerhaft blinken.

4.2.1 Ermittlung der IP Adresse

Nachdem der Systemservice läuft, muss nun ermittelt werden, wie ein Zugriff auf die Weboberfläche erfolgen kann. Wenn der Raspberry Pi in ein bestehendes Netzwerk integriert wird, kann dies in der Oberfläche des verwendeten Internetrouters nachgesehen werden. Sofern dieser dies unterstützt, kann auch der Hostname des Raspberry Pis für den Zugriff verwendet werden. Der Hostname im mitgelieferten Raspbian Image lautet `ControlPi3`. Der Standard bei einem offiziellen Raspbian Image ist `raspberrypi`. Sollte kein Zugriff auf den Internetrouter bestehen oder eine Ermittlung der IP Adresse aus sonstigen Gründen nicht möglich sein, kann ein Portscan durchgeführt werden. Die Steuerung öffnet in der Standardkonfiguration die Ports 80 für HTTP, den Port 443 für HTTPS bzw. SSL und den Port 22 für SSH Zugriffe. Angenommen die eigene IP Adresse (Ermittlung durch `ifconfig`) wäre 192.168.8.5 mit der Subnetzmaske 255.255.255.0, so wäre der Aufruf `nmap -p 22,80,443 192.168.8.0/24`. Sollten mehrere gefundene Geräte diese Ports als offen anzeigen, so sollten die IP Adressen dieser Geräte nacheinander im Webbrowser als URL eingetragen werden, um zu überprüfen, bei welcher es sich um die richtige handelt.

4.3 Funktionstest

4.3.1 Testen der Betriebsbereitschaft

Im Auslieferungszustand ist bereits ein zu Testzwecken erstelltes Steuerungsprogramm enthalten. Dieses lässt die LED an Ausgang 6 durch einen sich selbst aufrufenden Timer blinken. Zudem ist in diesem Testprogramm den Ausgängen `0` sowie `1` der Wert der Eingänge `0` beziehungsweise `1` zugewiesen. Ein Drücken der in Abbildung 4.1 gezeigten Taster S0 oder S1 sollte also ein Leuchten der entsprechenden LED zur Folge haben. Wie auch bei den Tastern werden die LEDs von rechts nach links durchnummeriert und sind den jeweiligen Ausgängen örtlich zugeordnet. Da an den Ausgängen `0` und `1` zudem Relais angeschaltet sind, sollte sich ein Zustandswechsel auf einem dieser Ausgänge außerdem in Form eines Klickgeräusches akustisch bemerkbar machen.

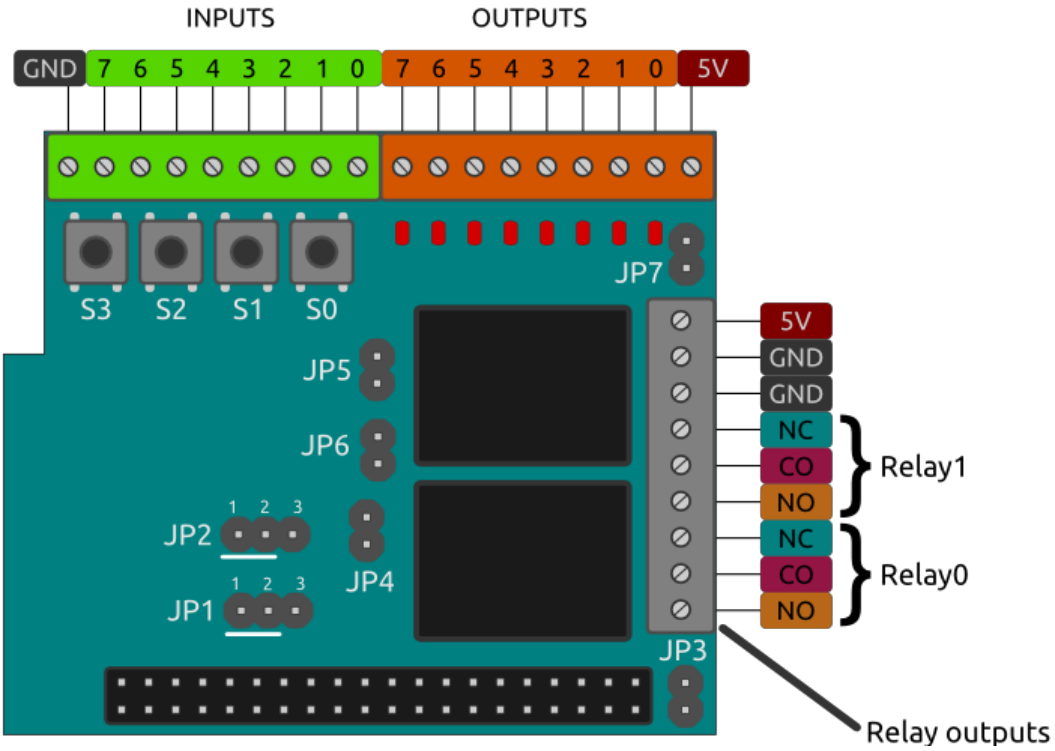

Abbildung 4.1: Darstellung eines PiFace Digital 2[25]

4.3.2 Aufruf der Weboberfläche

Ist der Hostname oder die IP Adresse des Raspberry Pis ermittelt (siehe Abschnitt 4.2.1 Ermittlung der IP Adresse), kann dieser im Internetbrowser aufgerufen werden. Bei Verwendung des beigelegten Raspbian Images wäre der Aufruf `http://ControlPi3`. Abbildung 4.2 zeigt die Übersichtsseite der Steuerung. In der Mitte befindet sich eine Abbildung vom aktuell verwendeten Steuerungsprogramm, während an den Seiten die Zustände der Ein- und Ausgänge angezeigt werden. Die Drop-Down Menüs oberhalb ermöglichen es, die gewünschte Ein- bzw. Ausgangsentität auszuwählen. Handelt es sich bei der ausgewählten Entität um einen Eingang, so wird anstelle einer LED ein Taste eingeblendet. Der Zustand kann dann, bei entsprechender Berechtigung (siehe Unterabschnitt *»3.8.3 Zugriffsbeschränkung«*), durch Anklicken verändert werden.

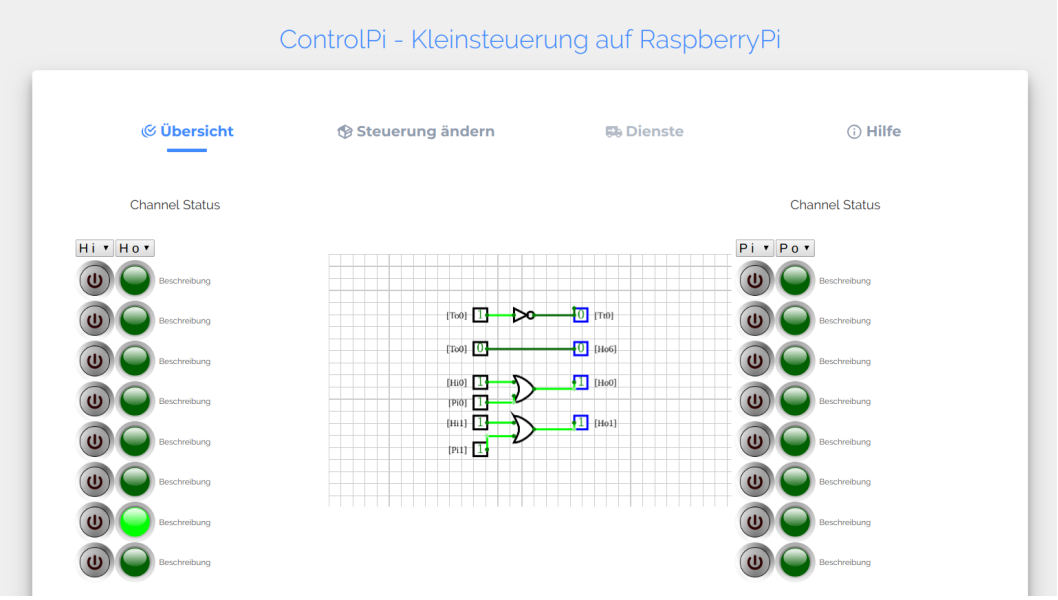

Abbildung 4.2: Darstellung Frontend Übersicht

4.3.3 Erstellen eines Steuerungsprogramms

Um ein Steuerungsprogramm zu erstellen, muss zunächst der Tab `Steuerung ändern` angewählt werden. Nun erscheint der in Abbildung 4.3 abgebildete Logik-Editor CicuitVerse.[8] Sollte die vorhandene Logikschaltung nicht sofort sichtbar sein, muss die Zeichenfläche zuerst durch Klicken auf das Fadenkreuz-Symbol in den Mittelpunkt gerückt werden. Soll nun eine komplett neue Schaltung gezeichnet werden, kann die Zeichenfläche durch den Unterpunkt `Clear Project` im Drop-Down Menü `Project` bereinigt werden. Alle verfügbaren Bauteile befinden sich im Bauteil-Menü auf der linken Seite. Dieses ist in die Kategorien `Input`, `Output` und `Gates` unterteilt, welche sich durch Anklicken aufklappen lassen. Möchte man ein Bauteil auf die Zeichenfläche schieben, so muss dieses angeklickt und der Cursor auf die Zeichenfläche bewegt werden. Ein erneutes Klicken lässt das Bauteil an der gewählten Position fallen. Die Bauteile werden schließlich mittels Drag & Drop miteinander verbunden. Werden Timer Bausteine angeklickt, so lassen sich deren Einschalt- und Ausschaltverzögerungszeiten im Menü `Properties` einstellen. Es gilt zu beachten, dass Ausgänge jeweils nur einmal vorkommen

dürfen. Gibt es mehrere Möglichkeiten, in denen ein Ausgang aktiv werden soll, empfiehlt sich die Verwendung eines Und- bzw. Oder-Gatters. Nach Vollendung des Steuerungsprogramms kann die Steuerung mit dem Menüpunkt `Save & Export`, welcher sich im Menü `Project` befindet, übernommen und getestet werden. Eine Überprüfung auf Richtigkeit des Steuerungsprogramms erfolgt nicht. Deshalb sollte der Infokasten `Backend Status` im Auge behalten werden. Falls ein Fehler mit dem Steuerungsprogramm auftritt, wird dieser dort angezeigt.

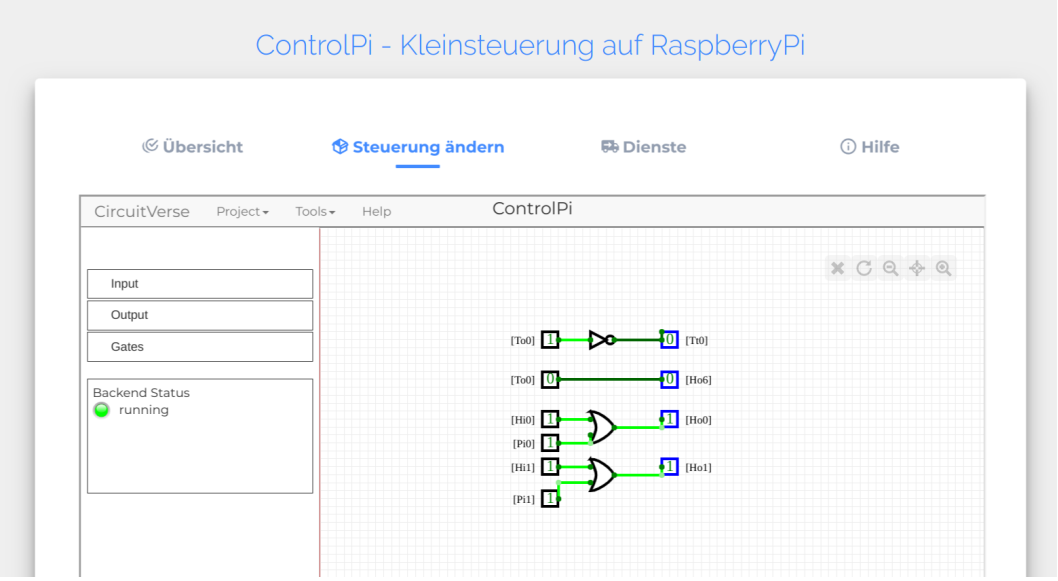

Abbildung 4.3: Darstellung Logikeditor

4.4 Langzeittest

Um zu testen, wie sich die Steuerung längerfristig verhält, wurde ein Timer Baustein verwendet, welcher sich nach Ablauf selbst schaltet und somit oszilliert. Als Testzeitraum wurden 48 Stunden gewählt. Dabei ließ sich feststellen, dass die Steuerung auch nach Ablauf dieser Zeit ohne Fehler weiterhin blinkte. Eine Recherche nach Neustarts des Systemdienstes blieb ebenfalls ohne Treffer.

4.5 Bugs

Zum derzeitigen Stand sind noch Probleme enthalten, auf die nachfolgend eingegangen wird. Zunächst ist hier zu nennen, dass die Reihenfolge, in welcher das Backend über die Zeilen des erzeugten Steuerungsprogramms iteriert, unter bestimmten Voraussetzung Probleme bereitet. So wird der Wert eines Ausgangs unter noch nicht näher bekannten Umständen erst mit der nächsten Iteration des Logikprogramms übernommen. Dies könnte durch generelles doppeltes Iterieren behoben werden, was allerdings die Performance vermindert. Der Logikeditor in der Benutzeroberfläche hat ebenfalls ein Darstellungsproblem. Beim Aufruf wird die bereits vorhandene Logikschaltung nicht automatisch zentriert, sondern ist beinahe unsichtbar in der oberen linken Ecke der Zeichenfläche. Ein Klicken auf das Fadenkreuz-Symbol bringt hier Abhilfe.

4.6 Fazit

Das Projekt teilte sich im Wesentlichen in zwei Teile auf. Das Backend ist einen Teil, welcher sich auf einer sehr hardwarenahen Ebene abspielte. Die Benutzeroberfläche hingegen ist sehr nahe am Benutzer. Damit spiegelte dieses Projekt viele Nuancen der Technischen Informatik wieder. Die meiste Arbeit entfiel dabei auf die Programmierung des Backends, das in mehrere parallele Handlungsstränge aufgeteilt ist, welche synchronisiert werden wollen. Auch das Arbeiten mit C++ Bibliotheken und das Kompilieren brachte einige Herausforderungen mit sich. Die Anbindung der Benutzeroberfläche beleuchtete eine relativ junge Technologie der modernen Webentwicklung. Auch das Anpassen des Logikeditors und die damit verbundene Einarbeitung in eine fremde Open-Source Software machten dieses Projekt sehr lehrreich. Zuletzt bot sich auch die Gelegenheit zur Erprobung, wie eine Software durch Systemabbilder oder Git verteilt werden kann und wie man diese mit LaTeX so dokumentiert, dass möglichst alle Aspekte beleuchtet werden und die Arbeit unkompliziert weitergeführt werden kann.

4.7 Ausblick

Die Steuerung funktioniert in den meisten Fällen ohne Probleme, ist jedoch noch nicht über ein frühes Betastadium hinaus. Es bedarf nun Feldtests, um weitere Bugs in der Software aufzuspüren. Außerdem sind noch einige Funktionserweiterungen denkbar. Zuerst anzuführen wäre hier die Kopplung zweier Geräte. Die Grundvoraussetzungen hierfür sind durch die Implementierung von WebSockets bereits geschaffen. Entweder müsste hier eine Mittelschicht geschrieben werden, welche die zwei WebSocket-Server auf den beiden Geräten miteinander kommunizieren lässt. Die elegantere Lösung jedoch wäre, die Implementierung eines WebSocket-Clients in das Backend. Mit einer angepassten Konfigurationsdatei könnte das Backend sich so zu einem anderen Gerät verbinden, um dessen virtuelle Ein- und Ausgänge auch am lokalen Gerät nutzbar zu machen. WebSockets haben durch die persistente Verbindung ein sehr gutes Verhältnis zwischen Overhead und Payload und sollten somit eine recht geringe zeitliche Verzögerung ermöglichen. Die Erfahrung von der Verbindung zwischen Web-Frontend und -Backend zeigen kaum merkliche Verzögerungszeiten. Eine weitere Funktion, über welche nachgedacht wurde, ist die Anlage von Unterstromkreisen. Somit könnten zum Beispiel Blinker Bausteine oder `Stromstoßschalter`, das sind Bauteile die ihren Logikzustand durch einen Eingangsimpuls wechseln, realisiert werden. Dabei würde die Implementierung abstrahiert und die Übersichtlichkeit erhöht. Ebenfalls denkbar wäre die Zusammenfassung der Timer Ein- und Ausgänge in ein einziges Bauteil. Die Umsetzung beim Export könnte hierbei beibehalten werden, jedoch könnten die Timer Bausteine dann zwischengeschaltet werden, was deren Einsatz vereinfacht.

5 Literatur

[1] Eaton. *Eaton Easy*. URL: https://de.rs-online.com/web/p/prozessmodule/4891412/ (besucht am 04. 05. 2019).

[2] René Preißel. *Git: Dezentrale Versionsverwaltung im Team – Grundlagen und Workflows*. März 2017. ISBN: 9783864904523.

[3] *Codesys*. URL: https://codesys.com/ (besucht am 10. 04. 2019).

[4] *Open-PLC*. URL: https://openplc.com/ (besucht am 10. 04. 2019).

[5] Günter Wellenreuther und Dieter Zastrow. *Automatisieren mit SPS - Theorie und Praxis: Programmieren mit STEP 7 und CoDeSys, Entwurfsverfahren, Bausteinbibliotheken Beispiele für Steuerungen, ... PROFINET, Ethernet-TCP/IP, OPC , WLAN*. Mannheim, Jan. 2015.

[6] Santosh. *Architecture of PLC*. URL: https://instrumentationforum.com/t/architecture-of-plc/7059 (besucht am 10. 04. 2019).

[7] Ulrich Kanngießer. *Kleinsteuerungen in Praxis und Anwendung: LOGO!, easy, Zelio, Millenium 3, Nanoline und Safety-Steuerungen*. Dez. 2012, S. 576. ISBN: 3800732874.

[8] CircuitVerse. *CircuitVerse Projekt*. URL: https://circuitverse.org/ (besucht am 10. 04. 2019).

[9] CircuitVerse. *CircuitVerse auf GitHub*. URL: https://github.com/CircuitVerse/CircuitVerse (besucht am 10. 04. 2019).

[10] Linus Torvalds. *GIT*. URL: https://git-scm.com/ (besucht am 07. 05. 2019).

[11] *Apache NetBeans*. URL: https://netbeans.org/ (besucht am 11. 04. 2019).

[12] PiFace. *PiFace Digital 2*. URL: http://www.piface.org.uk/products/piface_digital_2/ (besucht am 29. 10. 2018).

[13] PiFace. *PiFace Digital 2 Git*. URL: https://github.com/piface/libpifacedigital (besucht am 10. 05. 2019).

[14] Thomas Preston. *libmcp23s17*. URL: https://github.com/piface/libmcp23s17 (besucht am 10. 05. 2019).

[15] Peter Leo Gorski, Luigi Lo Iacono und Hoai Viet Nguyen. *WebSockets: Moderne HTML5-Echtzeitanwendungen entwickeln (mit JavaScript programmieren)*. Carl Hanser Verlag GmbH & Co. KG, Jan. 2015, S. 281. ISBN: 3446443711.

[16] *Benefits of WebSockets*. URL: https://www.websocket.org/quantum.html (besucht am 11. 04. 2019).

[17] Boris Schäling. *Die Boost C++ Bibliotheken*. Auflage 2. XML Press, Apr. 2015, S. 600. ISBN: 1937434370.

[18] Stack Overflow. *Boolean expression (grammar) parser in c++*. URL: https://stackoverflow.com/questions/8706356/boolean-expression-grammar-parser-in-c/8707598#8707598 (besucht am 10. 04. 2019).

[19] Vinnie Valco. *Get Rich Quick! Using Boost.Beast WebSockets and Networking TS*. URL: https://github.com/vinniefalco/CppCon2018 (besucht am 01. 11. 2018).

[20] Internet Security Research Group. *Let's Encrypt*. URL: https://letsencrypt.org/ (besucht am 07. 05. 2019).

[21] *Raspbian*. URL: https://www.raspbian.org/.

[22] *Enable SPI Interface on the Raspberry Pi 9*. URL: https://www.raspberrypi-spy.co.uk/2014/08/enabling-the-spi-interface-on-the-raspberry-pi/ (besucht am 11. 05. 2019).

[23] *Systemabbild zum Download*. URL: http://owncloud.wiche.eu/download/ControlPi/ControlPi-Raspbian-Strech.tar.gz (besucht am 11. 05. 2019).

[24] *Win32 Disk Imager*. URL: https://sourceforge.net/projects/win32diskimager/ (besucht am 11. 05. 2019).

[25] OpenLX SP Ltd. *PiFace Digital 2*. URL: http://www.piface.org.uk/products/piface_digital_2/ (besucht am 04. 05. 2019).

6 Anhang

Inhalt der beigelegten DVD

\DVD
- ControlPi/ ..... Git-Auszug des Projektordners
- libmcp23s17/ ..... Git-Auszug der Bibliothek Libmcp23s17
- libpifacedigitalcpp/ .... Git-Auszug der Bibliothek Libpifacedigitalcpp
- ControlPi-Raspbian-Strech.tar.gz ..... Komplettes System-Image
- Demo-Video.mkv ..... Demo Video mit Untertitel
- talk.pdf ..... Im Rahmen der Arbeit gehaltener Vortrag
- thesis.pdf ..... Dieses Dokument

Abbildung 6.1: Inhalt der beigelegten DVD

Eingesetzte Software

In Tabelle 6.1 werden die wichtigsten Programme, die zum Entwickeln verwendet wurden, aufgelistet.

Tabelle 6.1: Liste der eingesetzten Software

Programm	Version
CMake	3.13.4
GCC	8.1.0
Boost	1.6.9
Boost Beast	2.3.5
libmcp23s17	0.3.3
libpifacedigital	3.0.1
Linux Kernel	4.14.98
Raspbian	9 (strech)
Apache	2.4.25
PHP	7.0.33
Chromium	73.0.3683.86
Firefox	66.0.3