

Fachhochschule Köln
Cologne University of Applied Sciences

Institut für Medien- und Phototechnik

Bachelorarbeit Medientechnik

Webgestütztes GPIO Management am Beispiel des BeagleBone Black

vorgelegt von

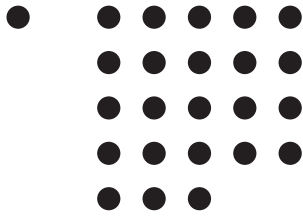
Caspar Friedrich

Mat.-Nr. 11062078

Erstgutachter: Prof. Dr. Klaus Ruelberg (Fachhochschule Köln)

Zweitgutachter: Prof. Dr. Luigi Lo Iacono (Fachhochschule Köln)

November 2014



Fachhochschule Köln
Cologne University of Applied Sciences

Institut für Medien- und Phototechnik

Bachelor Thesis

Webbased GPIO Management using the example of the BeagleBone Black

submitted by

Caspar Friedrich

Mat.-Nr. 11062078

First Reviewer: Prof. Dr. Klaus Ruelberg (Fachhochschule Köln)

Second Reviewer: Prof. Dr. Luigi Lo Iacono (Fachhochschule Köln)

November 2014

Bachelorarbeit

Titel: Webgestütztes GPIO Management am Beispiel des BeagleBone Black

Gutachter:

- Prof. Dr. Klaus Ruelberg (Fachhochschule Köln)
- Prof. Dr. Luigi Lo Iacono (Fachhochschule Köln)

Zusammenfassung: In der vorliegenden Bachelorarbeit wird ein netzwerk-gestütztes Messsystem vorgestellt, das mit Hilfe verschiedener GPIO des BeagleBone Black eine entfernte Messstationen ermöglicht. Die Besonderheit des Systems liegt darin, dass es als Basis ohne irgendeine Anwendungs-Spezialisierung besteht – also flexibel einsetzbar ist und gleichzeitig im Verhältnis zu heute gängigen Messsystemen sehr kostengünstig. Es ist einfach und vielseitig konfigurierbar und kann browser-basiert über ein Netzwerk gesteuert und überwacht werden.

Stichwörter: BeagleBone, Webinterface

Datum: Donnerstag 20 November, 2014

Bachelor Thesis

Titel: Webbased GPIO Management using the example of the BeagleBone Black

Reviewers:

- Prof. Dr. Klaus Ruelberg (Cologne University of Applied Sciences)
- Prof. Dr. Luigi Lo Iacono (Cologne University of Applied Sciences)

Abstract: In this Bachelor Thesis I present a net-supported measuring system which provides a remote measuring station by help of the BeagleBone Black GPIOs. The outstanding feature of this system is its basic configuration, that has no defined use case and thus allows a flexible implementation in various measurement situations. Consequently, the system is economical and easily applicable in almost any laboratory situation.

Keywords: BeagleBone, Webinterface

Date: Thursday 20th November, 2014

Inhaltsverzeichnis

I. Dokumentation	8
1. Einleitung	9
1.1. Zielsetzung	9
1.2. Definitionen	10
2. Grundlagen	11
2.1. Hardware	11
2.1.1. Single-Board Computer	11
2.1.2. System on a Chip	12
2.1.3. BeagleBone Black	13
2.2. Betriebssysteme	13
2.2.1. Linux	13
2.2.2. Linux Distributionen	14
2.3. Webtechnologien	15
2.3.1. Webserver	15
2.3.2. Node.js	16
2.3.3. WebSockets	16
3. Konfiguration des Betriebssystems	18
3.1. Software	19
3.1.1. Proxy Server – HAProxy	20
3.1.2. Webserver – Lighttpd	21
3.1.3. FTP-Server – vsftpd	22
3.1.4. JavaScript Engine – Node.js	22
3.2. Installation	23

4. Implementierung	24
4.1. Datenaustausch	25
4.2. Seitenaufruf	26
4.3. Interaktion	27
4.4. WebSocket Server	27
4.4.1. Models	28
4.4.2. Controller-Module	30
4.4.3. Bypass-Module	32
4.5. Website	33
4.5.1. Skriptdokumente	33
4.5.2. Bibliotheken	35
5. Erweiterungsmöglichkeiten	37
5.1. Zusätzliche Anpassungsoptionen	37
5.2. Höher aufgelöste A/D-Wandlung	37
5.3. Alternative Steuerungsbibliothek – octalbonescript	38
5.4. Alternative Hardware	38
5.5. Erweiterte Konfiguration in das Interface integrieren	39
5.6. Weitere GPIO	39
6. Zusammenfassung	40
 II. Anhang	 42
A. Betriebsanleitung	43
A.1. Hardware	43
A.2. Installation	43
A.2.1. SD-Karte vorbereiten	43
A.2.2. Installation im internen Speicher	45
A.2.3. boneserver installieren	46
A.3. Betrieb	46
A.3.1. Netzwerkverbindung herstellen	46
A.3.2. Webinterface aufrufen	47
A.3.3. Bedienelemente	48
A.3.4. Erweiterte Einstellungen	51

Inhaltsverzeichnis

A.4. Wartung	53
A.4.1. Backup	54
A.4.2. System aktualisieren	54
A.4.3. boneserver aktualisieren	55
A.4.4. System bereinigen	55
B. Tabellen	56
C. Literaturverzeichnis	57
Eidesstattliche Erklärung	59

Teil I.

Dokumentation

1. Einleitung

„We have a clear vision – to create a world where every object - from jumbo jets to sewing needles – is linked to the Internet.“[1, Helen Duce, Seite 1]

Das *Internet of Things* ist ein rasant wachsender Anwendungsbereich. Angetrieben von einer zunehmenden Akzeptanz digitaler Systeme und der günstigen Entwicklung der Baugröße, Leistung und Zuverlässigkeit und nicht zuletzt der sinkende Preis haben dazu geführt, dass digitale Systeme heute in allen Lebensbereichen anzutreffen sind [1, 2].

Es ist inzwischen selbstverständlich, Steuerungssysteme in der Industrie, zum Teil sogar schon im privaten Haushalt, miteinander zu vernetzen und im Netzwerk zugänglich zu machen. So ist es möglich, die vorhandenen Daten jederzeit auch extern z. B. via Mobiltelefon abzurufen. Heute sind viele Geräte und Programme sehr spezialisiert, so dass man in komplexen Arbeitszusammenhängen viele verschiedene Systeme braucht. Dadurch ist oft auch ein hoher finanzieller Aufwand erforderlich.

Gleichzeitig hat eine Annäherung der Hardware-Hersteller an die „Hobby“-Entwickler stattgefunden und Projekte wie Arduino und Co. haben den Entwicklungsaufwand eigener Hard- und Software erheblich reduziert. Damit wird es erheblich erleichtert, bei individuellen Fragestellungen selber individuelle Lösungen zu entwickeln. Hat man nun viele individuelle Lösungen gefunden und somit viele verschiedenartige Messdaten erhalten, besteht die Notwendigkeit, diese wiederum zusammenzuführen.

1.1. Zielsetzung

Aus dieser Tendenz entstand die Idee, die unterschiedlichen im Laboralltag anfallenden Messungen mit einem einzigen flexiblen System steuern und die Resultate gemeinsam verfügbar zu machen.

Ziel dieser Arbeit ist es, ein Steuersystem für Messanwendungen zu entwickeln, das sich einfach konfigurieren lässt, flexibel in der Anwendung ist und gleichzeitig kostengünstig bleibt. Besonderes Augenmerk soll dabei auf der ausreichenden Verfügbarkeit verschiede-

1. Einleitung

ner General Purpose Input/Output (GPIO) liegen. Insbesondere Pulsbreitenmodulation und Analog/Digital-Konverter sind für Messanwendungen interessant. Die anfallenden Messdaten sollen protokolliert werden und extern verwendbar sein. Die zu entwickelnde Applikation soll auf keinen bestimmten Anwendungsfall spezialisiert sein. Sie soll vielmehr dem Anwender die Möglichkeit geben, sich eine für sein jeweiliges Projekt passende Umgebung zusammenzustellen. Weiter soll das System ohne ständige Überwachung arbeiten können. Es soll möglich sein, eine entfernte Messstation aufzubauen, die nach belieben via LAN, WLAN oder auch GSM konfiguriert und überwacht werden kann.

[TODO: GRAFIK]

1.2. Definitionen

In dieser Arbeit wird ein Singleboard-Computer des Typs **BeagleBone Black Rev. A5C** verwendet. Andere Versionen dieses Computers sind, sofern kompatibel, ebenfalls verwendbar, allerdings nicht getestet. Um eine gute Lesbarkeit zu ermöglichen ist mit „BeagleBone“ im Folgenden immer diese Version gemeint.

2. Grundlagen

2.1. Hardware

2.1.1. Single-Board Computer

Ein Single-Board Computer (SBC), zu deutsch Einplatinenrechner, ist ein Computersystem, bei dem alle für die Verwendung nötigen Bauteile auf einer einzelnen Platine aufgebracht sind. Hierbei sind neben den essenziellen Komponenten wie Prozessor, RAM und ROM auch Controller für verschiedene I/O-Schnittstellen, Oszillatoren oder Co-Prozessoren verbaut. Single-Board Computer werden vor allem in der Industrie als Steuersysteme eingesetzt, da sie oft billiger und flexibler sind als fest verdrahtete Steuersysteme. Mit fortschreitender Miniaturisierung und steigender Leistungsfähigkeit finden SBC's heute auch in alltäglichen Geräten wie Autos, Waschmaschinen oder auch Fernbedienungen Verwendung.

Technisch gesehen sind auch erste Heimcomputer wie der *C64* oder *Atari ST* Single-Board Computer, allerdings lassen sich diese ohne Ein- und Ausgabegeräte wie Maus, Tastatur, Bildschirm nicht sinnvoll nutzen und werden in der Regel nicht als solche bezeichnet.

Schnittstellen

Single-Board Computer verfügen, je nach Anwendungsgebiet, über eine Vielzahl verschiedener analoger und digitaler I/O-Schnittstellen.

Übliche Schnittstellen sind:

- GPIO, darunter digitale I/O, Pulsbreitenmodulation (engl. Pulse Width Modulation) (PWM) und Analog/Digital Converter (ADC)
- Universal Asynchronous Receiver Transmitter (UART)

- Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I²C)

Über UART ist eine Implementierung der verbreiteten RS232/422/485-Schnittstellen möglich und auch üblich.

Aktuelle (Entwickler-)Systeme haben in der Regel einen oder mehrere USB-Anschlüsse (sowohl Client als auch Host Ports sind üblich) oder zumindest einen JTAG-Port, was die Programmierung wesentlich vereinfacht. In der Regel verfügen leistungsstärkere Systeme auch über einen Grafikausgang, Oft HDMI oder eine der Miniaturvarianten.

2.1.2. System on a Chip

Eng verknüpft mit der Entwicklung der SBC ist das Konzept des System on a Chip (SOC). Hierbei werden die meisten oben genannten Komponenten eines Systems direkt in einem einzelnen IC verbaut. Meist sind nur ROM und Controller für höhere Schnittstellen USB oder LAN (in manchen Fällen auch Grafik) extern angebunden.

Heutige Single-Board Computer mit einem SOC können sehr leistungstark sein, sind als Mehrkernsystem aufgebaut und haben Taktraten von mehreren GHz. Diese Computer sind vom Design her stark an Desktop-Systeme angepasst und können mit einem vollwertigen Linux- oder Windows-System betrieben werden.

Gerade bei diesen leistungsstarken SOC's hat sich die ARM-Architektur durchgesetzt. 1983 als Nebenprojekt gegründet hatte die 32-Bit-Architektur bereits 2002 einen Marktanteil von fast 80% [3].

SBC lassen sich (sehr) grob in zwei Klassen unterteilen:

1. Leistungsschwache Systeme

Die Taktraten dieser Prozessoren liegen üblicherweise deutlich unter 50MHz, in seltenen Fällen bis 100MHz. Diese Mikrocontroller werden meist direkt programmiert und finden vor allem im low energy-Sektor Anwendung.

2. Leistungsstarke Systeme

Hier liegen die Taktraten meist im GHz-Bereich. Hauptanwendungsbereiche sind Mobilfunksysteme und embedded computing in der Industrie. Gerade im Mobilfunkbereich sind oft Mehrkernsysteme anzutreffen und es wird bis auf wenige Ausnahmen oberhalb eines Betriebssystems, meist Linux bzw. Android, programmiert.

2.1.3. BeagleBone Black

Für diese Arbeit verwende ich einen BeagleBone Black Rev. A5C (im Folgenden BeagleBone), ein Entwickler-Board mit einem ARM® Cortex™-A8 Prozessor (Single Core) von Texas Instruments.

Die wichtigsten Features:

- 1GHz Taktrate
- 512MB DDR3 RAM
- 2GB¹ Onboard Flash Memory
- 10/100 Mbit/s Ethernet
- 69² GPIO mit mehreren PWM-Ausgängen und ADC-Eingängen
- Verhältnismäßig geringer Preis von ca. 45 €

2.2. Betriebssysteme

Da die Ressourcen des BeagleBone Black sehr begrenzt sind, wird für diese Arbeit ein schlankes Betriebssystem benötigt, welches nur wenig Speicher verbraucht und geringen Leistungs-Overhead verursacht. Für diesen Zweck gibt es spezielle Versionen der bekannten Betriebssysteme wie Microsoft Windows oder Linux sowie verschiedene „un-ixoiden“ Betriebssysteme.

2.2.1. Linux

Linux hat den Vorteil, dass nahezu alle Software als Quellcode verfügbar ist und im Zweifel angepasst werden kann. Zudem ist es üblich Lizenzen zu verwenden, die eine nicht-kommerzielle Anwendung sowie Anpassungen kostenfrei zulassen.

Ein eigenes Linux zu entwickeln oder ein Build System³ zu verwenden wäre aus Sicht der Performance sicherlich die beste Wahl und ist in der Industrie weitgehend üblich.

¹4GB ab Rev. C

²27 GPIO sind ohne weitere Konfiguration direkt verfügbar

³Einige Distributionen verwenden ein sog. Build System, bei dem die benötigten Kernelmodule und Softwarepakete selbst zusammengestellt werden können.

Das würde allerdings den Rahmen dieser Arbeit sprengen. Zudem gibt es einige sehr schlanke und bereits für den BeagleBone angepasste Linux Distributionen.

2.2.2. Linux Distributionen

BeagleBoard.org bietet für den BeagleBone Black zwei verschiedene Distributionen an: Ångström und Debian. Beide Distributionen haben Vor- und Nachteile, die weiter unten erläutert werden. Ein weiteres Projekt, welches sich unter Entwicklern großer Beliebtheit erfreut ist Arch Linux, das als Basis für diese Anwendung dienen soll.

The Ångström Distribution

The Ångström Distribution ist auf dem BeagleBone vorinstalliert und stellt die Hauptdistribution dar. Diese Distribution nutzt ein Build System und findet im Wesentlichen Anwendung bei Speichersystemen wie NAS oder FTP-server, wichtigstes Feature ist daher der geringe Leistungs- und Speicherbedarf. Bei dieser Distribution muss allerdings nahezu jede nicht-standard Software selbst kompiliert und eingerichtet werden.

Debian Linux

Im Allgemeinen gilt Debian Linux als (rock-)stable und ist eine der verbreitetsten Distributionen. Zudem basieren einige weitere namhafte Distributionen auf Debian Linux. Stärke und gleichzeitig auch Schwäche dieser Distribution sind die langen und umfangreichen Softwaretests. Wenn ein Paket in den offiziellen Repositories verfügbar ist kann man zwar davon ausgehen, dass es fehlerfrei funktioniert und mit allen anderen angebotenen Paketen kompatibel ist. Allerdings liegt es meist nicht mehr in der aktuellen Version vor. Das kann gerade bei Software aus dem Bereich Netzwerk/Internet problematisch werden.

Arch Linux

Gegenüber den oben genannten Distributionen hat Arch Linux zwei wesentliche Vorteile: Zum einen gibt es eine (Sub-)Distribution speziell für ARM-Prozessoren, bei der das Basissystem mit ca. 500MB sehr schlank ist. Zum anderen stellt Arch Linux ARM ein sehr umfangreiches Software Repository mit hoher Aktualität zur Verfügung. Zusätzlich gibt es das Arch User Repository (AUR), ein freies Repository in dem jeder Nutzer seine Pakete einstellen kann. Sämtliche in diesem Projekt verwendete Software lässt sich

entweder direkt via Paketverwaltung aus den offiziellen Repositories installieren oder kann vom Anwender selber kompiliert werden.

Arch Linux ARM verwendet ein Rolling-Release-Konzept, ein System kontinuierlicher Software-Entwicklung, bei der Pakete separat aktuell gehalten und weiter entwickelt werden. Es gibt keine explizite Betriebssystemversion sondern sog. Snapshots. So ist es wesentlich einfacher, das System aktuell und sicher zu halten. Zwar kann es durchaus passieren, dass die eingestellte Software nicht out-of-the-box funktioniert. Aber in der sehr aktiven Community hinter Arch Linux bekommt man relativ schnell Hilfe.

Die Kernel-Entwicklung schreitet derzeit sehr schnell voran, daher wird zusätzlich zur regulären Kernel-Entwicklung ein Legacy-Paket mit einer stabilen Version (3.8) gepflegt, um nach einem Update des Kernelpaketes nicht erst die Kompatibilität wieder herstellen zu müssen.

2.3. Webtechnologien

In dieser Arbeit kommen eine Vielzahl unterschiedlicher Webtechnologien zum Einsatz. An dieser Stelle alle zur Verfügung stehenden Technologien im Detail zu beschreiben, würden den Rahmen der Arbeit sprengen. Ich beschränke mich hier darauf, die Architektur des Projektes zu darzustellen. Meine Auswahl an Software erläutere ich im Zusammenhang in Kapitel 4: Implementierung.

2.3.1. Webserver

Als Webserver wird in der Regel eine Software bezeichnet, die verschiedene Dokumente an einen Client, z. B. ein Webbrowser, ausliefert. Oft wird auch die ganze Hardware, auf der ein Webserver betrieben wird, als Webserver bezeichnet. Dies ist dann der Fall, wenn der Rechner ausschließlich zu diesem Zweck betrieben wird, z. B. bei größeren Webseiten.

Auf Grund verschiedener Spezialisierungen und Ansätze gibt es heute eine große Anzahl verschiedener Webserver. Apache HTTP Server der gleichnamigen Firma und Microsofts IIS haben sich hier in den letzten Jahren deutlich von der Konkurrenz absetzen können. [4]. IIS kommt für diese Arbeit allerdings nicht in Frage, da er nicht für Linux angeboten wird.

Lighttpd

Lighttpd ist ein Webserver, der durch seine besonders ressourcenschonende Implementierung hervorsticht. Dieser Webserver ist darauf spezialisiert, statische Dokumente auszuliefern, was ihn besonders passend für diese Arbeit macht. Außerdem lässt er sich wesentlich einfacher konfigurieren und über Module erweitern als andere weiter verbreitete Webserver [5].

2.3.2. Node.js

Node.js ist ein JavaScript-Framework, das auf Googles „V8“ JavaScript-Engine basiert. Es stellt eine serverseitige Umgebung zur Verfügung, über die sich komplexe Netzwerkanwendungen leicht programmieren lassen. Dabei stellt Node.js ein Modulkonzept vor, mit dem man relativ unkompliziert Programmteile auslagern oder weitere Funktionalität einbinden kann. In der Basisinstallation liefert Node.js bereits eine große Bandbreite an Modulen mit, sodass viele Anwendungen ohne zusätzliche Software umgesetzt werden können [6].

npm

Der Node Package Manager erweitert Node.js um ein Werkzeug, mit dem sich externe Module leicht herunterladen und einbinden lassen. Unter npmjs.org können zudem weitere Informationen und in der Regel auch eine API der Module eingesehen werden. Dank einer großen Community lassen sich hier zu den meisten allgemeinen Problemen bereits einige Lösungsansätze finden.

bonescript

Ausschlaggebend für die Verwendung von Node.js ist das Modul *bonescript* von Jason Kridner, Mitgründer der BeagleBoard.org Foundation. Dieses Modul steuert über eine übersichtliche API weite Teile der BeagleBone-Hardware, sodass auch hier der Arbeitsaufwand deutlich verringert wird.

2.3.3. WebSockets

WebSockets stellen eine Möglichkeit dar, Daten zwischen Client (Browser) und Server auszutauschen. Wie auch Hypertext Transfer Protocol (HTTP) basieren WebSockets auf

2. Grundlagen

Transmission Control Protocol (TCP), sind dabei aber wesentlich schneller. Im Gegensatz zu HTTP ist über WebSockets eine Vollduplex-Verbindung möglich. Die Metadaten sind zudem überschaubar, sodass hierbei wenig zusätzliche Netzwerklast entsteht. Ein weiterer Vorteil ist, dass WebSockets von der Same-Origin Policy ausgeschlossen sind [7].

3. Konfiguration des Betriebssystems

Ein essentieller Punkt zu Beginn des Projektes ist die Auswahl des Betriebssystems, da im weiteren Verlauf alle Entwicklungen darauf aufbauen werden. Außerdem ist ein Teilziel des Projektes, schon existierende Software zu nutzen, um den Entwicklungsaufwand klein zu halten und den Fokus auf die eigentliche Fragestellung zu richten. Also habe ich das bereits auf dem BeagleBone vorinstallierte Betriebssystem - Ångström Linux - intensiv getestet.

Die grundlegenden Fragen waren:

- Gibt es genügend Speicherplatz für die zu entwickelnde Software?
Wie schaffe ich bei Bedarf weiteren Speicherplatz für meine Anwendungen?
- Wird die zusätzlich gebrauchte Software (Webserver, FTP-Server, etc...) in einem Repository zur Verfügung gestellt?
- Sind ausreichend Rechenleistung und Arbeitsspeicher verfügbar, um zusätzliche Dienste zu betreiben?
- Sind die benötigten Systemressourcen (Netzwerkports u.a.) verfügbar, bzw. kann ich sie ohne großen Aufwand freigeben?

Die Tests ergaben, dass das Betriebssystem mit einer graphischen Oberfläche und mehreren Netzwerkdiensten ausgeliefert wird. Die graphische Oberfläche verbraucht Systemressourcen und Speicherplatz, ist aber überflüssig, da das System headless betrieben werden soll. Dazu laufen ein Apache Webserver, eine Cloud9 Entwicklungsumgebung und ein Websocket Server. Des weiteren verwendet Ångström Distribution parallel zwei verschiedene Daemon Manager (systemd und SysVinit). Diese erschweren das Arbeiten mit Ångström Linux erheblich.

Die Software, die ich für mein Projekt verwenden möchte, liegt nicht in den offiziellen Repositories vor. Also müßte ich sie selber konfigurieren und kompilieren.

3. Konfiguration des Betriebssystems

Aus diesen Erfahrungen folgt, dass Ångström Linux nicht das richtige Betriebssystem für mein Projekt ist. Der Aufwand, es an meine Bedürfnisse anzupassen, wäre größer als auf eine andere Distribution zurückzugreifen.

Die zweite vom Hersteller des BeagleBone angebotene Distribution ist Debian Linux. Auch diese habe ich nach den oben genannten Kriterien geprüft. Die Konfiguration, die schon zum Ausschluss von Ångström Linux führte, fanden sich im Prinzip auch hier. Auf ARMhf.com gibt es zwar eine weitere Debian-Installation bereits fertig als Linux Image. Diese hat nicht die oben erwähnten Eigenheiten, nutzt aber die neueste Kernel-Version, die mit der Bonescript Library nicht kompatibel ist. Ein Kernel-Downgrade wäre möglich, ist aber grundsätzlich und unter Debian im Besonderen sehr aufwändig. Daher eignet sich Debian Linux ebenfalls nicht für mein Projekt.

Eine weitere Distribution für BeagleBone ist Arch Linux. Die Besonderheit dieser Distribution ist, dass die Basisinstallation sehr Ressourcen schonend ist. Sie enthält keinerlei Software, die nicht für den grundsätzlichen Betrieb erforderlich ist. Gleichzeitig stellt sie Werkzeuge zur Verfügung, um die benötigte Software bequem aus umfangreichen Repositories zu installieren. Abweichend von der regulären Kernel-Entwicklung wird die hier benötigte Kernel-Version (3.8) und entsprechende Header Files über die Pakete *linux-am33x-legacy* und *linux-headers-am33x-legacy* bereitgestellt.

Somit ist meines Erachtens Arch Linux das richtige Betriebssystem für dieses Projekt. Darüber hinaus kenne ich es seit langem, da ich es beruflich wie privat auf Desktop Computern verwende. Ich muss mich also nicht in eine neue Distribution einarbeiten.

3.1. Software

Zusätzlich zu den mitgelieferten Paketen der Distribution werden noch einige zusätzliche Pakete benötigt, die aber keiner besonderen Konfiguration benötigen und daher hier nicht näher erklärt sind. In Tabelle B.1 findet sich eine Liste mit diesen Paketen.

Desweiteren werden ein HTTP-Server und die JavaScript Engine für den WebSocket Server benötigt. Der Proxy-Server sorgt dafür, dass die Seite von außen über einen einzigen Port erreichbar ist und dass mit geringem Aufwand Secure Socket Layer (SSL)-

3. Konfiguration des Betriebssystems

verschlüsselte Verbindungen hergestellt werden können. Zusätzlich wird noch ein FTP-Server installiert, um Messdaten ohne Webinterface abrufen zu können (Abb. 3.1).

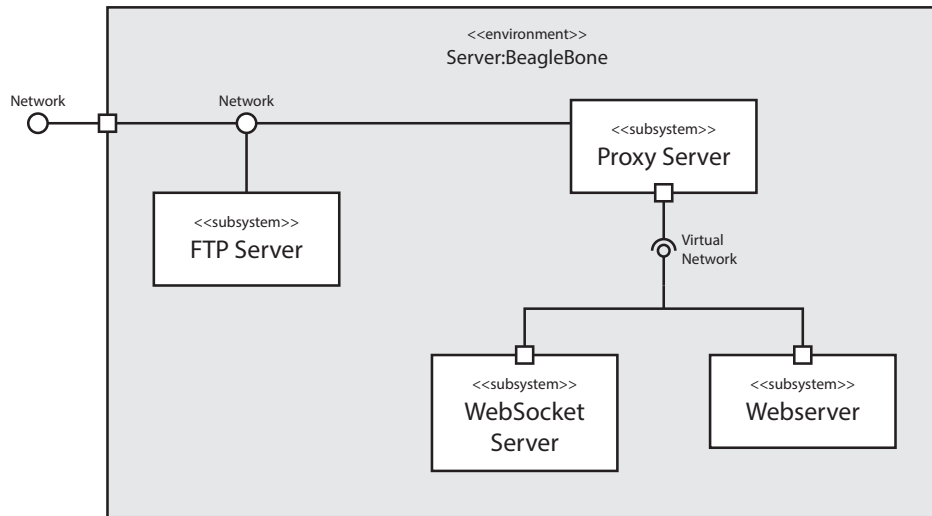


Abbildung 3.1.: Komponenten des boneserver

3.1.1. Proxy Server – HAProxy

HAProxy ist ein Proxy-Server, der eigentlich eingesetzt wird, um Anfragen via TCP auf mehrere Server zu verteilen. Wesentlich interessanter für diese Arbeit ist allerdings, dass der HAProxy, ab Version 1.5, sog. SSL-Offloading unterstützt und nativ SSL-verschlüsselte Verbindungen verarbeiten kann. Dabei erfüllt bereits eine einfache Basis-Konfiguration bereits diese Aufgabe [8].

In diesem Projekt wird HAProxy eingesetzt, um WebSocket Requests von regulären HTTP Requests zu trennen und auf unterschiedliche Dienste weiterzuleiten. Ziel dieser Maßnahme ist es, nach außen die gesamte Website hinter einem Port zu betreiben, obwohl die beiden Prozessen völlig von einander getrennt sind. So ist die Gefahr minimal, dass bei einem Feldeinsatz der Port für den WebSocket Server von einer Firewall blockiert wird. Die Website ist somit entweder vollständig erreichbar oder gar nicht. Auch ist der WebSocket Server, der systembedingt mit root-Rechten laufen muss, ausschließlich per WebSocket über den Proxy-Server zu erreichen und so gegenüber Angriffen von außen weitgehend sicher.

Ein weiterer wichtiger Punkt ist, dass sich für jeden Server die maximale Anzahl der aktiven Verbindungen bequem per Konfigurationsdatei einstellen lassen. So wird ohne besondere Programmierung sichergestellt, dass immer nur eine Verbindung zum Web-

Socket Server besteht. Alle weiteren Verbindungsanfragen werden auf eine Warteliste gesetzt und weitergeleitet, sobald der Websocket Server wieder frei wird.

Die Konfigurationsdatei findet sich im Repository unter *config/haproxy/haproxy.cfg* und wird bei der Installation automatisch verlinkt. Hervorgehoben werden sollen zwei Parameter:

```
maxconn 1
```

weist HAProxy an, wie oben beschrieben, nur eine aktive Verbindung zu diesem Server zuzulassen.

```
ssl crt /opt/boneserver/config/haproxy/testcert.crt
```

macht das Einbinden eines SSL-Zertifikates möglich. Nach der Installation ist hier ein Testzertifikat eingetragen um, SSL-Verbindungen zumindest technisch testen zu können.

3.1.2. Webserver – Lighttpd

Lighttpd wird in dieser Arbeit verwendet, um statische HTML-Dokumente und JavaScript-Dokumente auszuliefern und die Software vor nicht autorisierten Zugriffen zu schützen. Die Konfigurationsdatei findet sich unter *config/lighttpd/lighttpd.conf*.

Abweichend von der üblichen Verwendung eines Webserver sind einige Parameter hervorzuheben.

```
server.bind = "localhost"  
server.port = 8080
```

Abbildung 3.2.: Lighttpd ist nur lokal über Port 8080 erreichbar

Abbildung 3.2 zeigt, dass der Webserver ausschließlich lokal erreichbar ist. Weiter darf Lighttpd nicht an Port 80 arbeiten, um Konflikte mit HAProxy zu vermeiden.

In Abbildung 3.3 wird beschrieben, wie der Zugriff auf einzelne Verzeichnisse via Digest Access Authentication beschränkt wird. So wird sichergestellt, dass der boneserver nur von autorisierten Nutzern gesteuert werden kann.

Jeder *auth.require*-Block stellt eine Zugriffsregelung dar. User werden in der Datei *lighttpd.user* eingetragen, diese Einträge können bequem über diverse Online-Generatoren

```
auth.backend = "htdigest"
auth.backend.htdigest.userfile = "/etc/lighttpd/lighttpd.user"
auth.require = (
    "/" => (
        "method" => "basic",
        "realm" => "Administrators",
        "require" => "valid-user"
    )
)
```

Abbildung 3.3.: Digest Access Authentication-Konfiguration

erstellt werden. Das hat den Vorteil, dass bei einer eventuellen Weiterentwicklung der Software neue User und auch weitere Zugriffsebenen implementiert werden können. Das dafür benötigte Modul ist *mod_auth* und wird unter *server.modules* eingetragen.

3.1.3. FTP-Server – vsftpd

Um die Verwendung in einer weitgehend automatisierten Umgebung zu ermöglichen, ist vsftpd als FTP¹-Server installiert. Messdaten können dann auch per FTP abgerufen und verwaltet werden. Neben den üblichen Einstellungen wird über die beiden Parameter

```
local_enable=YES
anonymous_enable=NO
```

sichergestellt, dass auch hier nur autorisierte Nutzer Zugriff haben. Hierbei verwendet vsftpd die lokalen Nutzerkonten, in diesem nur das root-Konto, zur Authentifikation. Vsftpd lässt sich dabei leicht dahingehend erweitern, dass auch zusätzliche Nutzer mit spezifischen Zugriffsrechten eingerichtet werden können.

3.1.4. JavaScript Engine – Node.js

Zusätzlich zu den bereits enthaltenen Modulen werden noch drei Module benötigt, um einen WebSocket Server zur Verfügung zu stellen und die Steuerung der GPIO zu realisieren. Die Module selbst bedürfen keiner weiteren Konfiguration.

bonescript Dieses Modul übernimmt die Steuerung der Hardware. Mit Hilfe der Device Tree Overlays kann diese Bibliothek unter anderem PWM-Ausgänge, analoge Eingänge und digitale I/O verwalten.

¹File Transfer Protocol

shelljs	Dieses Modul ermöglicht es Shell-Befehle aus einem JavaScript/Node.js-Programm heraus auszuführen. <i>shelljs</i> nutzt dazu Wrapper-Funktionen um die direkte Ausführung aus Sicherheitsgründen zu verhindern.
ws	Diese WebSocket-Implementierung in JavaScript/Node.js wird von vielen komplexeren WebSocket-Modulen als Grundlage bzw. als WebSocket-Unterstützung verwendet und folgt vollständig der rfc6455. Das Modul ist sehr einfach zu verwenden, fehlende Funktionalität gegenüber z. B. <i>Socket.IO</i> fällt in diesem Projekt nicht ins Gewicht, da der Webserver separat via Lighttpd zur Verfügung gestellt wird.

3.2. Installation

Die gesamte Software liegt bei Github² als git Repository vor und wird auch von hier aus installiert. So wird sichergestellt, dass aktuelle Änderungen und Bugfixes in jede Neuinstallation übernommen werden. Auch die Aktualisierung bestehender Installationen kann vorgenommen werden.

Der Webserver, der FTP-Server und die *boneserver*-Software selbst wird bei Systemstart über einen *systemd*-Service gestartet. Diese Services werden bei der Installation ebenfalls automatisch installiert und gestartet.

Die Installation wird über das Shell-Skript *install.sh*, welches sich im root-Verzeichnis des Repositories befindet, gestartet. So wird sichergestellt, dass alle Komponenten des Systems vorhanden sind und funktionieren. Im Anhang befindet sich eine Vollständige Installations- und Betriebsanleitung.

²<https://github.com/XMrVertigoX/boneserver>

4. Implementierung

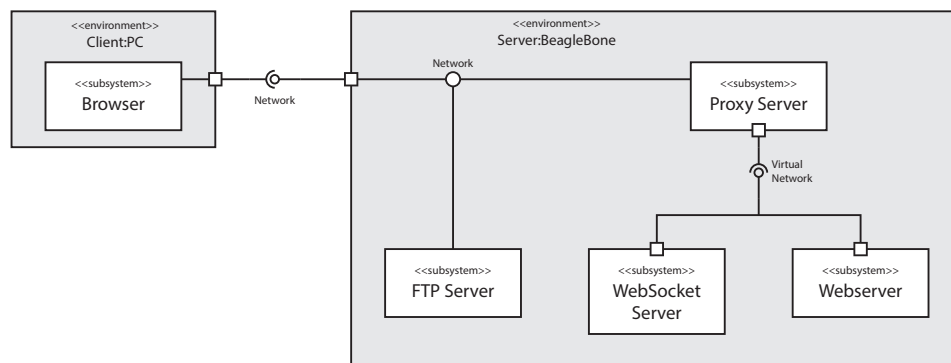


Abbildung 4.1.: Funktionale Komponenten

Das Webinterface besteht aus drei funktionalen Teilen. Der erste Teil ist der Webbrowser, der ausschließlich als Client dient. Zweitens besteht das Webinterface aus dem WebSocket Server, der die Steuerung der GPIO erledigt und die Daten für die Konfiguration der Weboberfläche liefert und drittens aus einem Webserver, der die Dokumente ausliefert. Um nach außen über einen einzigen Port zu kommunizieren, ist ein Proxy Server vorgelagert (Abb. 4.1), der den Datenverkehr an Hand des verwendeten Protokolls an den richtigen Server weiterleitet (Abb. 4.2).

Die Auslagerung des Proxy Servers hat folgende Vorteile:

1. Das SSL-Offloading wird von HAProxy automatisch vorgenommen.
2. Der Webserver ist nicht direkt zu erreichen. D. h. eventuelle Einbruchversuche über Fehler in der HTTP-Implementierung von Lighttpd werden bereits im Proxy Server abgefangen, da ungültige HTTP-Header nicht weiterverarbeitet werden.
3. Um die direkte Hardware-Steuerung zu übernehmen, muss der WebSocket Server mit root-Rechten betrieben werden. Da WebSocket Request nicht durch den Webserver geleitet werden, ist das System selbst bei einer Übernahme des Webserver weitgehend sicher. Zudem ist für den Betrieb keinerlei Kommunikation zwischen Webserver und WebSocket Server notwendig (Vgl. Abb. 4.5).

4. Implementierung

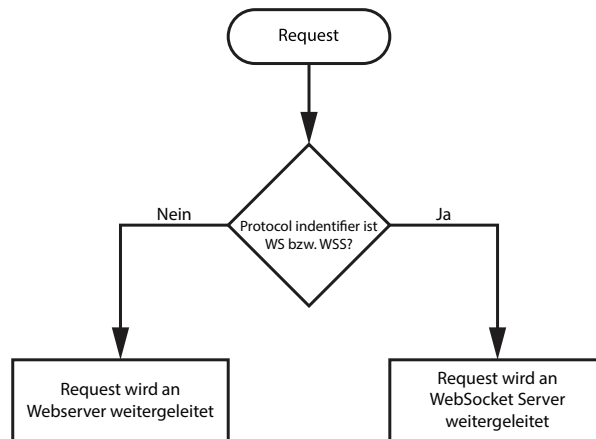


Abbildung 4.2.: Verarbeitungsschema eines Requests am Proxy Server

4. Das Projekt gestaltet sich übersichtlicher, da die einzelnen Komponenten über virtuelle Netzwerkverbindungen kommunizieren und somit leichter zu warten sind.
5. HAProxy ist im Feldeinsatz erprobt und kann angesichts der weiten Verbreitung auf namhaften Webseiten als ausreichend stabil angenommen werden [8], während die Proxy-Unterstützung in Lighttpd nur mäßig Dokumentiert ist.

4.1. Datenaustausch

Der gesamte Datenaustausch des Interfaces läuft über die WebSocket-Verbindung ab. Dabei beinhaltet jede Message den Key *type*, mit dem jeweils im Browser bzw. WebSocket Server entschieden wird, wie weiter zu verfahren ist. Das *parameters*-Objekt enthält alle weiteren Informationen (Abb. 4.3).

```
{
  type: "getPinMode",
  parameters: {
    pin: "P9_33"
  }
}
```

Abbildung 4.3.: Auszug aus einer Request Message

Um Antworten richtig zuordnen zu können, werden Rückgabewerte der Funktionen in einem *response*-Objekt an die ursprüngliche Message angehängt. Alle Daten die zu die-

ser Antwort geführt haben, sind dann noch vorhanden und können über einen Response Handler verarbeitet werden.

```
{
  type: "getPinMode",
  parameters: {
    pin: "P9_33"
  },
  response: {
    pin: "P9_33",
    name: "AIN4"
  }
}
```

Abbildung 4.4.: Auszug aus einer Response Message

Durch diese Technik ist gewährleistet, dass das Interface immer aktiv bleibt und nicht durch eine langsame Netzwerkanbindung blockiert wird. Gleichzeitig werden asynchron eingehende Messages zeitlich unabhängig verarbeitet.

4.2. Seitenaufruf

Um die Systembelastung durch das Webinterface möglichst gering zu halten, wird die Webseite dynamisch über Templates in weiten Teilen erst im Browser generiert. Dabei wird der Nutzer zunächst via HTTP Digest Authentication authentifiziert. Ist diese erfolgreich, werden die Dokumente der Webseite an den Browser ausgeliefert.

Abbildung 4.5 zeigt schematisch, wie die Initialisierung der Seite abläuft. Das Diagramm zeigt horizontal die vier beteiligten Entitäten und vertikal, ähnlich einem Sequenzdiagramm, den zeitlichen Ablauf. Antworten sind dabei immer Event-basiert. Gestrichelte Linien zeigen Antworten tieferliegender Schichten und Protokolle. Sie sind nicht Teil dieser Implementierung und sollen den Request/Response-Verlauf verdeutlichen. Die Authentifizierung ist im Webserver implementiert und Teil der Kommunikation zwischen Webbrowser und Webserver [9]. Sie ist daher nicht im Diagramm explizit dargestellt.

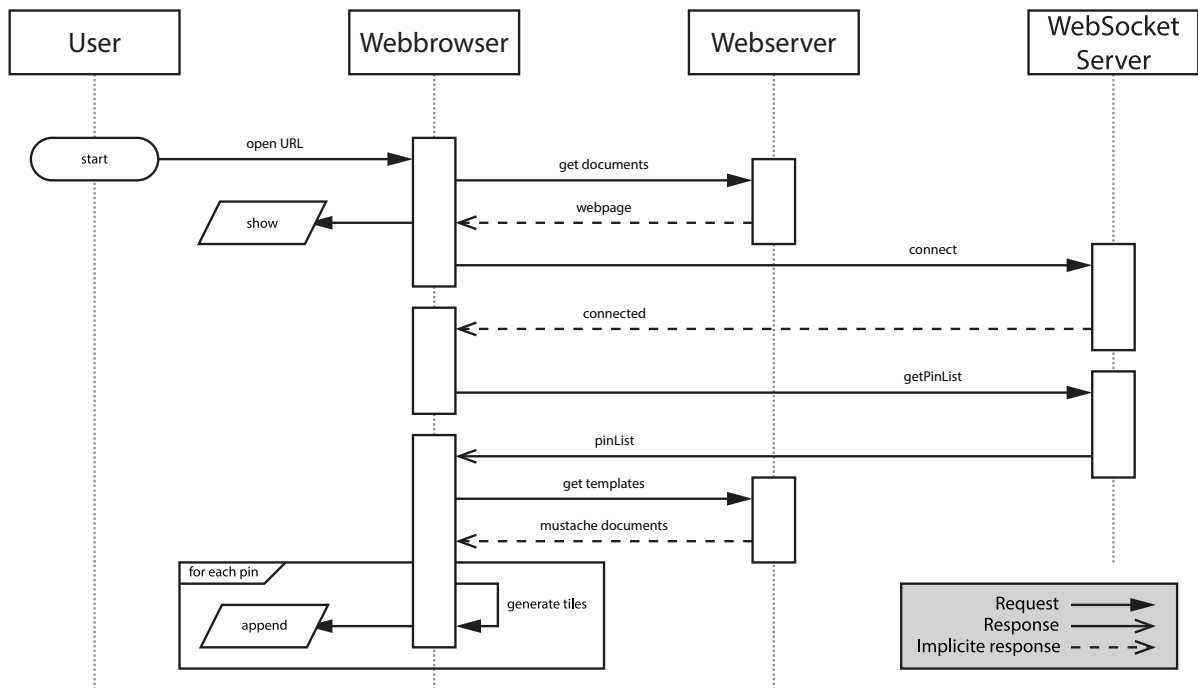


Abbildung 4.5.: Sequezieller Ablauf bei einem Seitenaufruf

4.3. Interaktion

Die Kommunikation mit dem WebSocket Server läuft vollständig asynchron ab. Es wird grundsätzlich keine direkte Antwort auf eine Anfrage erwartet um bei eventuell schlechter Netzwerkkonexion nicht das Interface zu blockieren. Abbildung 4.6 zeigt diesen Vorgang exemplarisch.

4.4. WebSocket Server

Der WebSocket Server ist vollständig in JavaScript/Node.js implementiert und modular aufgebaut. Alle notwendigen Dateien finden sich im Verzeichnis *node/*.

Eine Besonderheit von JavaScript/Node.js ist es, dass sich Module grundsätzlich ähnlich wie ein Singleton Pattern verhalten. Dabei wird bei einem erneuten Aufruf von *require()* keine neue Instanz erzeugt, sondern Referenzen auf die Erste übergeben.

So müssen bei asynchronen Funktionsaufrufen nicht alle benötigten Variablen übergeben werden und es wird sichergestellt, dass alle Funktionen auf denselben WebSocket zugreifen. Dies ist vor allem wichtig, da Intervalfunktionen, die via *setInterval()* aufgerufen werden, keinen direkten Zugriff erlauben. Zudem sollen laufende Timer bei einem er-

4. Implementierung

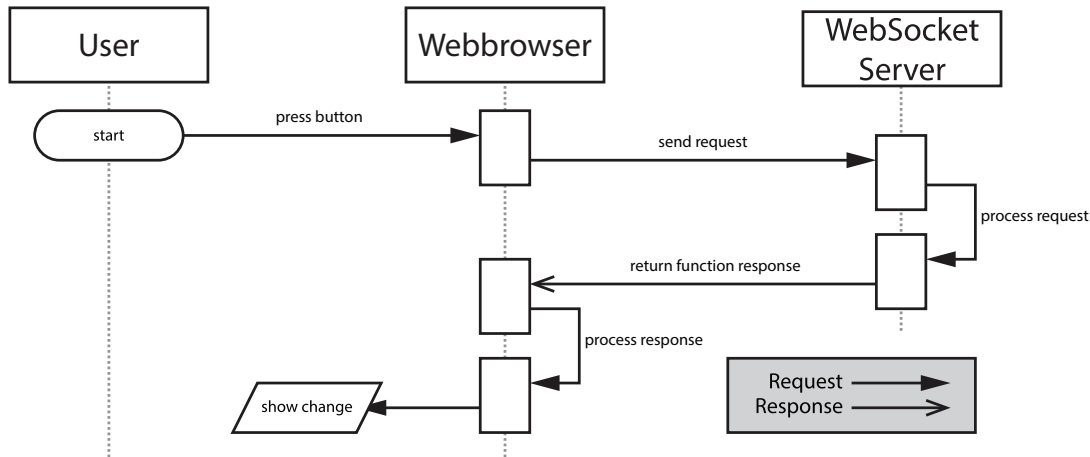


Abbildung 4.6.: Verarbeitung einer User-Interaktion

neuten Besuch der Seite (oder eine reload) ihre Daten direkt wieder an den Browser senden.

Abbildung 4.7 zeigt die Abhängigkeit zwischen den einzelnen Modulen. Im Kasten unter dem Modulnamen sind Abhängigkeiten dargestellt, die nicht Teil dieser Implementierung sind.

index.js

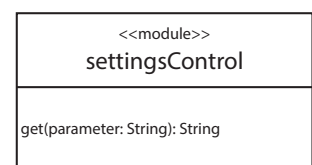
Die `index.js` stellt das Hauptdokument dar. Von hier aus wird der Server gestartet. Über den Aufruf `require()` werden die Module des Servers geladen und die Event Listener für die WebSocket-Verbindung registriert.

Bei einem erfolgreichen Verbindungsaufbau wird der WebSocket im `websocket`-Objekt referenziert und für alle anderen Module zugänglich gemacht.

4.4.1. Models

settingsControl.js

Das Modul `settingsControl.js` liest die Einstellungsdateien ein und stellt diese programmweit zur Verfügung. Das Modul nutzt dafür die Datei `settings-default.json`, in der alle Basiseinstellungen enthalten sind. Diese Einstellungen werden dann von denen in der Datei `settings.json` überschrieben, sofern das Dokument vorhanden ist. Bei den



4. Implementierung

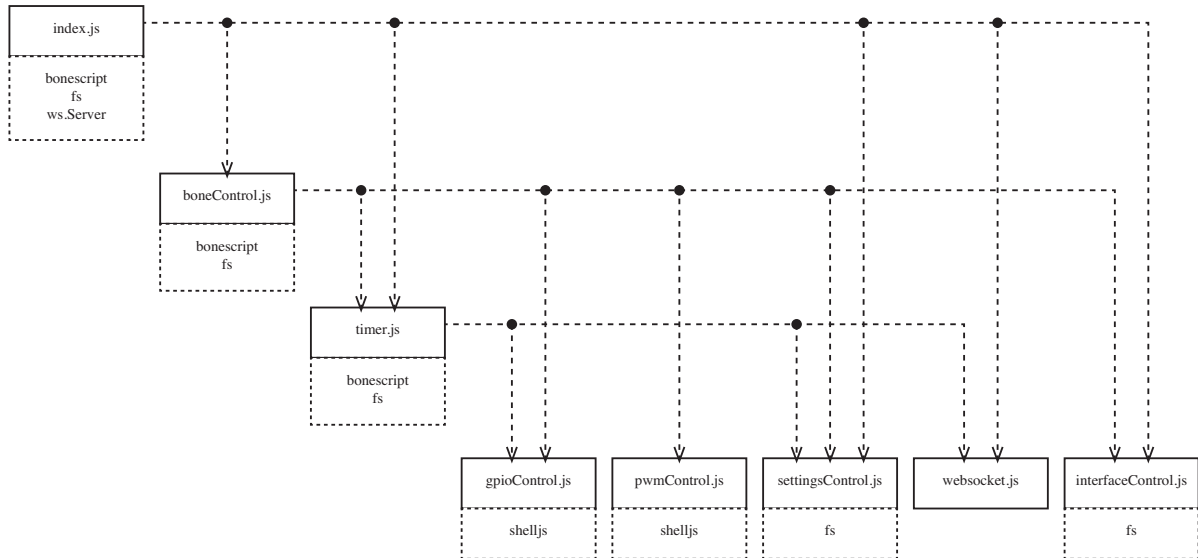


Abbildung 4.7.: Abhängigkeiten der Module des WebSocket Server

beiden Dateien handelt es sich um JavaScript Object Notation (JSON)-Dateien nach rfc6455 [7].

Es werden nur Parameter überschrieben, die sowohl in den Default-Einstellungen als auch in den eigenen vorhanden sind. So ist sichergestellt, dass keine Parameter versehentlich ins System gelangen, die nicht dafür vorgesehen sind. Desweiteren ist es möglich, nur die Einstellungen einzutragen, die geändert werden sollen.

interfaceControl.js

Dieses Modul generiert bei Bedarf eine Liste der verfügbaren Pins und deren Typen Sowie deren letztmalige Interface-Konfiguration bezüglich aktiver und inaktiver Kacheln.

<<module>> interfaceControl
get(pin: String, parameter: String): String set(pin: String, parameter: String) readFromFile() saveToFile()

Zwei Dateien werden zur Generierung des interface-Objektes ausgelesen:

- *whitelist.json* enthält eine Liste der Pins mit Informationen über die Verwendbarkeit. Die Liste muss zur restlichen Software passen und wird vom Programm nicht verändert.
- *interface.json* ist eine String-Version der letzten Interface-Konfiguration. Falls sie existiert, wird ein kombiniertes Objekt generiert um sicherzustellen, dass Infor-

4. Implementierung

mationen über das Interface nicht verloren gehen und eventuell zusätzliche Pins dennoch verfügbar sind.

Wenn die WebSocket-Verbindung geschlossen wird, wird das im Speicher befindliche Interface-Objekt in die Datei `interface.json` geschrieben und kann beim nächsten Aufruf der Seite erneut geladen werden.

websocket.js

Dieses Modul verwaltet die eigentliche WebSocket-Verbindung. Essenziell ist die Funktion `write()`. Hierüber können die restlichen Module direkt auf den Socket schreiben. Dabei wird intern geprüft ob der Socket noch offen ist. Wenn eine neue Verbindung hergestellt wurde, wird diese sofort wieder beschrieben.

<<module>> websocket
setConnected(state: Boolean) setSocket(socket: ws.WebSocket) write(data: JSON)

4.4.2. Controller-Module

Ein weiterer Teil des WebSocket Servers besteht aus einer Reihe von Controller-Modulen, die verschiedene Steuerungen übernehmen.

boneControl.js

Dieses Modul stellt den Kern der Hardware-Ansteuerung dar. Eingehende WebSocket Requests werden hier über `type` identifiziert und abgearbeitet. Hierbei wird über eine Switch-Case-Anweisung entschieden was zu tun ist. Requests mit unbekanntem Typ werden mit einer Fehlermeldung im `response`-Objekt an das Interface zurückgesendet (Abb. 4.8). Die API der `bonescript` Library ist in dieser Liste weitgehend abgebildet. Daneben werden hier auch sämtliche Parameter für das Interface zusammengestellt und an den Browser gesendet.

<<module>> boneControl
handleRequest(request: JSON): JSON

timer.js

Mit Hilfe dieses Moduls werden digitale und analoge Inputs verwaltet. Über eine Switch-Case-Anweisung wird geprüft ob es sich um einen analogen oder digitalen Input handelt und dann mit Hilfe von `setInterval()` ein Timer gestartet, der zyklisch eine anonyme Callback-Funktion aufruft. Das zeitliche Interval wird dem Settings-Modul ent-

<<module>> timer
addTimer(type: String, pin: String) deleteTimer(pin: String) isRunning(pin: String): Boolean

4. Implementierung

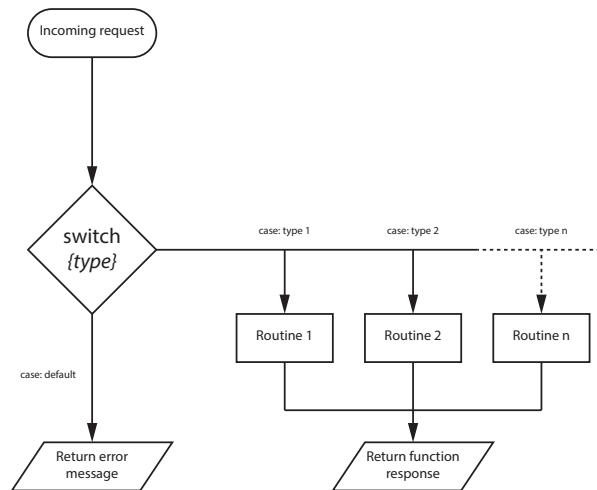


Abbildung 4.8.: Schematische Arbeitsweise des Response Handlers

nommen. Die Funktionen senden dann die Ergebnisse selbstständig an die Webseite. Fehlende Dateien, Links oder Ordner werden ebenfalls automatisch erstellt. Um den Timer wieder beenden zu können, wird die Funktion zusätzlich in einem lokalen Timer-Objekt registriert.

Digital Input Um unnötiges Datenvolumen zu vermeiden prüft diese Funktion zunächst ob sich der Pin-Status gegenüber der letzten Abfrage geändert hat. Nur wenn sich der Wert unterscheidet, wird er an das Interface gesendet (Abb. 4.9).

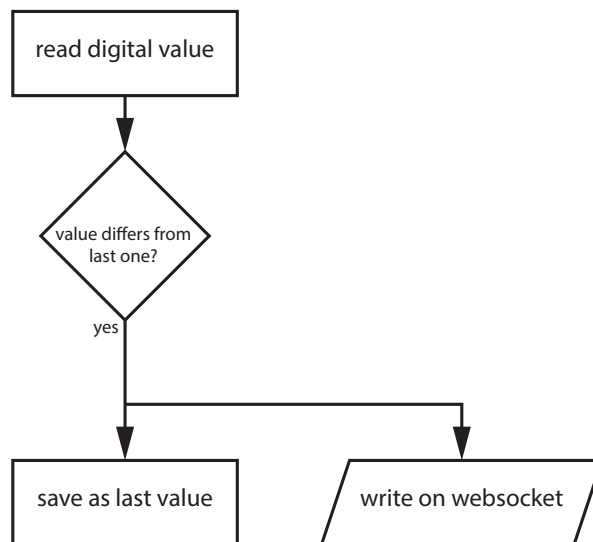


Abbildung 4.9.: Anonyme *digitalRead*-Funktion

Analog Input Die Funktion für den analogen Input produziert einen kontinuierlichen Datenstrom um ein fließendes Diagramm auf der Webseite zu realisieren und speichert diese Daten parallel in einer standardkonformen CSV-Datei [10] (Abb. 4.10).

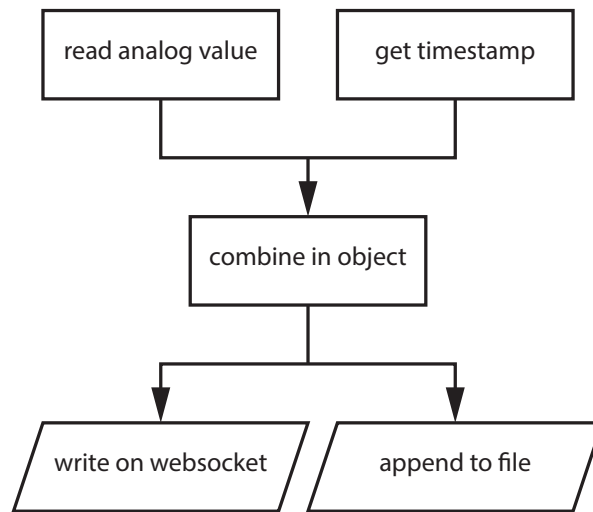


Abbildung 4.10.: Anonyme *analogRead*-Funktion

4.4.3. Bypass-Module

Die Bonescript-Bibliothek weist in einigen Fällen Fehler oder fehlende Features auf. Intern arbeitet die Bibliothek mit Device Tree Overlays, wobei es nicht möglich ist diese wieder zu entladen. Dies ist vor allem wichtig, wenn PWM-Ausgänge bockiert wurden um einen zweiten synchronen Ausgang zu erhalten. Um diese Funktionalität zu implementieren wurden zwei Bypass-Module geschrieben, die das digital I/O- und das PWM-Management übernehmen.

gpioControl.js

Dieses Modul steuert die digitalen I/O direkt über das Dateisystem. Dazu werden die in der Bonescript-Bibliothek hinterlegten Pin IDs genutzt.

<<module>> gpioControl	
enable(gpio: String): Boolean	
disable(gpio: String): Boolean	
read(gpio: String): JSON	
write(gpio: String, options: JSON): JSON	

Das Modul stellt Funktionen zum aktivieren und deaktivieren der GPIO zur Verfügung, sowie zum Lesen und schreiben der Parameter. Parameter werden in Form eines JSON-Objektes übergeben und sind jeweils optional.

pwmControl.js

Die API dieses Moduls verhält sich analog zu dem der GPIO. Intern werden die Device Tree Overlays der Bonescript-Bibliothek verwendet. Diese werden per Filesystem über den Capemgr geladen.

<<module>> pwmControl
enable(gpio: String) disable(gpio: String) read(gpio: String): JSON write(gpio: String, options: JSON)

4.5. Website

Bei der Implementierung steht die Funktionalität im Fokus. Daher wurde so viel wie möglich mit bereits existierenden Bibliotheken gearbeitet, um Design und Darstellung umzusetzen. Diese Bibliotheken basieren intern auf Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript (JS).

Die Website selbst besteht daher nur aus einem einzigen HTML-Dokument, in dem die Basisstruktur der Seite definiert wird. In der index.html werden auch alle globalen Objekte wie die WebSocket-Verbindung und die Pin-Liste angelegt.

Bei einem Seitenaufruf werden zunächst die benötigten Bibliotheken beim Webserver angefordert und geladen. Das Event *document.ready* stößt dann den Verbindungsaufbau zum WebSocket Server an. Ist die Verbindung hergestellt, wird das Event *websocket.onopen* ausgelöst und die aktuelle Pin-Konfiguration vom Server angefordert.

Die weiteren Inhalte werden dynamisch via JavaScript und HTML-Templates generiert.

4.5.1. Skriptdokumente

Die Funktionen für die Webseite sind analog zu den Modulen des WebSocket Servers aufgeteilt. Die Skriptdokumente der Webseite sind jeweils via JSON mit einem Pseudo-Namespace versehen. Funktionen und Objekte sind dabei in einer JSON-Struktur angelegt und können darüber abgerufen werden. Auch Sub-Namespace sind möglich, um beispielsweise Hilfsfunktionen zu beherbergen (Abb. 4.11).

```
var bonescriptCtrl = {};  
  bonescriptCtrl.util = {};
```

Abbildung 4.11.: Pseudo-Namespace in JavaScript

4. Implementierung

Da es innerhalb des Browsers keine modulare Struktur wie im WebSocket Server gibt und um daher keinen falschen Eindruck zu erwecken, sind im Folgenden alle Skriptdokumente alphabetisch zusammengestellt.

- bonescriptCtrl.js** Hier werden alle Funktionen zum Absetzen von Steuerbefehlen an den WebSocket Server definiert. Sowohl Befehle für die Hardware-Steuerung als auch für das Interface sind hinterlegt.
- diagramCtrl.js** Enthält Funktionen, um die Diagramme zu verwalten. Zum einen werden hierüber die Diagramme initialisiert und zum anderen Funktionen zum Hinzufügen neuer Wertepaare und Zurücksetzen des Diagramms. Die Zeichengeschwindigkeit ist fest auf 25Hz gesetzt, um die Systemlast möglichst gering zu halten.
- init.js** Dieses Dokument liefert eine Funktion, *init()*, in der der eigentliche Inhalt der Seite generiert wird. Sobald die Pin-Liste übermittelt wurde, arbeitet eine For-Each-Schleife jeden einzelnen Pin ab. Dabei werden via synchronem Ajax Request die Templates vom Webserver angefordert. Da die Dokumente zunächst im Cache des Browsers verbleiben, entsteht durch den häufigen Aufruf kein erhöhter Netzwerkverkehr. Im Anschluss werden die Listener-Funktionen an Buttons, etc. angehängt.
- responseHandler.js** Der Response Handler ist eine Sammlung von Funktionen, um auf eingehende Messages zu reagieren. Zu jedem im WebSocket Server definierten *type* gibt es hier eine entsprechende Funktion. Funktionsnamen sind identisch, damit die Funktionen direkt aufgerufen werden können.
- util.js** Dieses Dokument enthält einige Utility-Funktionen, zur einfacheren Bedienung von Bootstrap.

websocketCtrl.js Hier liegen die Listener-Funktionen für die WebSocket-Verbindung. Bei einem erfolgreichen Verbindungsaufbau wird der Button im Hautfenster verändert und der Init-Prozess angestoßen. Eingehende Messages werden in JSON-Objekte übersetzt und die entsprechende Funktion aus dem Response Handler aufgerufen.

4.5.2. Bibliotheken

jQuery

jQuery ist ein De-facto-Standard zur Document Object Model (DOM)-Verwaltung. Diese Bibliothek ermöglicht objektähnliche Verwendung von HTML-Elementen und stellt viele Render- und Animationsfunktionen zur Verfügung.

jQuery ist Voraussetzung für alle weiteren Bibliotheken.

Bootstrap & jQuery-UI

Bootstrap ist für die grafische Darstellung verantwortlich und wird dabei durch einzelne Funktionen aus jQuery-UI ergänzt. Hierbei ermöglicht dieses Framework ein Kachelsystem, dass zur Anordnung der Bedienelemente verwendet wird. Dieses System ermöglicht eine dynamische Darstellung, sodass deaktivierte Bedienelemente nicht einfach nur leere Felder hinterlassen. Die aktiven Felder werden vielmehr übersichtlich zusammen gerückt. Weiter generiert Bootstrap mit Hilfe von Themes alle Elemente wie Kopfleiste, Buttons oder Eingabemasken.

Flot Diagrams

Diese Bibliothek dient zum Zeichnen der Diagramme. Im Dokument wird hierfür nur ein Platzhalter eingetragen. Alles weitere ist Aufgabe der Bibliothek. Um neue Punkte zu generieren, wird zunächst das Array, welches alle darzustellenden Wertepaare enthält, aktualisiert und per Funktionsaufruf neu gerendert.

Mustache.js

Mustache.js ist eine JavaScript Implementierung des Mustache-Template-Systems. Mit diesem Werkzeug lassen sich ohne großen Aufwand HTML-Templates schreiben, bei denen Variablen in doppelten geschweiften Klammern – `{{variable}}` – eingesetzt werden. Diese werden in der Laufzeit mittels *Mustache.render()* zu einem gültigen HTML-String verarbeitet. Da sich die einzelnen Kacheln nur in der Pin-Bezeichnung unterscheiden, wurde für jeden der drei Kacheltypen jeweils ein Template entworfen.

5. Erweiterungsmöglichkeiten

Im Folgenden Kapitel werden weitere Möglichkeiten erläutert, durch die der Boneserver noch besser an bestehende Laborumgebungen angepasst werden kann, oder die weiteres Potential des Konzeptes nutzbar machen.

5.1. Zusätzliche Anpassungsoptionen

Das Interface könnte durch zusätzliche Anpassungsoptionen erweitert werden. Eine Möglichkeit besteht darin, die einzelnen Kacheln mit individuellen Bezeichnungen zu versehen.

Der *Interface Controller* kann hierzu dahingehend erweitert werden. Da im WebSocket Server die Pink-Konfiguration ohnehin in einer JSON-Struktur gespeichert wird, können eventuelle Namen einfach hier gespeichert werden. Die Daten werden im Initialisierungsprozess der Seite automatisch an den Browser übertragen.

Im *Init Script* kann die Weiterverarbeitung ähnlich dem Status der Kacheln implementiert werden (Abb. 5.1).

```
for(pin in pins) {  
  if (!pins[pin].active) {  
    responseHandler.toggle({parameters: {pin: pin}, response: false});  
  }  
}
```

Abbildung 5.1.: Fake response message um geladene Kacheln zu verändern

5.2. Höher aufgelöste A/D-Wandlung

Der maximale Takt der digitalen Inputs und der ADC ist durch die kleinste Zeiteinheit in JavaScript (1ms) vorgegeben. Denkbar wäre eine höher aufgelöste Taktung in einer

separaten Software zu implementieren. In C/C++ könnte eine wesentlich höhere Abstrakte erreicht werden. Diese Samples können dann zu mehreren als Antwort übertragen.

Der *Diagram Controller* muss dahingehend erweitert werden, nicht nur einzelne Wertepaare zu verarbeiten (Abb. 5.2), sondern oben beschriebene Frames. Je nach Abstrakte sollte auch über einen entsprechenden Downsampling für die Anzeige nachgedacht werden.

```
diagramCtrl.util.addValue = function(pin, data) {  
    diagramCtrl[pin]['data'].push(data);  
}
```

Abbildung 5.2.: Tupel in ein Diagramm einfügen

5.3. Alternative Steuerungsbibliothek – octalbonescript

Die Bibliothek *octalbonescript*¹, ein bonescript Fork, löst laut API einige Probleme der Bonescript-Bibliothek. Diese könnte als Ersatz für die bisher verwendete Bonescript-Bibliothek verwendet werden. Dadurch würden auch die Module *gpioControl.js* und *pwmControl.js* nicht mehr nötig sein, da diese nur Bypass-Funktionen für fehlerhafte Funktionen enthalten.

5.4. Alternative Hardware

Denkbar wäre eine Portierung des Boneservers auf eine andere Hardware-Plattform. Die Voraussetzung hierfür ist ein Linux-System mit einem Netzwerkzugang und entsprechendem Speicher.

Um auf einer anderen Hardware arbeiten zu können, ist eine passende Bibliothek, wie die Bonescript-Bibliothek BeagleBone, nötig, um die Hardware zu steuern. Das Modul *boneControl.js* die gesamte Steuerung und muss dahingehend erweitert werden, dass die enthaltenen Fälle (Abb. 5.3) passende Funktionen aus ihrer Bibliothek starten.

¹<https://github.com/theoctal/octalbonescript>

```
case 'analogRead':  
  var pin = parameters.pin;  
  
  response = bonescript[request.type](pin);  
  break;
```

Abbildung 5.3.: Beispiel-Case aus dem Modul *boneControl.js*

5.5. Erweiterte Konfiguration in das Interface integrieren

Die erweiterten Konfigurationsmöglichkeiten, die bisher aus Sicherheitsgründen nur über eine Konfigurationsdatei möglich waren, können in das Interface integriert werden. Das Authentifizierungssystem bietet hierfür die Möglichkeit verschiedene Benutzer- und Administrator-Accounts einzurichten (vgl. Abb. 3.3). Dazu muss eine Administrationsseite geschrieben und in einem passenden Verzeichnis abgelegt werden. Die Zugriffsbeschränkungen können dann, wie oben beschrieben, separat für dieses Verzeichnis konfiguriert werden. Externe Laufwerke und USB-Sticks sollten dafür automatisch mit passenden Rechten eingebunden werden und in einem Menü auswählbar sein.

5.6. Weitere GPIO

Neben den bereits verfügbaren GPIO sind noch einige weitere vorhanden, die standardmäßig für den HDMI-Ausgang verwendet werden. Da das System primär per Fernzugriff verwaltet werden soll kann der Anschluss deaktiviert werden. Diese GPIO können dann über die *whitelist.json* freigegeben werden.

6. Zusammenfassung

Im Rahmen dieser Arbeit wurde von mir ein webbasiertes Steuersystem für Messanwendungen entwickelt und vorgestellt. Die Grundidee war, ein vereinfachtes, flexibles System anzubieten, das je nach Anwendungskontext individuell und ohne großen Aufwand angepasst werden kann. Hardware und Programmierung sollten hauptsächlich mit auf dem Markt vorhandenen Mitteln realisiert werden, um Entwicklungsaufwand und Kosten gering zu halten. Auch Rechenleistung und Energie sollten überschaubar bleiben. Von zentraler Bedeutung für das Messsystem allerdings ist, dass es über das Internet, abruf- und steuerbar ist.

Die Wahl der Hardware fiel auf den BeagleBone Black, weil bei ihm das Verhältnis zwischen Rechenleistung, Schnittstellenumfang und Preis gegenüber den Konkurrenzsystemen am günstigsten ist. Gleichzeitig ist der Prozessor eine aktive Produktlinie von Texas Instruments, einem namhaften Hersteller für Mikroprozessoren und es gibt bereits werksseitig eine Software-Bibliothek, die die rudimentäre Hardware-Steuerung übernimmt. Die Wahl des Betriebssystems fiel auf Arch Linux, ein minimales Linux System mit großen Anpassungsmöglichkeiten, da eine Eigenentwicklung den Rahmen dieser Arbeit bei weitem gesprengt hätte und Linux Systeme im Embedded-Bereich zurzeit einen De-Facto-Standard darstellen.

Da die Bibliothek zur Steuerung der Hardware in JavaScript bzw. Node.js implementiert ist, habe ich zunächst versucht, möglichst viel der benötigten Funktionalität via JavaScript umzusetzen. Größtes Problem dabei war, dass die meisten Webserver-Lösungen in Node.js im Zusammenhang mit dem BeagleBone viel zu langsam arbeiten. Zudem ist es sehr aufwändig, die vollständige Funktionalität eines Webserver selbst zu implementieren und das gesamte System hätte mit root-Rechten laufen müssen, was bei Webservern aus Sicherheitsgründen grundsätzlich vermieden werden sollte. Hier wäre weiterer Aufwand durch gewissenhaftes Implementieren von Sicherheitsmechanismen entstanden. Dies war allerdings nicht Teil des Projektes. Aus diesen Gründen habe ich mich für ein

6. Zusammenfassung

mehrstufiges System entschieden, bei dem ein regulärer Webserver Dokumente ausliefert und ein zweites System sich ausschließlich mit der Verwaltung der GPIO beschäftigt.

Dabei stellte sich die Frage, in welcher Weise Steuerbefehle sinnvoll an den BeagleBone übertragen und Antworten bzw. Messdaten zurück erhalten werden könnten. Die Lösung dieses Problems fand sich in WebSockets und JSON. Mit deren Hilfe ist es möglich Daten ohne großen Daten-Overhead strukturiert zu auszutauschen. Vorteilhaft ist ebenfalls, dass es bereits einige Implementierungen in Node.js in Form von Modulen gibt.

Das größte Problem während der Entwicklung bestand darin, dass die Bonescript-Bibliothek durch ihr frühes Entwicklungsstadium noch einige Fehler insbesondere in der Steuerung der Pulsbreitenmodulation aufweist. Lange Versuche diese Fehler zu beheben oder zu umgehen haben dazu geführt, dass ich die Funktionalität selbst in Modulform separat implementiert habe. Ziel ist es aber dennoch, diese Funktionalität wieder in die Bibliothek zu verlagern, sobald die Probleme dort gelöst sind.

Gegen Ende des Projektes bin ich auf eine alternartive Bibliothek gestoßen (*octal-bonescript*), die ihrer Beschreibung nach die API der Bonescript-Bibliothek vollständig implementiert, dabei allerdings wesentliche Fehler behebt. Im Rahmen dieser Arbeit blieb leider keine Zeit diese vielversprechende Alternative eingehend zu prüfen.

Das Ergebnis meiner Arbeit ist ein System, mit dem sich einfache Messumgebungen schnell erstellen lassen und das außer einer Netzwerkverbindung keiner weiteren Technik bedarf. Dabei ist es unerheblich, ob es sich bei dem Netzwerk um ein lokales Netzwerk, ein Intranet im Laborkomplex oder das Internet handelt. Da nur wichtig ist, dass das Netzwerk Betriebssystemweit vorhanden ist, sind auch GSM- oder WLAN-Verbindungen kein Problem.

Diese Arbeit zeigt, dass es mit Hilfe von Webtechnologien wie HTML5 und WebSockets möglich ist Echtzeitmesssysteme zu realisieren, die existierende Hardware verwenden und dadurch wesentlich kostengünstiger sind als spezialisierte Systeme namhafter Hersteller.

Teil II.

Anhang

A. Betriebsanleitung

A.1. Hardware

Dieses Handbuch ist für den **BeagleBone Black Revision A5C** (im Folgenden kurz als BeagleBone bezeichnet) geschrieben und getestet. Sofern nachfolgende oder vorangegangene Revisionen zu dieser kompatibel sind, sollte die Installation aber dennoch problemlos möglich sein.

A.2. Installation

Als Betriebssystem wird Arch Linux ARM verwendet, eine Portierung von Arch Linux für ARM-Prozessoren. Arch Linux ARM stellt auch ein spezielles package repository zur Verfügung.

A.2.1. SD-Karte vorbereiten

Auf der Homepage von Arch Linux ARM gibt es eine Installationsanleitung, die laufend aktualisiert wird. Die folgende Anleitung ist daher im Wesentlichen eine Übersetzung ausgehend von einem Linux als Host-System. Stattdessen kann auch die mitgelieferte Ångström Distribution verwendet werden, die mit dem BeagleBone ausgeliefert wird.

Voraussetzungen sind die Softwarepakete *dosfstools* und *wget* sowie root-Rechte und eine Micro SD-Karte mit mindestens 2GB Speicherkapazität.

1. Finden Sie zunächst heraus, welcher Laufwerkspfad der vorgesehenen SD-Karte entspricht. Meist `/dev/sd[a, b, ...]` oder `/dev/mmcblk[0, 1, ...]`.

ACHTUNG Überprüfen Sie Laufwerkspfade genau, bevor Sie mit der Installation beginnen, da sonst irreparable Schäden am Host-System auftreten können!

A. Betriebsanleitung

2. Starten Sie *fdisk*, um die SD-Karte zu formatieren:

```
fdisk /dev/sdX
```

3. Erstellen Sie eine neue Partitionstabelle und die nötigen Partitionen.

Dazu geben Sie nacheinander die folgenden Kommandos ein (jeweils mit *enter* bestätigen):

Kommando	Funktion
<code>o</code>	Erzeugt eine neue Partitionstabelle
<code>n, p, 1</code>	Erzeugt eine neue (n), primäre (p), erste (1) Partition
<code>enter</code>	Bestätigt den Default-Wert für den ersten Sektor
<code>+64M</code>	+64M als letzten Sektor setzt die Partitionsgröße auf 64MByte
<code>t, e</code>	Ändert den Partitionstyp auf <i>W95 FAT16 (LBA)</i>
<code>a, 1</code>	Setzt das boot flag der ersten Partition (je nach <i>fdisk</i> -Version wird die erste Partition automatisch ausgewählt, da nur eine zur Verfügung steht)
<code>n, p, 2</code>	Erzeugt eine neue (n), primäre (p), zweite (2) Partition
<code>2x enter</code>	Setzt die Default-Werte für den ersten und letzten Sektor der Partition
<code>w</code>	Schreibt Änderungen in die Partitonstabelle

4. Formatieren der ersten Partition:

```
mkfs.vfat -F 16 /dev/sdX1
```

5. Formatieren der zweiten Partition:

```
mkfs.ext4 /dev/sdX2
```

6. Laden Sie den *bootloader tarball* herunter und entpacken Sie ihn auf die erste Partition der SD-Karte:

```
wget http://archlinuxarm.org/os/omap/BeagleBone-bootloader.tar.gz
mkdir boot
mount /dev/sdX1 boot
tar -xvf BeagleBone-bootloader.tar.gz -C boot
sync && umount boot
```

7. Laden Sie den *rootfs tarball* herunter und entpacken Sie ihn auf die zweite Partition der SD-Karte (hierzu müssen Sie als *root* eingeloggt sein, *sudo* reicht in diesem Fall nicht):

```
wget http://archlinuxarm.org/os/ArchLinuxARM-am33x-latest.tar.gz
mkdir root
mount /dev/sdX2 root
tar -xf ArchLinuxARM-am33x-latest.tar.gz -C root
sync && umount root
```

8. Stecken Sie die SD-Karte in den BeagleBone und halten Sie die Taste S2 gedrückt, um von der SD-Karte zu booten, während Sie die Power-Taste (S3) betätigen. Wenn das System gestartet ist, können Sie sich auf der Kommandozeile oder via *ssh* einloggen.

Benutzernahme/Passwort lautet **root/root**.

Aus Sicherheitsgründen sollten Sie nach dem Systemstart als erstes das root-Passwort ändern:

```
passwd root
```

Da man sich, außer zu Wartungszwecken, nicht am System anmelden muss, kann auf die Erstellung eines regulären Benutzers verzichtet werden.

A.2.2. Installation im internen Speicher

Hinweis Der BeagleBone hat zwar eine eingebaute Uhr allerdings keine Batterie. Nach einem Neustart kann es daher passieren, dass die interne Uhr auf den Default-Wert zurück gesetzt wird. Überprüfen Sie mittels *date* die aktuelle Systemzeit und aktualisieren diese gegebenenfalls via *ntpdate -u pool.ntp.org*

1. Um Arch Linux direkt auf der eMMC zu installieren, aktualisieren Sie zunächst das eben gestartete System und installieren die Pakete *wget*, *dosfstools* und *ntp*.

```
pacman -Syu wget dosfstools ntp
```

Das Paket *ntp* stellt hierbei das Programm *ntpdate* zur Verfügung (s. O.).

2. Der interne Speicher ist bereits korrekt partitioniert, folgen Sie daher nur den Schritten 4 bis 7. Die Partionen sind *mmcblk1p1* bzw. *mmcblk1p2* (s. O.).

3. Fahren Sie das System herunter und warten Sie bis alle LEDs erloschen sind.
4. Entfernen Sie die SD-Karte und starten das System erneut.

A.2.3. boneserver installieren

Repository klonen *boneserver* ist via GitHub¹ verfügbar. Führen Sie dazu zunächst ein Systemupdate durch, um alle Pakete auf den neusten Stand zu bringen und installieren Sie das Paket *git*. Anschließend klonen Sie das Repository nach */opt*.

```
pacman -Syu git
git -C /opt clone https://github.com/XMrVertigoX/boneserver.git
```

Installationsskript ausführen Im root-Verzeichnis des Repositories befindet sich ein Skript, welches die weitere Installation übernimmt. Wechseln Sie dazu in das Verzeichnis und führen das Installationsskript aus.

```
cd /opt/boneserver
./install.sh
```

Hierbei werden alle erforderlichen Pakete und Module installiert, die Konfigurationsdateien verlinkt sowie die Daemons installiert und gestartet.

Wenn das Skript fehlerfrei durchgelaufen ist, wird der BeagleBone automatisch neu gestartet und die Installation ist abgeschlossen.

A.3. Betrieb

Für die Verwendung des Webinterface wird eine Netzwerkverbindung zum BeagleBone und ein Webbrowser² mit aktiviertem JavaScript vorausgesetzt.

A.3.1. Netzwerkverbindung herstellen

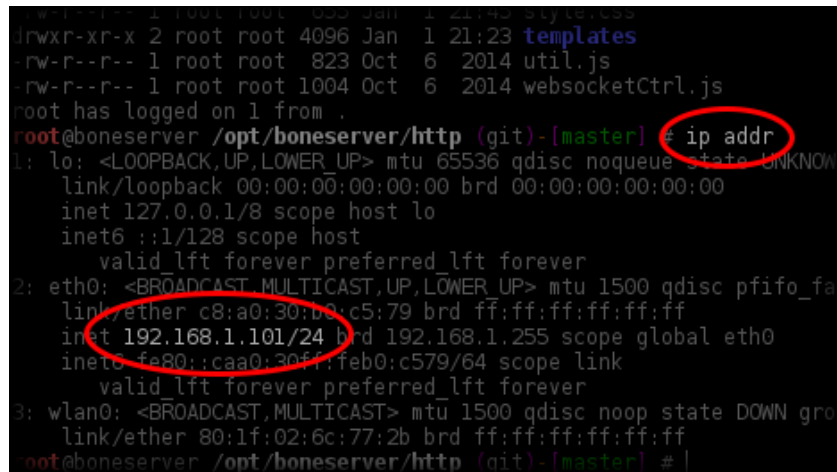
Hinweis: Die Standardkonfiguration des BeagleBone sieht den Betrieb mit einem DHCP-Server vor. Sollte dies nicht gewünscht oder möglich sein, kann über die üblichen Wege

¹<https://github.com/XMrVertigoX>

²Das Webinterface verwendet *jQuery* in der Version 2.1.1, aktuelle webbrowser sollten hier keine Probleme bereiten. Ansonsten kann die Homepage von JQuery konsultiert werden: <http://jquery.com/browser-support/>

eine statische IP eingestellt werden. Anleitungen hierzu findet man im Internet.

Steht ein DNS-Server zur Verfügung, kann der BeagleBone über seinen Hostname erreicht werden, standardmäßig *boneserver*. Ansonsten finden Sie zunächst heraus, welche IP dem BeagleBone zugewiesen wurde. Hierfür kann entweder die Routing-Tabelle des DHCP-Servers konsultiert werden oder in der Kommandozeile via *ip* die aktuelle Adresse der einzelnen Netzwerkadapter abgerufen werden (Abb. A.1).



```
drwxr-xr-x 2 root root 4096 Jan  1 21:23 templates
-rw-r--r-- 1 root root 823 Oct  6 2014 util.js
-rw-r--r-- 1 root root 1004 Oct  6 2014 websocketCtrl.js
root has logged on 1 from .
root@boneserver /opt/boneserver/http (git)-[master] # ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    link/ether c8:a0:30:b0:c5:79 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.101/24 brd 192.168.1.255 scope global eth0
        inet6 fe80::c8a0:30ff:feb0:c579/64 scope link
            valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group
    link/ether 80:1f:02:6c:77:2b brd ff:ff:ff:ff:ff:ff
root@boneserver /opt/boneserver/http (git)-[master] #
```

Abbildung A.1.: IP des BeagleBone abrufen

A.3.2. Webinterface aufrufen

Das Webinterface kann einfach über DNS-Namen oder die IP im Webbrowser aufgerufen werden. Ist die Verbindung hergestellt, wird dies durch einen grünen Haken rechts in der Titelleiste angezeigt (Abb. A.2) und die Steuerelemente werden generiert. Sollte die Verbindung einmal unterbrochen werden, wechselt dieser Haken in ein rotes Kreuz. In diesem Fall kann die Seite neu geladen werden, eventuelle Konfigurationen bleiben erhalten.

Passwortschutz Um unbefugten Zugriff zu verhindern ist das Webinterface passwortgeschützt. Wenn Sie dieses Passwort ändern möchten, generieren Sie zunächst einen neuen Datensatz z. B. mit dem *htdigest Generator Tool*³ und tragen die Zugangsdaten in die Datei *config/lighttpd/lighttpd.user* ein. Die Default Login-Daten sind **admin/AgG7KgW4**

³ <http://jesin.tk/tools/htdigest-generator-tool/>



Abbildung A.2.: Webinterface verbunden

Hinweis Das Webinterface kann immer nur von einem Fenster aus aufgerufen werden. Es kann daher passieren, dass bei einem schnellen Fensterwechsel oder Neuladen der Seite die Verbindung nicht sofort hergestellt wird. In dem Fall einfach ein paar (Milli-)Sekunden warten, bis die Verbindung wieder frei ist.

A.3.3. Bedienelemente

Wie in Abbildung A.3 gezeigt hat die Weboberfläche des boneserver drei Anzeigegruppen:

1 Verbindungsanzeige

zeigt grün, wenn die Steuereinheit verbunden ist und rot, wenn die Verbindung unterbrochen ist (vgl. Abb. A.2).

2 Bedienfelder für digitale I/O, PWM und AIN

Hier findet die tatsächliche Bedienung der GPIO statt. Es gibt drei Sektionen jeweils für digitale I/O, PWM und AIN. Die Bedienung der verschiedenen Kacheln wird weiter unten beschrieben.

3 Anzeigenschalter für die einzelnen Pins

Hier können einzelne Kacheln ein- bzw. ausgeblendet werden, um die Oberfläche übersichtlicher zu gestalten und an die Arbeitsumgebung anzupassen. Diese Funktion dient ausschließlich der Übersicht, eine ausgeblendete Kachel bleibt weiterhin

aktiv und kann jederzeit wieder eingeblendet werden. Diese Einstellungen bleiben auch nach einem Neustart erhalten.

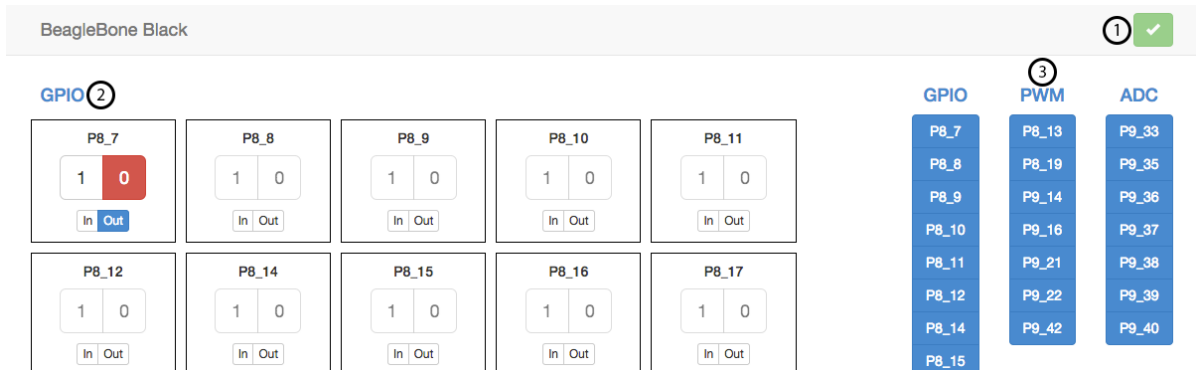


Abbildung A.3.: Weboberfläche

Digitale In- und Outputs

Die Konfigurationskachel (Abb. A.4) für digitale I/O besteht aus zwei Schaltern: Betriebsrichtung und logic level.

Betriebsrichtung Jeder digitale I/O kann entweder als Input oder als Output konfiguriert werden. Dazu kann über den Wahlschalter **In/Out** jederzeit das Gewünschte ausgewählt werden.

logic level Wenn der GPIO als Output konfiguriert ist, kann hier mittels der beiden Schaltflächen, **1** und **0**, ein logisches high und ein logisches low eingestellt werden. Ist der GPIO als Input konfiguriert, ist diese Schaltfläche deaktiviert und zeigt stattdessen den Status der Leitung an. Die GPIO sind mit einem internen Pulldown-Widerstand beschaltet.

Pulsbreitenmodulation (PWM)

Mit Hilfe der PWM-Kacheln (Abb. A.5) werden die GPIO konfiguriert, über die eine Pulsbreitenmodulation möglich ist.

Der BeagleBone stellt insgesamt sieben PWM-Ausgänge mit insgesamt vier Timern zur Verfügung. Dabei teilen sich jeweils die Pinne P8_13/19, P9_14/16 und P9_21/22

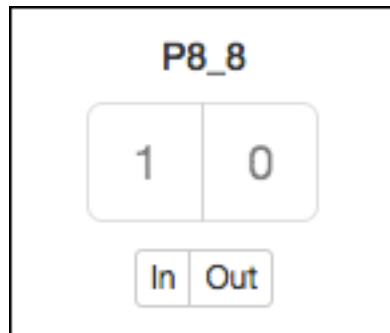


Abbildung A.4.: Deaktivierte GPIO-Kachel

einen Timer. P9_42 hat einen exklusiven Timer. Die Ausgänge mit einem gemeinsamen Timer haben dementsprechend immer dieselbe Frequenz und laufen absolut synchron.

Über die Buttons **Enable** und **Disable** kann der jeweilige Ausgang aktiviert bzw. deaktiviert werden. Wenn ein PWM-Ausgang deaktiviert wird, werden alle Einstellungen bezüglich Frequenz und Pulsbreite gelöscht!

Periodendauer Über das Eingabefeld **Period** wird die Periodendauer in Nanosekunden (ns) eingestellt. Kleinster Wert ist hier 1 ns ($\hat{=}$ 1GHz) und der größte 10^9 ns (= 1s $\hat{=}$ 1Hz).

Pulsbreite Über das Eingabefeld **Duty** wird die Pulsbreite zwischen 0 und 1 eingestellt. Hier wird intern ebenfalls mit Nanosekunden gearbeitet, daher kann der tatsächliche Wert zusätzliche Nachkommastellen bekommen.

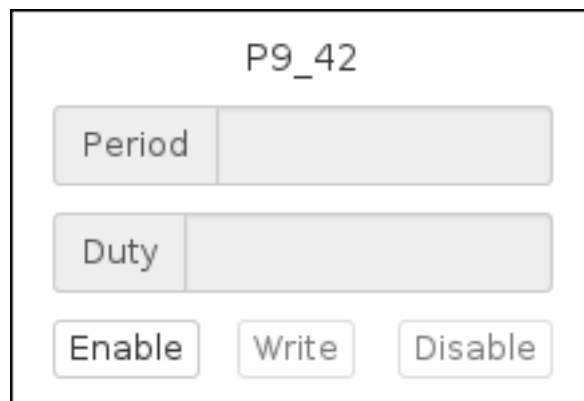


Abbildung A.5.: Deaktivierte PWM-Kachel

Mit **Write** werden die Parameter abgesendet.

Hinweis Der Linux Kernel arbeitet intern mit ganzen Nanosekunden, daher ist die Genauigkeit der Pulsbreite von der Höhe der Periodendauer anhängig.

Bug: Wenn beide Ausgänge eines PWM-Generators aktiviert sind, lässt sich die Frequenz bei keinem der beiden ändern. Wenn bei einem der beiden PWMs die Frequenz zunächst geändert wurde, kann der zweite Ausgang nicht verwendet werden. Dies ist ein Problem der im Hintergrund verwendeten device tree overlays und wird in nachfolgenden Versionen der Bibliothek behoben.

Analoge Inputs

Mit diesen Kacheln (Abb. A.6) werden die Analog/Digital-Converter gesteuert und die Eingangswerte in einem Echtzeitdiagramm angezeigt.

Mit den Tasten **Start** und **Stop** wird die Aufzeichnung gestartet bzw. gestoppt. Parallel zur Anzeige werden die Messdaten aufgezeichnet. Über **Download** können sie als CSV-Datei heruntergeladen werden.

Die Taste **Delete** löscht die zu diesem Eingang gespeicherte Messreihe um Speicherplatz frei zu machen.

ACHTUNG Laden sie Messreihen immer herunter, bevor Sie sie löschen, die Messdaten sind sonst unwiederbringlich verloren!

A.3.4. Erweiterte Einstellungen

Zusätzlich gibt es die Möglichkeit, über die Datei *settings.json* im root-Verzeichnis des boneserver weitere Einstellungen vorzunehmen. Ist keine solche Datei vorhanden, wird die mitgelieferte *settings-default.json* verwendet. Die Parameter überschreiben die Default-Werte, es müssen daher nur abweichende Werte eingetragen werden.

Die Datei ist eine JSON⁴-Datei in der folgende Parameter eingestellt werden können:

⁴JavaScript Object Notation (JSON) ist in der RFC 7159 beschrieben

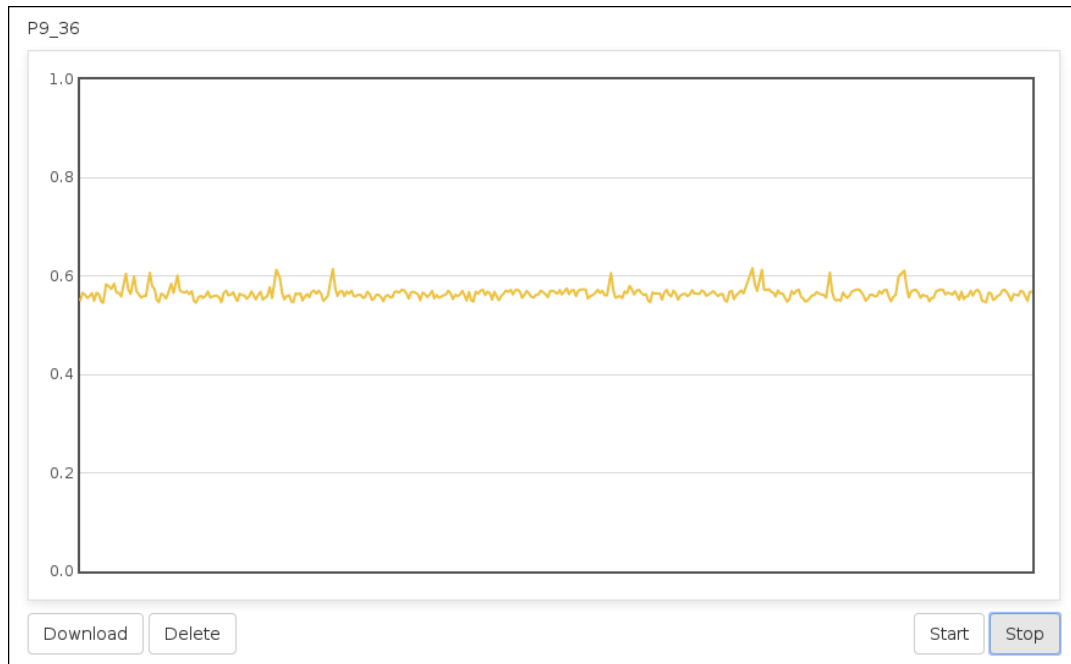


Abbildung A.6.: ADC-Kachel

host	IP des WebSocket Servers Dieser Wert sollte nicht verändert werden, da sonst der WebSocket Server möglicherweise nicht mehr über das Webinterface erreichbar ist. <i>default: localhost</i>
port	Netzwerk-Port des WebSocket Servers Dieser Wert sollte nicht verändert werden, da der WebSocket Server sonst nicht mehr über das Webinterface erreichbar ist. <i>default: 8081</i>
gpioSampleRate	Abtastrate der digitalen Inputs in Millisekunden Angegeben wird die Zeit zwischen den Abfragen. Dieser Wert kann erhöht werden um die System- und Netzwerklast zu verringern. <i>default: 100</i>

adcSampleRate	Abtastrate der analogen Inputs in Millisekunden Angegeben wird die Zeit zwischen den Abfragen. Dieser Wert kann erhöht werden, um die System- und Netzwerklast sowie um die Menge der erhobenen Messwerte zu verringern. Dies ermöglicht bei gleichem Speicherplatz längere Messreihen. <i>default: 10</i>
dataLocation	Speicherpfad für die Messdaten der ADC Hier kann ein alternativer Pfad zur Speicherung der Messdaten eingetragen werden. Es können auch externe Speicherorte wie z. B. USB-Sticks, USB-Festplatten oder Netzlaufwerke verwendet werden. <i>default: ./data</i>

Hinweis Die Datei *settings-default.json* sollte nicht verändert werden, da sonst nicht ohne weiteres ein Update durchgeführt werden kann.

A.4. Wartung

Als Wartungssystem wird die oben erstellte SD-Karte verwendet. Dazu starten Sie den BeagleBone von der SD-Karte und führen die oben beschriebenen Installationsschritte aus.

Die Paketverwaltung unter Arch Linux heißt *pacman*, über sie können neue Pakete aus den Repositories installiert bzw. aktualisiert werden.

Ein kurzer Auszug aus der man-page zu den hier verwendeten Parametern:

Synopsis: `pacman <operation> [options] [targets]`

Parameter	Beschreibung
<i>Operations</i>	
<code>-S, --sync</code>	Synchronize packages. Packages are installed directly from the ftp servers, including all dependencies required to run the packages.
<i>Sync Options</i>	
<code>-c, --clean</code>	Remove packages that are no longer installed from the cache as well as currently unused sync databases to free up disk space.

- u, --sysupgrade Upgrades all packages that are out of date.
- y, --refresh Download a fresh copy of the master package list from the server(s) [...]. This should typically be used each time you use -u or --sysupgrade.

A.4.1. Backup

Da der interne Speicher des BeagleBone „nur“ 2GB beträgt, kann ohne größerem Zeitaufwand ein komplettes Speicherabbild erstellt werden. Dies hat den Vorteil, dass es beim Einspielen von Backups keine Kompatibilitätsprobleme auftreten können.

Im Ordner „scripts“ sind zwei shell-Skripte, die diesen Vorgang vereinfachen: *backup.sh* und *restore.sh*. Dabei wird das Image automatisch mit *gzip* komprimiert, um Speicherplatz zu sparen. Das restore-Skript verwendet diese Dateien, um das Speicherabbild wieder auf den BeagleBone zu kopieren.

Sollte die SD-Karte nicht genügend Speicherplatz zur Verfügung stellen, kann ein USB-Stick verwendet werden. Dazu einfach das USB-Laufwerk einhängen⁵ und das entsprechende Verzeichnis als Zielverzeichnis angeben.

```
backup.sh [Zielverzeichnis]
```

Das Zielverzeichnis ist dabei optional. Wenn kein Parameter übergeben wird, erstellt das Skript automatisch eine Datei in der Form *backup-[timestamp].img.gz* im aktuellen Verzeichnis.

```
restore.sh <Quelldatei>
```

Die Quelldatei ist hier Voraussetzung.

Hinweis Die Skripte verwenden intern *dd*, um eine bitweise Kopie der eMMC des BeagleBone anzufertigen. Zudem ist die Quelle bzw. das Ziel immer */dev/mmcblk1*. Daher sollten diese Skripte nur von der SD-Karte aus verwendet werden.

A.4.2. System aktualisieren

Arch Linux verwendet die Rolling-Release-Technik, ein System, bei dem es keine großen Upgrades des gesamten Betriebssystems gibt, sondern die Softwarepakete einzeln laufend aktualisiert werden.

⁵Anleitungen hierzu gibt es in ausreichender Zahl im Internet

Trotz umfangreicher Tests der Pakete kann es zu Inkompatibilitäten kommen. Dies ist wahrscheinlicher, je mehr Pakete gleichzeitig aktualisiert werden. Daher sollte, gerade wenn das System nur selten aktualisiert wird, vorher ein vollständiges Backup angelegt werden (s. o.).

Das System kann jederzeit via *pacman* aktualisiert werden:

```
pacman -Syu
```

A.4.3. boneserver aktualisieren

Um die boneserver-Software zu aktualisieren, aktualisieren Sie zunächst Ihre Kopie des git Repositories und führen das Installationsskript erneut aus. Pakete, die bereits installiert sind, werden dabei nicht erneut installiert.

```
cd /opt/boneserver  
git pull  
./install.sh
```

A.4.4. System bereinigen

pacman speichert bei jeder Aktualisierung die alten Pakete, um jederzeit auf frühere Versionen zurückgreifen zu können. Je nach Häufigkeit der Aktualisierung und gemessen an der Kapazität der eMMC, kann der Speicher schnell knapp werden. Daher können alte Pakete via *pacman* in zwei Stufen gelöscht werden:

```
pacman -Sc
```

Löscht alle Paketversionen nicht mehr installierter Pakete und

```
pacman -Scc
```

löscht sämtliche nicht verwendeten Pakete.

B. Tabellen

Paketname	Kurzbeschreibung
base-devel	Enthält die benötigten Entwicklertools
python2	Python in der Version 2. Wird für den Node Package Manager benötigt.
lighttpd	Der Webserver (siehe Abschnitt 3.1.2).
vsftpd	Ein FTP-Server. Wird verwendet um Messdaten ohne Webinterface automatisch abzurufen (siehe Abschnitt 3.1.3).
linux-headers-am33x-legacy	Linux Header Files. Werden benötigt um Device Tree Overlays zu kompilieren.
nodejs	JavaScript Engine (Siehe Abschnitt 2.3.2).
wget	Freies Kommandozeilenprogram um Dateien aus dem Internet herunterladen zu können.
zsh	Alternative Kommandozeile (Z-Shell). Hohe Anpassungsmöglichkeiten.
grml-zsh-config	Konfigurationsdateien für zsh.
wpa_supplicant	Unterstützung für WPA-Verschlüsselte WLAN.
pv	Pipe Viewer. Wird von den Skripten <i>backup.sh</i> und <i>restpre.sh</i> verwendet um Speicherabbilder des BeagleBone zu erstellen.
dtc-git-patched-20130410-1	Device Tree Compiler mit <i>dynamic link</i> -Patch. Wird von der bonescript library benötigt.
haproxy-1.5.3-1	Der Proxy-Server (siehe Abschnitt 3.1.1).

Tabelle B.1.: Liste der benötigten Zusatzpakete

C. Literaturverzeichnis

- [1] BASSI, Alessandro ; EUROPE, Hitachi ; HORN, Geir: Internet of Things in 2020 - A Roadmap for the Future / INFOS D.4 Networked Enterprise / RFID INFOS G.2 Micro & Nanosystems. 2008. – Workshop Report
- [2] WEISER, Mark: The Computer for the 21st Century. In: *Scientific American* (1991), Nr. 265, S. 94–104
- [3] STILLER, Andreas: Die ARM-Story. In: *c't - magazin für computertechnik* (2002), Nr. 2, S. 70
- [4] NETCRAFT (Hrsg.). NETCRAFT LTD.: October 2014 Web Server Survey / Netcraft Ltd. <http://news.netcraft.com/archives/category/web-server-survey>, October 2014. – Report
- [5] KRIEG, Michael: *Lighttpd - kurz & gut*. O'Reilly Verlag, 2009. – ISBN 978-3-89721-549-8
- [6] SPRINGER, Sebastian: *Node.js - Das umfassende Handbuch*. Galileo Press, 2013. – ISBN 978-3-8362-2119-1
- [7] FETTE, I. ; MELNIKOV, A.: The WebSocket Protocol / Internet Engineering Task Force. <https://tools.ietf.org/html/rfc6455>, December 2011. – Standards Track
- [8] KÜHNAST, Charly: Passthrough und Offloading: HTTPS balancieren mit HA-Proxy 1.5. In: *ADMIN Magazin* (2014), Nr. 3, S. 32–34
- [9] FIELDING, R. (Hrsg.) ; RESCHKE, J. (Hrsg.). INTERNET ENGINEERING TASK FORCE: Hypertext Transfer Protocol (HTTP/1.1): Authentication / Internet Engineering Task Force. <http://tools.ietf.org/html/rfc7235>, June 2014. – Standards Track

- [10] SHAFRANOVICH, Y.: Common Format and MIME Type for Comma-Separated Values (CSV) Files / Network Working Group. <http://tools.ietf.org/html/rfc4180>, October 2005. – Informational

Eidesstattliche Erklärung

Ich versichere hiermit, die vorgelegte Arbeit in dem gemeldeten Zeitraum ohne fremde Hilfe verfasst und mich keiner anderen als der angegebenen Hilfsmittel und Quellen bedient zu haben.

Köln, den 20. November 2014

(Caspar Friedrich)