



Bachelorarbeit

Entwicklung eines vollautomatisierten
Embedded-Linux-Systems zur Ansteuerung und
Auswertung eines Langzeittests

zur Erlangung des akademischen Grades

Bachelor of Engineering

von

Tim Nieter

05. Januar 2015 – 16. März 2015

Erstbetreuer: Prof. Dr. F. Bauernöppel

Zweitbetreuer: Dipl.-Ing. Gunnar Schoel

Eingereicht am: 16. März 2015

Eidesstattliche Erklärung

Tim Nieter
Rudower Straße 95
12351 Berlin

Hiermit versichere ich, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Berlin, den 16. März 2015

Tim Nieter

(Unterschrift)

Tim Nieter

Thema der Bachelorarbeit

Entwicklung eines vollautomatisierten Embedded-Linux-Systems zur Ansteuerung und Auswertung eines Langzeittests.

Stichworte

BeagleBone Black, Qt, MySQL, Embedded Linux, Degradation

Kurzzusammenfassung

Im Rahmen dieser Arbeit wird ein Embedded-Linux-System unter Verwendung eines Einplatinencomputers zur Ansteuerung und Auswertung eines Langzeittests entwickelt. Zur Erfassung und Ablage von Messdaten wird ein RS232-Protokoll und ein MySQL-Datenbanksystem implementiert. Außerdem wird für die Auswertung und Verwaltung der Daten eine Ethernet Schnittstelle, sowie ein Webinterface geschaffen.

Tim Nieter

Title of the paper

Development of a fully automated embedded Linux system for control and evaluation of a long-term test.

Keywords

BeagleBone Black, Qt, MySQL, Embedded Linux, Degradation

Abstract

This paper describes the development of an embedded Linux system for control and evaluation of a long-term test by using a single-board computer. A RS232 communication protocol and a MySQL database system are implemented to capture and store measurement data. In addition, an ethernet and a web interface is provided for the evaluation and management of the data.

Inhaltsverzeichnis

Abbildungsverzeichnis	II
Tabellenverzeichnis	III
Quellcodeverzeichnis	IV
Abkürzungsverzeichnis	V
1. Einleitung	1
1.1. Motivation	1
1.2. Aufbau der Arbeit	1
2. Grundlagen	3
2.1. Degradation	3
2.2. Qt	4
2.2.1. Signale und Slots	4
2.2.2. Entwicklungsumgebung	5
2.3. Datenbank	6
2.4. Embedded Linux System	8
3. Anforderungsanalyse	9
3.1. Szenario	9
3.2. Analyse	10
3.2.1. Mess-Server	10
3.2.2. Mess-Slave	11
3.2.3. PC-Client	13
3.3. Anforderungen	13
4. Design	16
4.1. Hardware	16
4.2. Software	18
4.2.1. Mess-Slave Verwaltung	19
4.2.1.1. Anmeldung	19

4.2.1.2. Messdatenerfassung	19
4.2.2. Statusüberwachung	20
4.2.3. TCP/UDP Server	21
4.2.4. Web Server und WLAN Hotspot	22
4.2.5. Datenbank Server	24
4.2.5.1. Tabellen	24
4.3. RS232 Protokoll	29
4.3.1. Aufbau	29
4.3.2. Befehle und Unterbefehle	33
5. Implementierung	35
5.1. Systemkonfiguration	35
5.2. RS232 Kommunikation	37
5.3. Datenbank	38
5.4. Webinterface	40
5.5. BeagleBone Black Benutzeroberfläche	41
6. Testen und Validieren	43
6.1. Speicherverwaltung	43
6.2. Fehlerfälle	44
6.3. Testaufbau	44
6.4. Grenzen	45
7. Zusammenfassung und Ausblick	47
7.1. Zusammenfassung	47
7.2. Ausblick	48
Literaturverzeichnis	i
Anhang	iii
A. Hardware	iii
A.1. BeagleBone Black	iii
A.2. Mess-Slave	iii
B. Inhalt der beigefügten CD-ROM	vii

Abbildungsverzeichnis

2.1.1.	Aufgezeichnete Degradation	4
2.2.1.	QtCreator Version 2.0.1 (Quelle: [Wik])	6
2.4.1.	Links:Kommunikation zwischen Applikation und Hardware. Rechts: Beispiel Kommunikation zwischen Applikation und UART über den Kernel.	8
3.1.1.	Szenario	9
3.2.1.	BeagleBone Black	10
3.2.2.	Mess-Slave Oberseite	11
3.2.3.	Mess-Slave Unterseite	11
3.2.4.	Messintervalle	12
3.2.5.	Pulsperioden/Pulsbreiten-Paar	13
4.1.1.	Übersicht Hardware	16
4.1.2.	RS232 Cape	17
4.1.3.	RTC Cape	17
4.1.4.	LCD Cape	17
4.1.5.	Gestapelte Capes	18
4.2.1.	Übersicht Software	18
4.2.2.	Mess-Slave Anmeldung	19
4.2.3.	RS232 Tunnel	22
4.2.4.	Entity Relationship Modell	24
5.2.1.	RS232 Kommunikation: Leseanfrage	38
5.4.1.	Webinterface	40
5.5.1.	Mess-Server GUI: Status Tab	41
5.5.2.	Mess-Server GUI: Events Tab	41
5.5.3.	Mess-Server GUI: Boards Tab	42
5.5.4.	Mess-Server GUI: Debug Tab	42
6.3.1.	Testaufbau	44
A.1.1.	BeagleBone Black	iii

Tabellenverzeichnis

3.3.1. Beispiel: Intervalle	14
3.3.2. Anforderungen	15
4.2.1. Parameter der Messintervalle	20
4.2.2. Ethernet Befehlsliste	22
4.2.3. Tabelle D_Board	25
4.2.4. Tabelle Setting	26
4.2.5. Tabelle DUT	26
4.2.6. Tabelle Pattern	27
4.2.7. Tabelle PulsePattern	27
4.2.8. Tabelle Measurement	28
4.3.1. Übertragungsrahmen	29
4.3.2. 1. Byte: Adresse & Read/Write	30
4.3.3. Read/Write	30
4.3.4. 2. Byte: Rahmenlänge	30
4.3.5. 3. Byte: Befehl	31
4.3.6. 4. Byte: Unterbefehl	31
4.3.7. 5. Byte und folgend: Nutzdaten	31
4.3.8. Letztes Byte: Checksumme	32
4.3.9. RS232 Befehlsliste	33

Quellcodeverzeichnis

2.1. Qt <i>connect</i> -Statement	4
2.2. Qt <i>emit</i> -Statement	5
5.1. Systemd Service	36
5.2. Systemstart Script	36
5.3. Logrotate	37
5.4. MySQL Zugriffsrechte	38
5.5. MySQL Index	39
5.6. onBoardDelete Trigger	39
5.7. DHCP/DNS Konfiguration	40

Abkürzungsverzeichnis

SQL	Structured Query Language	6
DB	Datenbank	6
DBMS	Datenbankmanagementsystem.....	6
DBS	Datenbanksystem	6
ERM	Entity-Relationship-Modell	24
DUT	Device Under Test	9
GUI	Graphical User Interface	4
IDE	Integrated Developement Environment	5
SDK	Software Developement Kit	5
LED	Light-Emitting-Diode	3
UART	Universal Asynchronous Receiver Transmitter	17
RTC	Real Time Clock.....	17
CGI	Common Gateway Interface.....	23
GCC	GNU-C-Compiler.....	35
GPIO	General Purpose Input/Output.....	36
SSH	Secure Shell.....	35
UDP	User Datagram Protocol	21
TCP	Transmission Control Protocol	21

1. Einleitung

Zur Einleitung dieser Arbeit wird in diesem Kapitel zunächst auf die Motivationen, die zur Erstellung dieser Arbeit geführt haben, eingegangen. Anschließend wird ein Überblick über den Aufbau der Arbeit vermittelt.

1.1. Motivation

Wann immer Systeme in der realen Welt eingesetzt werden, sollen sie so zuverlässig, fehlerfrei und vorhersehbar wie möglich arbeiten. Gerade wenn diese Systeme an kritischen Punkten zum Einsatz kommen und über lange Zeiträume agieren, sind diese Eigenschaften besonders wichtig. Um diesen Anforderungen gerecht zu werden, müssen alle Bauelemente eines solchen Systems diese Vorgaben erfüllen, denn die Zuverlässigkeit ist immer abhängig vom schwächsten Glied. Bei der Entwicklung eines Systems ist es somit entscheidend, alle Bauelemente vorher anhand der gegebenen Umstände zu qualifizieren. Dafür müssen die Grenzen ausgelotet werden, in denen sie zuverlässig betrieben werden können.

Eine dieser Grenzen ist die altersbedingte Änderung der Betriebsparameter, die sogenannte Degradation. Um das Degradationsverhalten eines Bauelementes bestimmen zu können, muss es über lange Zeiträume unter erschweren Bedingungen betrieben und ausgewertet werden. Nur wenn ein Bauelement ein für die Anwendung akzeptables Degradationsverhalten aufweist, ist es für den Einsatz im Gesamtsystem geeignet.

Da ein hoher Einsatz von Ressourcen nötig ist, um jedes Bauelement individuell zu Prüfen, existieren automatisierte Teststände mit deren Hilfe der Aufwand minimiert werden soll.

1.2. Aufbau der Arbeit

Im Rahmen dieser Arbeit wird ein Embedded-Linux-System zur Ansteuerung und Auswertung eines solchen Langzeittests entwickelt. Dazu wird zunächst in Kapitel 2 auf einige Grundlagen eingegangen. Darunter fallen die Begrifflichkeiten *Embedded-Linux-System* und *Degradation*, sowie das Konzept einer Datenbank und der C++ Klassenbibliothek Qt.

Anschließend wird in Kapitel 3 das Szenario vorgestellt, sowie die Akteure des Systems analysiert und im Anschluss die Anforderungen an das System definiert.

In Kapitel 4 wird das Design der Hard- und Softwarekomponenten erläutert. Außerdem wird ein Übertragungsprotokoll für die RS232 Kommunikation entworfen.

Im Anschluss befasst sich Kapitel 5 mit der Implementierung des Designs in das System. Dabei wird sich vor allem mit der Konfiguration der einzelnen Elemente auseinander gesetzt.

Um die Ergebnisse zu validieren, wird in Kapitel 6 auf einige Tests zur Validierung der Ergebnisse eingegangen.

Zum Abschluss erfolgt in Kapitel 7 eine Zusammenfassung der Arbeit und ein Ausblick für Verbesserungen in der Zukunft.

2. Grundlagen

Bei der Entwicklung des Teststandes kommen verschiedene Systeme und Konzepte zum Einsatz. Dieses Kapitel befasst sich mit der C++ Klassenbibliothek Qt und der Frage, was ist die Degradation oder ein Embedded-Linux-System. Des Weiteren wird auf das Konzept einer Datenbank und die Evaluierung der beiden Datenbanksysteme MySQL und SQLite eingegangen.

2.1. Degradation

Der Hauptgrund für mechanisches Versagen im Lebenszyklus eines Systems oder Bauelementes, liegt an der langsamen Ansammlung von nicht reversiblen Schäden. Dieser Prozess ist bekannt als Degradation (vgl. [ZSG11]).

Die Schwierigkeit liegt in der Feststellung des mechanischen Versagens in messbaren Schritten. So ist die Bestimmung der Lebenszeit über die Zeit bis zum Ausfall zeitaufwändig. Einfacher ist es die Leistung des Bauelementes oder Systems zu erfassen, z.B. die optische Leistung einer Light-Emitting-Diode (LED). Auf diesem Weg kann ein Trend ausgemacht werden.

Um die Länge des Lebenszyklus eines Bauelementes bestimmen zu können, wird die Leistung dessen unter Last über einen langen Zeitraum hinweg aufgezeichnet. Anhand der daraus resultierenden Daten kann die zu erwartende Länge des Lebenszyklus ermittelt werden.

In Abbildung 2.1.1 ist ein Beispiel für erfasste Daten zu sehen. Sie zeigt die relative Abweichung der optischen Leistung über die Zeit von sieben verschiedenen LEDs. Bei den durch die rote und braune Kurve repräsentierten LEDs dürfte nach den hier vorliegenden Messergebnissen nicht davon ausgegangen werden, dass nach 20.000 Stunden mehr als 70% der ursprünglichen Lichtleistung vorhanden ist. Dagegen deuten die grünen Kurven bereits nach 5.000 Stunden auf eine sehr hohe Lebenserwartung der entsprechenden LEDs hin. Diese Aussagen können nur getroffen werden, da über einen sehr langen Zeitraum Messungen in sehr kurzem Abstand erfasst wurden.

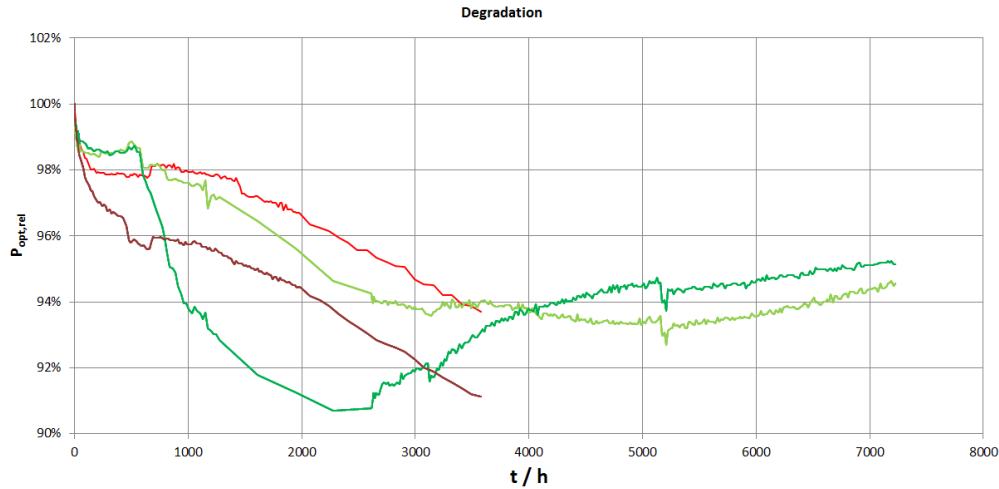


Abb. 2.1.1.: Aufgezeichnete Degradation

2.2. Qt

Qt [QtP] ist eine umfangreiche C++-Klassenbibliothek zur Gestaltung und Entwicklung von Anwendungen. Vor allem bei Applikationen mit grafischen Benutzeroberflächen (englisch: Graphical User Interface (GUI)) ist Qt sehr beliebt.

Zusätzlich bringt Qt eine große Auswahl an Tools und Modulen mit sich, welche die Programmierung erheblich erleichtern (z.B. Netzwerkprogrammierung, Datenbankanbindung, OpenGL, etc.). Ein weiterer Vorteil ist die Plattformunabhängigkeit. So unterstützt Qt aktuell (Version 5.4, 10. Dezember 2014) einen Großteil der aktuellen Betriebssysteme wie Windows, Linux, Android, iOS und einige mehr. Die hohe Kompatibilität wird dadurch gegeben, dass Qt die meisten Systemaufrufe abstrahiert.

2.2.1. Signale und Slots

Die Kommunikation unter den Objekten in Qt erfolgt über Signale und Slots. Dabei sendet ein Objekt ein Signal aus und ein anderes empfängt dieses. Dafür muss zunächst eine Verbindung zwischen den Objekten aufgebaut werden. Das erfolgt mit dem *connect*-Statement.

```
connect(Calculate, SIGNAL(clicked()), this, SLOT(addAB()));
```

Quellcode 2.1: Qt *connect*-Statement

Im Quellcode 2.1 wird ein Beispiel für die Verbindung von zwei Objekten mit dem *connect*-Statement gezeigt. Hier wird das Signal *clicked()* des Objektes *Calculate* mit dem Slot *addAB()* des Objektes *this*, was dem aktuellen Objekt entspricht, verbunden.

```
1 void Calculate :: my_function () {  
2     /*  
3      Do Something  
4     */  
5     emit clicked ();  
6 }
```

Quellcode 2.2: Qt *emit*-Statement

Im Quellcode 2.2 wird das Signal *clicked()* nun mittels *emit*-Statement ausgelöst und der Slot des verbundenen Objektes aktiviert. Dadurch wird die Funktion *addAB()* ausgeführt. Somit ist es möglich beispielsweise die GUI und den Hauptprogrammablauf von einander zu trennen und lediglich über Signale und Slots kommunizieren zu lassen. Beide Programmteile agieren dabei komplett unabhängig von einander.

2.2.2. Entwicklungsumgebung

Als Entwicklungsumgebung (englisch: Integrated Developement Environment (IDE)) dient der Qt Creator (siehe Abbildung 2.2.1), welcher Teil des Software Developement Kit (SDK) von Qt ist und sowohl für Linux, Windows als auch Mac OS X zur Verfügung steht. Er kommt mit einem Debugger, einem integrierten GUI Designer und einem Texteditor, welcher unter anderem Funktionen wie Syntax-Hervorhebung und automatischer Vervollständigung beherrscht.

Dabei kommen gängige Compiler wie MinGW unter Windows zum Einsatz und es besteht die Möglichkeit eigene Toolchains anzulegen.

Eine Toolchain dient zum Übersetzen von Quellcode auf einem Host-System für ein Ziel-System. Sie setzt sich meistens aus einer Kette (englisch: chain) von Werkzeugen (englisch: tool) zusammen, welche nacheinander für die Übersetzung des Quellcodes sorgen. Somit ist es möglich, einen Programmcode auf einem Windowsrechner zu schreiben und anschließend für ein Linux-System zu übersetzen.

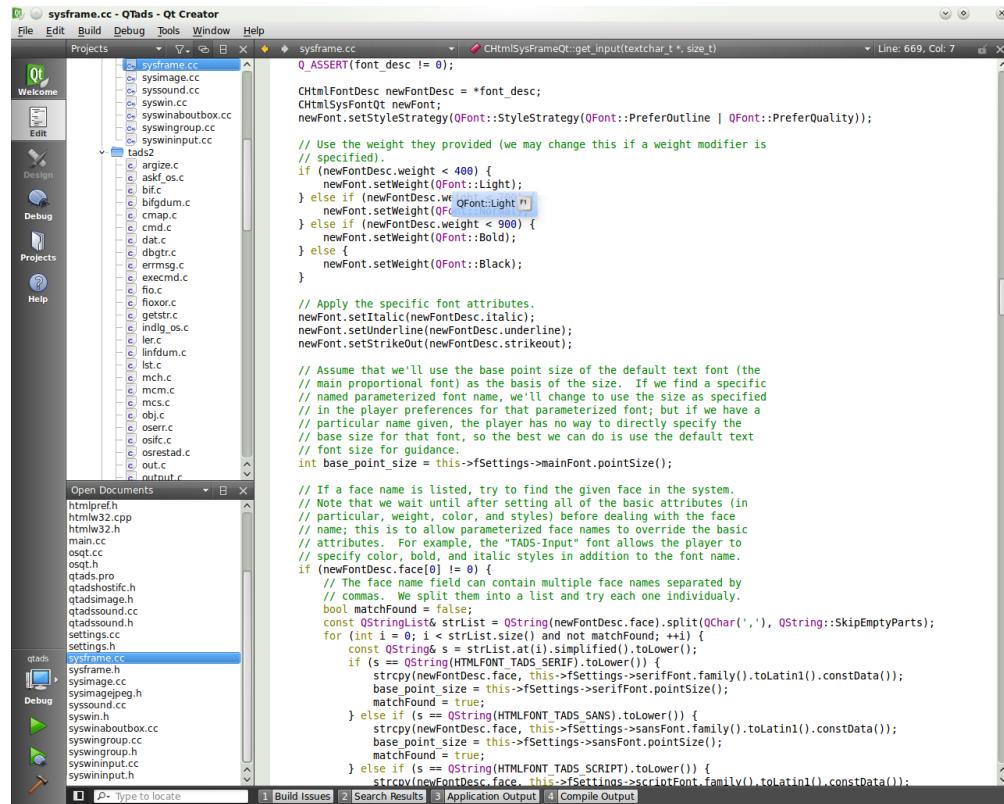


Abb. 2.2.1.: QtCreator Version 2.0.1 (Quelle: [Wik])

2.3. Datenbank

Um große Mengen an Daten effizient verarbeiten zu können, ist der Einsatz eines Datenbankmanagementsystems (DBMS) ratsam (vgl. [SSH10]).

Für das Speichern von Daten und Betriebsparametern wird daher eine Structured Query Language (SQL) Datenbank (DB) benötigt. Bei der Wahl des DBMS, stehen mehrere Systeme zur Auswahl. Dabei kommen SQLite und MySQL aufgrund der kostenlosen Verfügbarkeit in die engere Auswahl. Beide System haben ihre Vor- und Nachteile.

SQLite ist ein SQL DBMS, welches ohne einen Server auskommt und operiert stattdessen in einer einzigen Datei. Alle Informationen wie Tabellenstruktur und Daten sind darin enthalten. Es wird vor allem im eingebetteten Bereich eingesetzt, da kaum Konfigurationen oder Verwaltung notwendig sind. Deshalb eignet es sich ausgezeichnet für sich schnell weiterentwickelnde Applikationen. Aufgrund dieser Eigenschaften existieren allerdings auch Nachteile. So unterstützt SQLite nur eingeschränkt mehrere Nutzer, die gleichzeitig auf die Datenbank zugreifen. Da das gesamte Datenbanksystem (DBS) in einer einzigen Datei zusammengefasst ist, können mehrere zur selben Zeit durchgeführte Schreibzugriffe nicht unterstützt werden. Denn die einzige Zugriffskontrolle und Sicherstellung der Datenintegrität erfolgt durch das Dateisystem des Betriebssystem,

was nicht immer fehlerfrei implementiert ist.

Des Weiteren ist SQLite aufgrund des Ein-Datei-Systems nur eingeschränkt skalierbar. Bei einer größeren Datenmenge oder erhöhten Anzahl an Zugriffen ist es nicht möglich diese Datei auf mehrere Systeme zu separieren, um somit die Last gleichmäßig zu verteilen.

MySQL ist ein weiteres SQL DBMS, welches allerdings auf einer Serverarchitektur beruht. Es ermöglicht die Verwaltung von Nutzern und Rechten. Dadurch ist es ausgezeichnet für den gleichzeitigen Zugriff mehrerer Nutzer geeignet.

Außerdem ist das System gut hinsichtlich Performance und Größe zu skalieren, da die Last auf mehrere Server verteilt werden kann. Zusätzlich bietet MySQL viele weitere Möglichkeiten für Performanceoptimierung, wie z.B. Query-Caching. Beim Query-Caching werden Ergebnisse von SQL-Abfragen (englisch: SQL-Query) zwischengespeichert, so dass sie bei erneuter Abfrage bereits vorliegen.

Jedoch gibt es auch hier Nachteile. So ist die Konfiguration wesentlich schwerer und komplexer. Während bei SQLite kaum Konfigurationen nötig sind, müssen bei MySQL viele Einstellungen an das Host-System angepasst werden. Darunter fallen beispielsweise verschiedene Zwischen speichergrößen und die Zugriffsrechte. Durch die Notwendigkeit eines Servers benötigt MySQL außerdem wesentlich mehr Ressourcen auf dem Host-System.

Für diese Arbeit ist das relationale DBMS MySQL besser geeignet. Auch wenn SQLite einige Vorteile im eingebetteten Bereich besitzt, ist die fehlende Unterstützung von mehreren Nutzern gleichzeitig das Ausschlusskriterium.

2.4. Embedded Linux System

Ein eingebettetes System (englisch: embedded system) ist laut K. Bender (vgl. [Ben05]) ein Rechner, der in ein Gerät oder eine Maschine eingebettet ist und von außen nicht als solcher zu erkennen, sondern lediglich als Träger intelligenter Systemfunktionen sichtbar ist. Des Weiteren übernehme es dabei meist Überwachungs-, Steuerungs- oder Regelungsaufgaben.

Eingebettete Systeme sind oft spezifisch für die gegebene Aufgabe konzipiert. Dadurch ist es möglich die Hard- und Softwarekonfiguration des Systems zur Verminderung der Kosten und Maximierung der Leistung zu optimieren. Zu finden sind sie in den unterschiedlichsten Bereichen. Beispielsweise in mobilen Geräten, industriellen Maschinen, Netzwerkhardware und Verbrauchergeräten.

Ein Embedded Linux (deutsch: eingebettetes Linux) ist ein Betriebssystem, das auf dem Linux-Kernel basiert und in einem eingebetteten System zum Einsatz kommt. K. Yaghmour (vgl. [YMBYG08]) sagt, dass Embedded Linux dabei typischerweise für ein Komplettsystem, bzw. ein Betriebssystem für ein spezifisches eingebettetes Gerät steht. Außerdem verwendet man dabei den normalen Linux-Kernel, der sich lediglich in betriebsbedingten Eigenschaften des Zielsystems unterscheidet.

Das Embedded Linux System setzt sich aus einem Linux-Kernel nutzenden Rechner, dem Embedded Linux Betriebssystem mit für den Zielrechner passender Software und Werkzeugen für die Entwicklung zusammen. Diese Werkzeuge sind eine Entwicklungsumgebung mit Debugger und Cross-Compiler.

Der Vorteil bei der Verwendung des Linux Kernel ist u.a. die Abstraktion der Hardware. Über die Schnittstelle des Kernels kann mit einfachen Systemaufrufen mit der Hardware kommuniziert werden ohne die genaue Hardware zu kennen (siehe Abbildung 2.4.1).

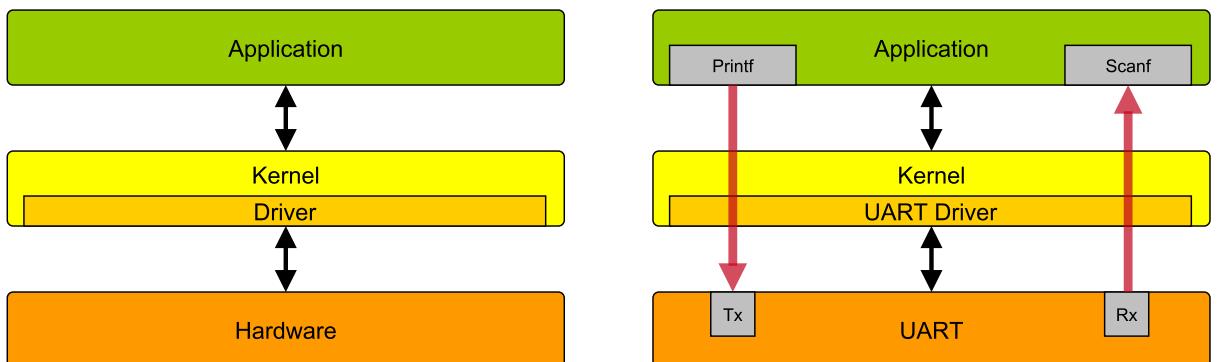


Abb. 2.4.1.: Links: Kommunikation zwischen Applikation und Hardware.

Rechts: Beispiel Kommunikation zwischen Applikation und UART über den Kernel.

3. Anforderungsanalyse

Das Ziel dieser Arbeit ist es, eine Steuereinheit für einen Teststand zu entwickeln, der die Messung der Degradation von optoelektronischen Sendern ermöglicht. In diesem Kapitel wird zunächst das Szenario aus dem sich die Notwendigkeit des Systems ergibt näher beschrieben. Dann erfolgt eine Analyse der beteiligten Akteure und im Anschluss werden die Anforderungen definiert.

3.1. Szenario

Ein Unternehmen stellt verschiedene optoelektronische Sensoren her. Zur Sicherstellung der Zuverlässigkeit der verwendeten LEDs sollen diese mittels eines automatisierten Teststandes bezüglich ihres Degradationsverhaltens qualifiziert werden.

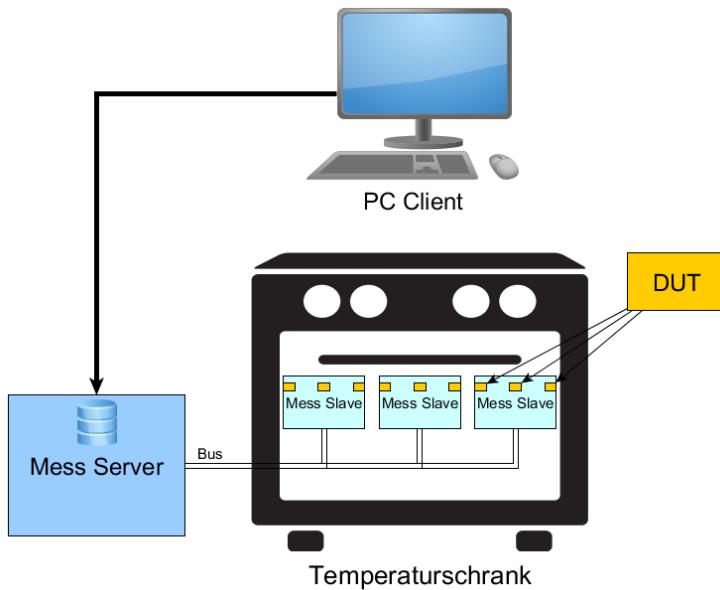


Abb. 3.1.1.: Szenario

Der Teststand hat den in Abbildung 3.1.1 zu sehenden Aufbau. In einem Temperaturschrank befinden sich mehrere Mess-Slaves. An diesen Mess-Slaves sind jeweils 64 Devices Under Test (DUTs) fest angeschlossen, bei denen das Degradationsverhalten aufgezeichnet werden

soll. Dazu befindet sich gegenüber jedes DUT eine Photodiode, die als Empfänger dient und die Auswertung der von den DUTs abgegebenen optischen Leistung ermöglicht.

Die Mess-Slaves sind über einen Bus mit einem außerhalb des Temperaturschrankes befindlichen Mess-Server verbunden, welcher zyklisch Messdaten von den Mess-Slaves abruft und alle anfallenden Daten in einer Datenbank speichert.

Von einem PC-Client kann auf den Mess-Server zugegriffen werden, um die Messdaten zu beziehen und grafisch auszuwerten.

3.2. Analyse

Die Akteure des Systems sind der Mess-Server, Mess-Slave und PC-Client. Im Zuge dieser Arbeit soll der Mess-Server realisiert werden. Wobei die Interaktionsfähigkeit mit den anderen Akteuren sichergestellt werden muss.

3.2.1. Mess-Server

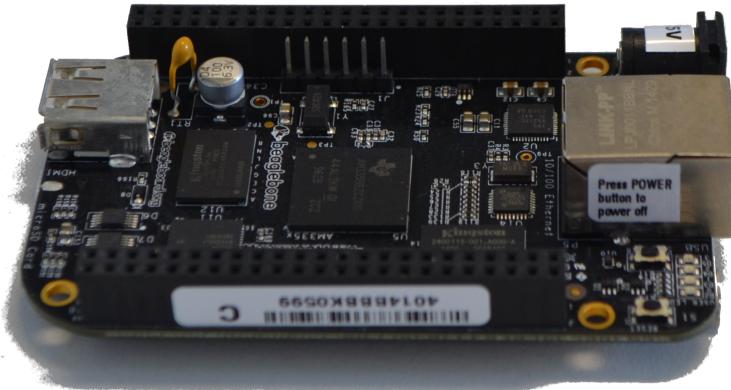


Abb. 3.2.1.: BeagleBone Black

Als Mess-Server wird ein BeagleBone Black von Texas Instruments eingesetzt (siehe Abbildung 3.2.1). Dabei handelt es sich um einen kostengünstigen Einplatinencomputer mit offener Hardware. Dadurch ist es möglich, das BeagleBone Black auf individuelle Anforderungen anzupassen und selbst herzustellen. Auch gibt es eine große Community, die ständig die Entwicklung vorantreibt. Die aktuelle Revision C arbeitet mit einem AM335x 1GHz ARM® Cortex-A8 Prozessor, verfügt über 512MB DDR3 RAM und 4GB 8-bit eMMC internen Flash Speicher. Als Spannungsversorgung dient ein 5V 2A Netzteil. Trotz der Kompaktheit des BeagleBone Black, bietet er ein ausreichendes Maß an Performance.

Auf ihm kommt ein eingebettetes Debian-GNU/Linux Betriebssystem (siehe Abschnitt 2.4) zum Einsatz. Wie auch J. Schröder et al. (vgl. [SGD09]) sagt, ist es dadurch möglich die umfangreichen Linux-Funktionen wie die Paketverwaltung zu nutzen. Es bietet auch den Vorteil, dass eine große Ähnlichkeit zu PC-Distributionen wie Ubuntu besteht und somit die Linux Mechanismen einfach nutzbar sind. So kann das RS232 Interface beispielsweise wie eine normale Datei beschrieben und gelesen werden (siehe Abschnitt 2.4).

Für die Kommunikation mit den anderen Akteuren im System muss ein RS232 Bussystem und eine Ethernet Schnittstelle realisiert werden. Des Weiteren soll ein MySQL Datenbankserver auf dem Mess-Server die effiziente Verwaltung von Daten übernehmen. Zusätzlich soll eine Statusanzeige zur Überwachung des Mess-Servers entwickelt werden.

3.2.2. Mess-Slave

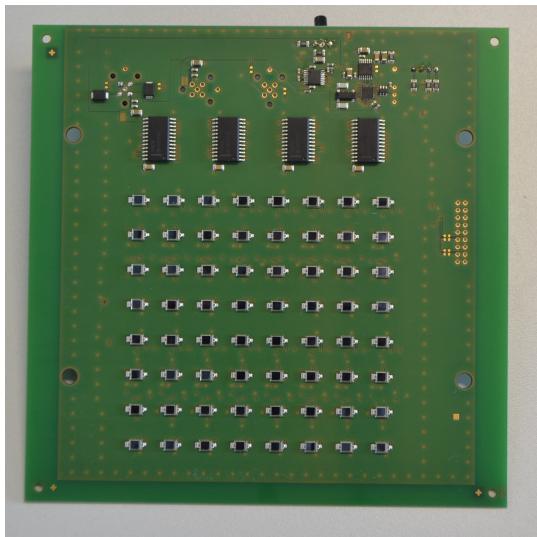


Abb. 3.2.2.: Mess-Slave Oberseite

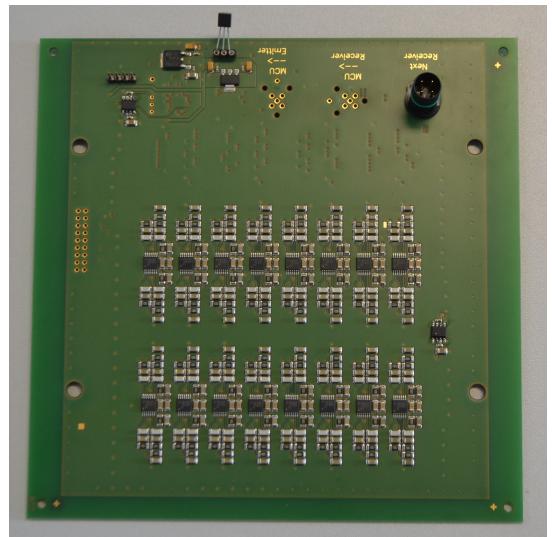


Abb. 3.2.3.: Mess-Slave Unterseite

Ein Mess-Slave (auch: Degradationsboard) ist eine Leiterplatte (siehe Anhang A.2) auf der sich 64 optische Empfänger zur Messung von optischen Signalen befinden (siehe Abbildung 3.2.3 und 3.2.2). Auf jedem Mess-Slave können somit bis zu 64 Prüfobjekte befestigt werden. In zyklischen Abständen werden mittels eines Mikrocontrollers die Messdaten der Prüfobjekte aufgenommen und über eine RS232-Schnittstelle zur Verfügung gestellt.

Für die Konfiguration speichert der Mess-Slave die Parameter in einem persistenten Speicher. Folgende Parameter stehen dabei zur Verfügung:

Name

Für die Identifikation hat jeder Mess-Slave einen Namen der aus maximal 30 Zeichen bestehen kann. Er dient vor allem für die Identifizierung der verschiedenen Mess-Slaves.

RS232 Adresse

Um über die RS232 Schnittstelle ansprechbar zu sein, verfügt jeder Mess-Slave über eine Adresse. Lediglich wenn eine Nachricht diese Adresse beinhaltet, reagiert der Mess-Slave. Die Adresse ist mit 7 Bit aufgelöst und bildet einen Adressraum von 128 Adressen.

Messintervalle

Zur Steuerung der Intervalle, in denen Messwerte vom Mess-Server aufgezeichnet werden sollen, speichert der Mess-Slave diese Informationen. Es ist möglich, Messintervalle für drei Zeiträume zu definieren.

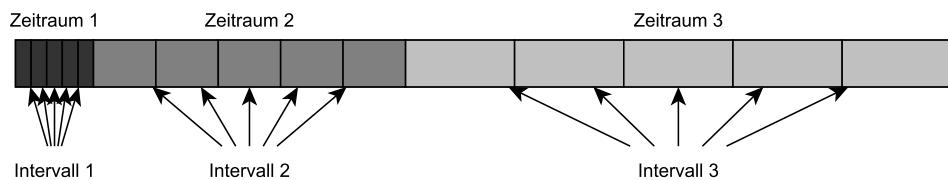


Abb. 3.2.4.: Messintervalle

In Abbildung 3.2.4 ist die Unterteilung der Zeiträume und Messintervalle zu sehen. Jeder definierte Zeitraum hat seine eigenen Messintervall, welches den Abstand zwischen den Messdatenabfragen bestimmt.

Pulsmuster

Der Mess-Slave steuert die Prüfobjekte mit einem Pulsmuster (englisch: pulse pattern) an. Dieses Muster ist konfigurierbar und wird an die unterschiedlichen Prüfobjekte angepasst. Ein Pulsmuster besteht aus bis zu 20 verschiedenen Pulsen, die periodisch wiederholt werden. Ein Puls besteht dabei aus einer Pulsperiode und einer Pulsbreite. In Abbildung 3.2.5 ist ein Pulsperiode/Pulsbreiten-Paar zu sehen.

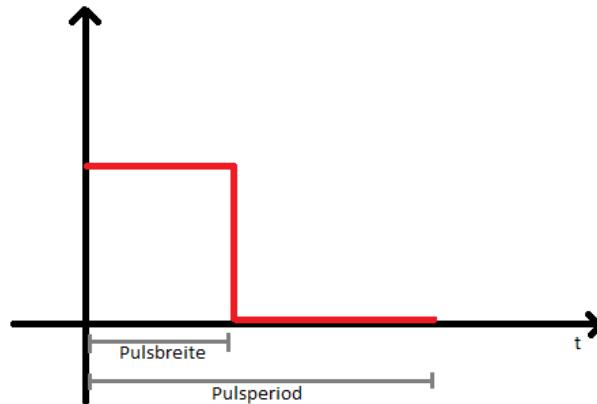


Abb. 3.2.5.: Pulsperioden/Pulsbreiten-Paar

3.2.3. PC-Client

Der PC-Client stellt eine Qt Desktop-Anwendung mit einer Benutzeroberfläche zur Auswertung der Messdaten dar, die nicht im Rahmen dieser Bachelorarbeit entwickelt wird. Über die Ethernet Schnittstelle ruft er die Messdaten von dem Mess-Server ab und bereitet sie zur genaueren Auswertung grafisch auf.

Außerdem werden über die Desktop-Anwendung die Mess-Slaves parametriert. Dafür wird ein Mess-Slave lokal über die RS232 Schnittstelle an den PC-Client angeschlossen. Dies ist notwendig, da einige Parameter in Abhängigkeit von den Prüfobjekten eingestellt werden müssen.

Im laufenden Betrieb können die Mess-Slaves von einem PC-Client aus per Fernzugriff verwaltet werden. Dies beinhaltet die Änderung der Parametrierung und das Auslesen der Messdaten. Dafür werden auf dem Mess-Slave persistent Parameter gespeichert, welche über die RS232 Schnittstelle geschrieben und ausgelesen werden können.

3.3. Anforderungen

Folgende Hauptanforderungen werden an das System gestellt:

- Individuelle Parametrierung der Prüfobjekte
- Automatische Erfassung der Messdaten
- Fernzugriff auf den Mess-Server zum Abrufen der Messdaten

Individuelle Parametrierung der Prüfobjekte

Immer 64 Prüfobjekte befinden sich auf einem Mess-Slave. Dabei sollen verschiedene Parameter für die Prüfobjekte berücksichtigt werden. Zum Einen sollen die Intervalle in denen Messwerte aufgenommen werden, konfigurierbar sein. Dies soll in mindestens 3 verschiedenen Intervallen möglich sein. In Tabelle 3.3.1 ist ein Beispiel für die Messintervalle zu sehen.

Zeitraum	Zeit zwischen Messungen
1. Woche	12 Stunden
2. bis 4. Woche	2 Tage
ab 5. Woche	7 Tage

Tab. 3.3.1.: Beispiel: Intervalle

Zum Anderen soll für jeden Mess-Slave ein Pulsmuster definierbar sein. Dieses Pulsmuster wird dann als Ansteuerungssignal für die DUTs verwendet.

Automatische Erfassung der Messdaten

Die Messdaten der DUTs sollen zyklisch erfasst werden. Es soll die derzeitige Temperatur im Temperaturschrank, der gemessene Wert des Empfängers und ein Zeitstempel gespeichert werden. Dabei sollen, wie bereits erwähnt, die Intervalle zwischen den Messungen konfigurierbar sein. Für den einfachen und effizienten Zugriff auf die Daten sollen sie in einer SQL Datenbank abgelegt werden. Dafür ist eine Kommunikationsschnittstelle zwischen der Datenbank und den Mess-Slaves erforderlich, welche über den Mess-Server realisiert wird.

Fernzugriff auf den Mess-Server

Zur Auswertung der Messdaten soll es möglich sein, von einem PC-Arbeitsplatz aus eine Verbindung zu dem Mess-Server aufzubauen. Die Messdaten sollen grafisch auf dem PC-Client zur Auswertung aufbereitet werden. Auch soll ein Tunnelmodus direkten Zugriff von einem PC-Client auf einen Mess-Slave ermöglichen. Dabei sollen die Parameter des Mess-Slaves verändert werden können.

Diese Basis-Anforderungen und einige zusätzliche Sekundäranforderungen können wie in Tabelle 3.3.2 in funktionale und nicht-funktionale Anforderungen unterteilt werden.

Art	Anforderung	Kommentar
Nicht-Funktional	Das System soll jederzeit verfügbar sein.	Bei Fehlern soll das System ohne große Ausfallzeit wieder Einsatzbereit sein. Zuverlässigkeit ist sehr wichtig.
Nicht-Funktional	Das Benutzerinterface soll zeitnah auf Anfragen reagieren.	Um Benutzerfreundlichkeit zu gewährleisten, soll auf Nutzeranfragen ohne lange Wartezeiten reagiert werden.
Funktional	Das System soll das Degradationsverhalten eines Prüfobjektes aufnehmen	Hauptanforderung des Systems. Messdaten sollen zu einem DUT gesammelt werden, um den Grad der Degradation bestimmen zu können.
Funktional	Neue, bereits parametrierte Mess-Slaves sollen automatisch in das System integrierbar sein.	Ein parametrierter Mess-Slave kann direkt an den Kommunikationsbus angeschlossen werden.
Funktional	Zyklische Erfassung von Messdaten.	Intervalle der Messdatenerfassung sind konfigurierbar.
Funktional	Messdaten sollen grafisch dargestellt werden.	Um die Daten auswerten zu können sollen sie grafisch aufbereitet werden.
Funktional	Die Messdaten sollen in einer Datenbank abgelegt werden.	Zum einfachen und effizienten Zugriff auf die Daten.
Funktional	Die Messdaten sollen via Fernzugriff erreichbar sein.	Von einem PC-Client aus, soll auf die Daten im lokalen Netzwerk zugegriffen werden können.
Funktional	Der Status des Systems soll ablesbar sein.	Über eine Anzeige soll das System lokal überwacht werden können.
Funktional	Nutzer-Fehler sollen abgefangen werden.	Fehler bei der Bedienung durch den Nutzer sollen unterbunden werden.

Tab. 3.3.2.: Anforderungen

4. Design

Im folgenden Kapitel wird das Design des Systems beschrieben. Zunächst wird auf den Aufbau der Hardware eingegangen. Anschließend wird sich näher mit dem Design und dem Aufbau der einzelnen Softwareelemente befasst. Zum Ende des Kapitels gibt es einen genaueren Einblick in die Gestaltung und den Aufbau des Protokolls für die RS232 Schnittstelle.

4.1. Hardware

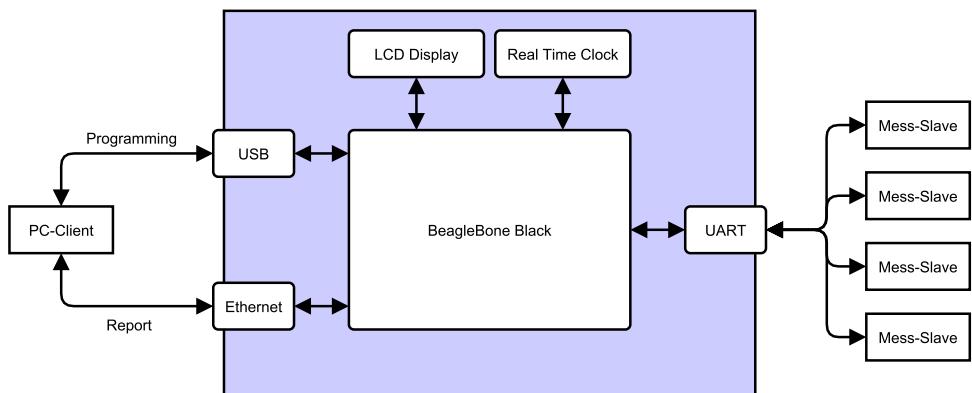
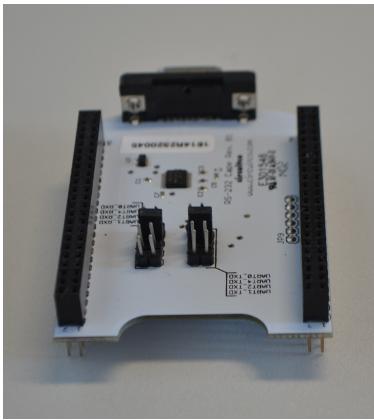
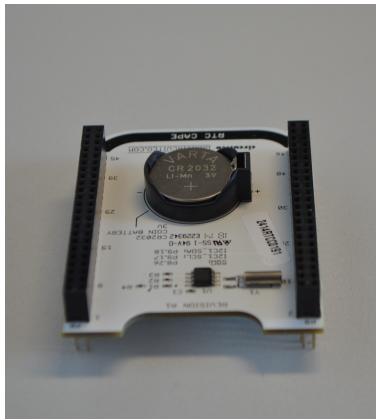


Abb. 4.1.1.: Übersicht Hardware

Das BeagleBone Black selbst ist bereits sehr Leistungsstark. Um jedoch weitere Funktionen und Schnittstellen hinzuzufügen, existieren so genannte Capes. Ein Cape ist eine für das BeagleBone konzipierte Erweiterung, die direkt auf das BeagleBone aufgesteckt werden kann. Die Treiber vieler dieser Capes sind bereits in dem Betriebssystem des BeagleBones integriert oder werden vom Hersteller bereitgestellt. Somit ist die Inbetriebnahme sehr komfortabel.

**Abb. 4.1.2.: RS232 Cape****Abb. 4.1.3.: RTC Cape****Abb. 4.1.4.: LCD Cape**

Zur Kommunikation mit den Mess-Slaves wird die Universal Asynchronous Receiver Transmitter (UART) Schnittstelle des BeagleBone Black verwendet. Dafür wird ein RS232 Cape eingesetzt (siehe Abbildung 4.1.2). Das Cape führt die seriellen Ports UART0, UART1, UART2 und UART4 auf einen 9-poligen seriellen Stecker. Es bietet die Möglichkeit zwischen den verschiedenen Ports mittels eines Jumpers auf dem Cape zu wechseln. Angeschlossen wird nach dem 3-wire Prinzip. Dabei werden lediglich Rx, Tx und die Masse verbunden. Somit ist keine Hardware-Flusssteuerung möglich.

Da das BeagleBone Black kein eigenes Real Time Clock (RTC) Modul besitzt, wird auch dieses durch ein Cape hinzugefügt (siehe Abbildung 4.1.3). Es beinhaltet eine 3V Knopfbatterie um auch im Falle einer Stromunterbrechung die aktuelle Zeit nicht zu verlieren. Dies ist sehr wichtig, da es erforderlich ist, dass das BeagleBone die aktuelle Uhrzeit und das aktuelle Datum jederzeit kennt. Denn beim Erfassen der Messdaten wird ein Zeitstempel angelegt um die Daten später zeitlich zuordnen zu können. Sollte dieser Zeitstempel nicht korrekt sein, sind die Daten bei der Auswertung nicht gültig.

Um die Statusanzeige detailliert darstellen zu können, wird ein resistives LCD-Touchscreen Display eingesetzt. Es hat eine Größe von 4,3 Zoll bei einer Auflösung von 480x272 Pixeln. Dabei handelt es sich ebenso um ein Cape (siehe Abbildung 4.1.4) . Dadurch ist es möglich, das BeagleBone Black trotz der Erweiterungen kompakt zu halten. Denn die Capes sind untereinander stapelbar (siehe Abbildung 4.2.1).



Abb. 4.1.5.: Gestapelte Capes

Die USB Schnittstelle, welche zur Programmierung des BeagleBones verwendet wird, ist bereits vollständig einsatzbereit. Ebenso ist die Ethernet Schnittstelle, welche für den Fernzugriff auf das BeagleBone genutzt wird standardmäßig vollständig integriert.

4.2. Software

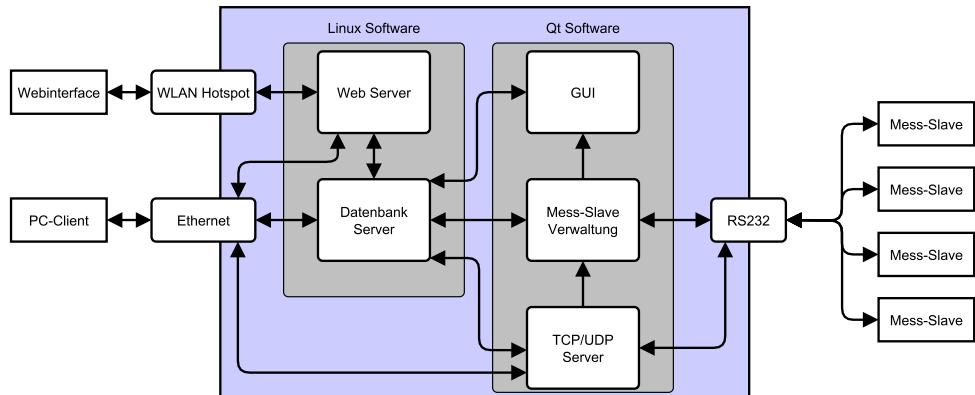


Abb. 4.2.1.: Übersicht Software

Das Software System setzt sich aus zwei Teilen zusammen. Zum Einen die selbst entwickelte Software, welche in C++ unter Verwendung der Klassenbibliothek Qt (siehe Abschnitt 2.2) programmiert ist und zum Anderen die verschiedenen Softwarepakete die für das eingesetzte Linux Betriebssystem zur Verfügung stehen.

Im folgenden Abschnitt wird auf beide Softwareteile eingegangen. Zunächst werden die Bestandteile Mess-Slave Verwaltung, Statusüberwachung und TCP/UDP-Server der Qt Anwendung näher erläutert und anschließend wird auf den Datenbankserver, den Webserver und den WLAN Hotspot eingegangen.

4.2.1. Mess-Slave Verwaltung

Die Verwaltung der Mess-Slaves ist der Hauptprozess der Software. In diesem Programmteil wird die Kommunikation mit den Mess-Slaves realisiert. Dabei sind die Grundaufgaben die Messdatenerfassung, das Erkennen von neuen Mess-Slaves und die automatische Eingliederung dieser in das System.

4.2.1.1. Anmeldung

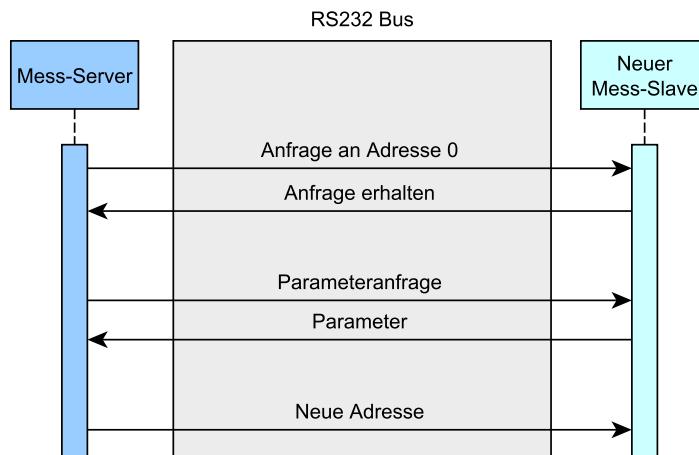


Abb. 4.2.2.: Mess-Slave Anmeldung

Um in das bestehende System eingegliedert zu werden, muss jeder Mess-Slave sich bei dem Mess-Server anmelden (siehe Abbildung 4.2.2). Dafür hat jeder neue Mess-Slave die RS232 Adresse 0. Der Mess-Server überprüft in regelmäßigen Abständen, ob über die Adresse 0 ein Mess-Slave erreichbar ist. Sobald dies der Fall ist, übermittelt der entsprechende Mess-Slave seine Parameterierung an den Mess-Server. Die Parameter werden dann automatisch vom Mess-Server in der Datenbank abgelegt. Im Anschluss bekommt der Mess-Slave eine neue Adresse zugewiesen über die er in Zukunft im System erreichbar ist.

4.2.1.2. Messdatenerfassung

Der Hauptzyklus der Software ruft kontinuierlich die Messdaten von den Mess-Slaves ab. Dafür wird ständig geprüft ob Messungen erforderlich sind. Dies geschieht durch den Vergleich der vergangenen Zeit zur letzten Messung und den gegebenen Parametern für die Messintervalle. In Tabelle 4.2.1 finden sich diese Parameter.

Parameter	Beschreibung
duration_int1	Dauer des 1. Zeitraums
duration_int2	Dauer des 2. Zeitraums
interval_1	Abstand zwischen den Messungen im 1. Zeitraum
interval_2	Abstand zwischen den Messungen im 2. Zeitraum
interval_3	Abstand zwischen den Messungen nach dem 2. Zeitraum

Tab. 4.2.1.: Parameter der Messintervalle

Ob eine Messung erforderlich ist, lässt sich aus den gegebenen Parametern ableiten. So wird zuerst geprüft, welcher Zeitraum (*duration_int1-2*) derzeit zutrifft. Dazu wird die vergangene Zeit seit der ersten Messung mit dem Zeitraum abgeglichen. Sollten noch keine Ergebnisse vorhanden sein, ergibt dies die Erforderlichkeit einer Messung.

Wenn der passende Zeitraum ausgemacht ist, wird das dazugehörige Intervall zwischen den einzelnen Messungen (*interval_1-3*) ausgemacht. Dann wird geprüft, ob die vergangene Zeit seit der letzten Messung größer ist als dieses Intervall.

Anschließend wird geprüft, ob der Mess-Slave verfügbar ist. Dazu wird eine Namensanfrage über die RS232 Schnittstelle verschickt. Bei einer positiven Antwort wird dann eine Messung durchgeführt. Dabei werden alle Empfänger der 64 möglichen DUTs mittels *ADC-Value*-Befehls (siehe Tabelle 4.3.9) ausgelesen. Sobald alle 64 Werte erfolgreich ermittelt sind, werden sie in der Datenbank abgelegt.

Sollte keine Antwort auf die Namensanfrage erfolgen, wird in den nächsten Zyklen erneut versucht eine Verbindung zu etablieren. Bei kontinuierlich erfolglosen Verbindungsversuchen informiert das System den Nutzer nach einer festgelegten Zeit der Abwesenheit über die Unerreichbarkeit.

4.2.2. Statusüberwachung

Um die Anforderung der Statusüberwachung zu erfüllen, verfügt das BeagleBone über eine GUI. Sie soll einen einfachen Überblick über die aktuellen Vorgänge bieten und auf einen Blick den Status des Mess-Servers wiedergeben. Angezeigt wird die GUI über das LCD Display Cape des BeagleBones.

Es sollen die wichtigsten Funktionen für den Betrieb überwacht und angezeigt werden, sodass der Administrator informiert wird, sobald ein manuelles Eingreifen erforderlich ist. Überwacht werden die korrekte Funktion der angeschlossenen Mess-Slaves und die lokale Systemauslastung des BeagleBones.

Zur Überwachung der Mess-Slaves, werden sie in drei Gruppen unterteilt. Alle Mess-Slaves, die eine RS232 Adresse besitzen, gelten als aktiv. Mess-Slaves, denen keine RS232 Adresse zugewiesen ist, gelten als inaktiv. Sie sind zwar noch in der Datenbank gespeichert, wurden allerdings durch das Löschen der RS232 Adresse vom System abgemeldet.

Die dritte und kritische Gruppe sind Mess-Slaves denen eine RS232 Adresse zugewiesen ist, die aber nicht über den RS232 Bus erreichbar sind. Bei ihnen liegt mit großer Wahrscheinlichkeit ein Fehler vor, der vom Administrator manuell behoben werden muss.

Bei der Überwachung der Systemauslastung, ist vor allem die Speichernutzung interessant. Da das System über lange Zeiträume aktiv ist, fallen auch große Mengen an Daten an. Da allerdings nur ein beschränkter Speicherplatz zur Verfügung steht, wird der Nutzer über die Auslastung informiert. Im Notfall kann der Administrator Speicher durch das Löschen von nicht mehr relevanten Messdaten freigegeben.

4.2.3. TCP/UDP Server

Um auf Netzwerkanfragen reagieren zu können, ist sowohl ein UDP-Server für Broadcast-Nachrichten als auch ein TCP-Server für die direkte Kommunikation realisiert.

Der UDP-Server dient dabei zur dynamischen Erkennung von Mess-Servern im Netzwerk. Das User Datagram Protocol (UDP) ist ein verbindungsloses Netzwerkprotokoll. Es werden Nachrichten an ein Ziel versandt, ohne eine Rückmeldung zu erwarten. Dabei handelt es sich um ein nicht-zuverlässiges, ungesichertes und ungeschütztes Protokoll. Diese Art der Übertragung eignet sich vor allem für das Aufspüren von Kommunikationspartnern. Es kann eine UDP-Nachricht gleichzeitig an alle möglichen Adressen von Kommunikationspartnern im Netzwerk verschickt werden, um diese zu ermitteln. Das geschieht meist über die Broadcast-Adresse eines Netzwerkes. Über die Broadcast-Adresse können alle Teilnehmer eines Netzes gleichzeitig angesprochen werden. Der Mess-Server antwortet auf Broadcast-Nachrichten mit seiner IP-Adresse und seinem Namen. Dies ermöglicht die einfache Eingliederung der Mess-Server in ein bestehendes System.

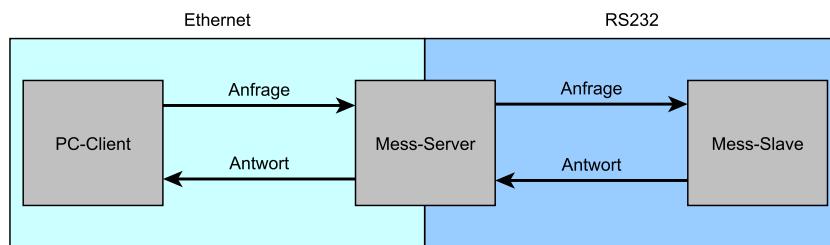
Das Transmission Control Protocol (TCP) ist im Gegensatz zu UDP ein verbindungsorientiertes Netzwerkprotokoll. Es ist ein zuverlässiges Übertragungsprotokoll und stellt die Gültigkeit der Übertragung sicher. Der TCP-Server nimmt RS232-Befehle an und leitet diese weiter. Das ermöglicht den Fernzugriff auf die einzelnen Mess-Slaves über ein Netzwerk. In Tabelle 4.2.2 ist eine Liste der möglichen Befehle zu sehen.

Befehl	Typ	Daten	Kommentar
BeagleSendYourIP	UDP	-	Das BeagleBone antwortet dem Sender mit seiner IP Adresse und seinem Namen
BeagleUpdateConfig	UDP	-	Das BeagleBone aktualisiert seine Konfiguration aus der Config-Datei
RS232CMD:	TCP	Nachricht	Das BeagleBone sendet den empfangenen Rahmen über seine RS232 Schnittstelle

Tab. 4.2.2.: Ethernet Befehlsliste

Der TCP/UDP-Server ist vor allem für die Kommunikation mit dem PC-Client zuständig. Um Mess-Server im Netzwerk zu finden, wird vom PC-Client ein *BeagleSendYourIP*-Befehl als Broadcast-Nachricht versendet. Sollte ein Mess-Server diese Nachricht empfangen, antwortet er an die Sendeadresse mit der eigenen IP Adresse und dem eigenen Namen.

Über den *RS232CMD*: Befehl kann von einem PC-Client aus direkt mit einem am RS232 Bus angeschlossenen Mess-Slave kommuniziert werden. Der TCP-Server dient dabei als Tunnel für die Kommunikation (siehe Abbildung 4.2.3). Anfragen werden vom Mess-Server angenommen und weitergeleitet. Die Antworten nehmen den umgekehrten Weg zurück. Zur Sicherstellung der alleinigen Verwendung der RS232 Schnittstelle, pausiert der TCP-Server die anderen Prozesse, die auf die Schnittstelle zugreifen, für die Dauer der Kommunikation.

**Abb. 4.2.3.:** RS232 Tunnel

4.2.4. Web Server und WLAN Hotspot

Mit Hilfe eines Web Servers ist es auch lokal möglich eine Übersicht über die gesammelten Daten auf dem Mess-Server zu erhalten. Als Schnittstelle steht ein WLAN Hotspot zur Verfügung.

Als Webserver dient Lighttpd (auch: Lighty). Er stellt das Webinterface für Clients zum Abruf zur Verfügung. Lighttpd ist laut [Bog08] ein erweiterbarer, modularer, Single-Threaded, hochperformanter Webserver, der selbst Apache oder Microsofts IIS in den meisten Einstellungen übertrifft. Besonders auf kleinen Systemen mit begrenzten Ressourcen kommt er häufig zum Einsatz. Trotzdem vertrauen auch große Webanbieter wie YouTube oder Wikipedia auf seine Effizienz. Zusätzlich sind das PHP und MySQL Common Gateway Interface (CGI) Modul installiert um die Messergebnisse aus der Datenbank dynamisch darstellen zu können.

Der Webserver ist sowohl über die normale Ethernet Verbindung als auch einen WLAN Hotspot erreichbar. Für die Realisierung des WLAN Hotspots wird ein EDIMAX EW-7811Un WLAN Dongle über den Host-USB Port (siehe Anhang A.1) des BeagleBones betrieben. Der Hotspot ermöglicht den drahtlosen Zugriff auf den Mess-Server. Er stellt einen Verbindungspunkt für drahtlose Verbindungen dar.

Als Hotspot Software dient Hostapd. Hostapd ist ein IEEE 802.11 Accesspoint und IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authentificator (vgl. [Wir]) und verwaltet die Zugriffe auf den Hotspot.

Zusätzlich müssen Geräten, die sich über den Hotspot mit dem BeagleBone Black verbinden eine IP-Adresse zugewiesen werden. Dafür kommt der DNS- und DHCP-Server dnsmasq zum Einsatz. Dnsmasq kann als DNS-Server auch alle Anfragen an die Adresse des Webservers umleiten. Somit ist es für den Nutzer nicht nötig, die Adresse des Webservers manuell einzugeben. Er kann einfach über einen Standard-Webbrowser eine beliebige Adresse aufrufen und wird automatisch an die Adresse des Webservers umgeleitet.

4.2.5. Datenbank Server

Wie in vorangehenden Kapiteln bereits erwähnt, werden alle Messdaten und Betriebsparameter in einer MySQL-Datenbank abgelegt. Der MySQL-Datenbankserver läuft auf dem BeagleBone Black und ist über das Netzwerk erreichbar.

Aus den Anforderungen ergibt sich für die Datenbank folgendes Entity-Relationship-Modell (ERM) (siehe Abbildung 4.2.4).

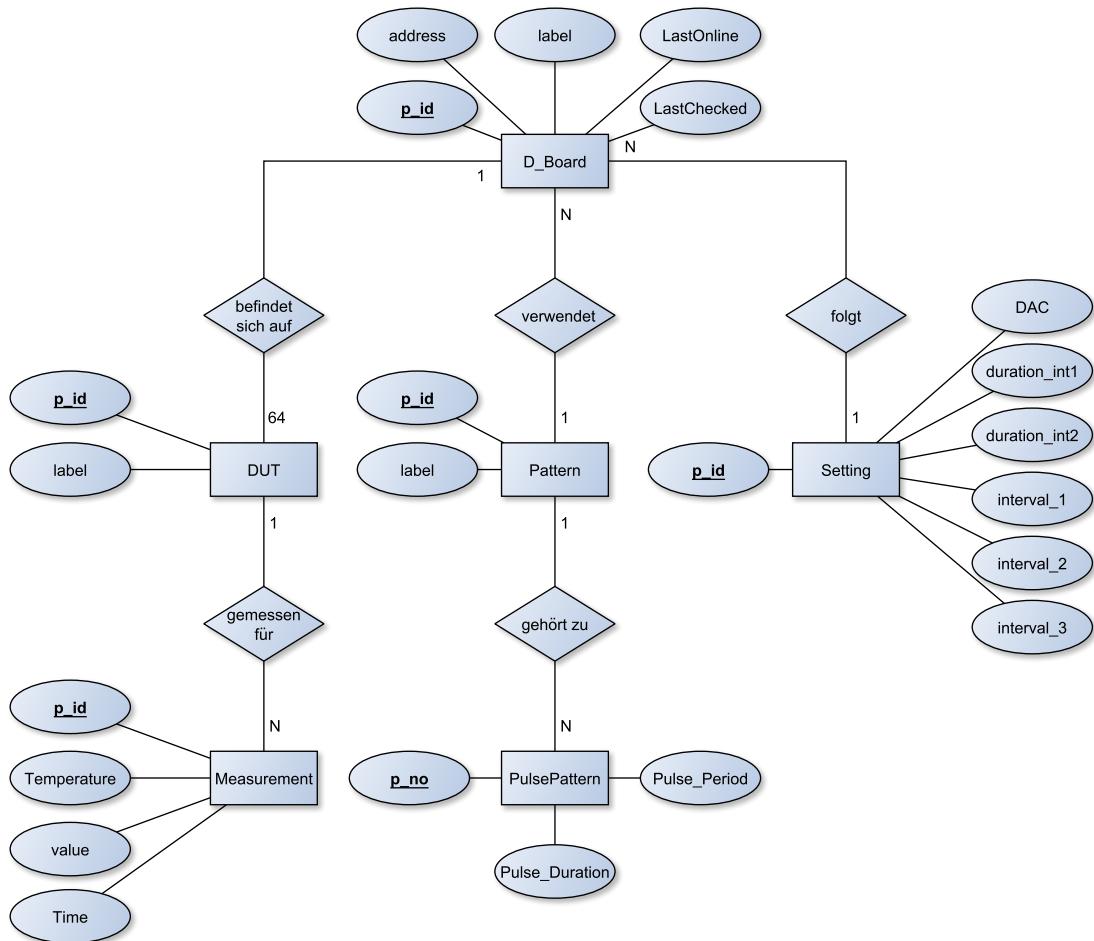


Abb. 4.2.4.: Entity Relationship Modell

4.2.5.1. Tabellen

Im folgenden Abschnitt wird näher auf die einzelnen Tabellen und ihre Beziehungen untereinander eingegangen. Außerdem werden die Attribute näher erläutert.

Tabelle D_Board

Parameter	Datentyp
p_id	INTEGER PRIMARY KEY
address	INTEGER UNIQUE
label	VARCHAR(255) UNIQUE
LastOnline	TIMESTAMP
LastChecked	TIMESTAMP
f_settingid	INTEGER FOREIGN KEY
f_Patternid	INTEGER FOREIGN KEY

Tab. 4.2.3.: Tabelle D_Board

Die Tabelle D_Board (auch: Degradations-Board) speichert alle im System registrierten Mess-Slaves. Jeder Mess-Slave hat eine einzigartige Identifikationsnummer (*p_id*), welche gleichzeitig den Primärschlüssel (englisch: primary key) repräsentiert. Um die Mess-Slaves später leserlich auseinanderhalten zu können, verfügen sie zusätzlich über einen Namen (*label*), der einzigartig sein muss.

Zur Kommunikation wird jedem Mess-Slave (auch:Degradationsboard) eine einzigartige Adresse (*address*) zugeordnet. Über den RS232 Bus ist er dann mittels dieser Adresse erreichbar.

Jedes Degradationsboard besitzt einen Zeitstempel (*LastOnline*) für den Zeitpunkt der letzten erfolgreichen Kontaktaufnahme. Diese Information ist notwendig um die Dauer der Abwesenheit ermitteln und den Nutzer dahingehen in Kenntnis setzen zu können.

Des Weiteren gibt es noch einen Zeitstempel (*LastChecked*) für den letzten Zeitpunkt des Versuches einer Kontaktaufnahme, welcher notwendig ist, um unnötig häufige Kontaktversuche zu vermeiden. So soll ein Degradationsboard nicht häufiger als alle 10 Minuten kontaktiert werden um die Performance zu erhöhen.

Zusätzlich folgt jedes Board genau einer Einstellung (Tabelle Setting). Diese Relation wird durch einen Fremdschlüssel (englisch: foreign key) mit der Identifikationsnummer einer Einstellung (*f_Settingid*) referenziert.

Außerdem hat jedes Board ein Pulsmuster (Tabelle Pattern) zur Ansteuerung der Prüfobjekte. Dieses wird durch die Identifikationsnummer eines Pulsmusters(*f_Patternid*) angegeben.

Tabelle Setting

Parameter	Datentyp
p_id	INTEGER PRIMARY KEY
DAC	INTEGER
duration_int1	INTEGER
duration_int2	INTEGER
interval_1	INTEGER
interval_2	INTEGER
interval_3	INTEGER

Tab. 4.2.4.: Tabelle Setting

In der Tabelle Setting werden die Einstellungen für die verschiedenen Degradationsboards abgelegt. Jede Einstellung kann dabei mehreren Boards zugeordnet sein, wobei jedes Board nur eine Einstellung besitzt.

Der Primärschlüssel der Tabelle ist die Identifikationsnummer (*p_id*) jeder Einstellung. Als Nächstes gibt es die Einstellung für optische Leistung der Prüfobjekte (*DAC*) auf den Degradationsboards. Mithilfe dieses Wertes wird die Leistung der Prüfobjekte für die bessere Analyse durch die optischen Empfänger angepasst. Darauf folgend kommen die Parameter zur Konfiguration der Messintervalle (siehe auch Tabelle 4.2.1). Sie bestimmen die Zeiträume (*duration_int1 + 2*) und die Abstände zwischen den Messungen (*interval_1-3*) innerhalb dieser Zeiträume.

Tabelle DUT

Parameter	Datentyp
p_id	INTEGER PRIMARY KEY
label	VARCHAR(255)
f_Boardid	INTEGER FOREIGN KEY

Tab. 4.2.5.: Tabelle DUT

Die Tabelle DUT nimmt die einzelnen Prüfobjekte (DUTs) auf. Jedes Prüfobjekt gehört fest zu einem Degradationsboard. Dies wird durch den Fremdschlüssel mit der Identifikationsnummer des zu gehörenden Boardes (*f_Boardid*) referenziert. Außerdem hat jedes Prüfobjekt eine eindeutige Identifikationsnummer (*p_id*) als Primärschlüssel. Des Weiteren verfügt es über eine Bezeichnung (*label*), um sich von den anderen Prüfobjekten des selben Boardes zu unterscheiden.

Tabelle Pattern

Parameter	Datentyp
p_id	INTEGER PRIMARY KEY
label	VARCHAR(255)

Tab. 4.2.6.: Tabelle Pattern

In der Pattern Tabelle sind die Muster für die Signale zur Ansteuerung der Prüfobjekte. Die Muster haben eine Identifikationsnummer (*p_id*) als Primärschlüssel und eine Bezeichnung (*label*) für die Leserlichkeit. Die Tabelle fasst die Signale aus der PulsePattern Tabelle zusammen um einfacher referenziert werden zu können.

Tabelle PulsePattern

Parameter	Datentyp
pf_Patternid	INTEGER PRIMARY KEY FOREIGN KEY
p_no	INTEGER NOT NULL
Pulse_Duration	INTEGER
Pulse_Period	INTEGER

Tab. 4.2.7.: Tabelle PulsePattern

Die Tabelle PulsePattern enthält die Signale für die Muster in Tabelle Pattern. Die Identifikationsnummer des Musters (*pf_Patternid*) und die Position im Muster (*p_no*) dienen zusammen als ein Primärschlüssel. Die Identifikationsnummer des Musters (*pf_Patternid*) dient dabei gleichzeitig als Fremdschlüssel und referenziert ein Muster. Dadurch ist es möglich, sowohl die Zugehörigkeit zu einem spezifischen Muster, als auch die Position in ihm einzigartig zu bestimmen. Die Signaldauer (*Pulse_Duration*) beschreibt die Pulsbreite des aktiven Pegels innerhalb des Signals, während die Signalperiode (*Pulse_Period*) die Pulsperiodendauer des Signals angibt. So ergibt sich ein High-Pegel für die Dauer von *Pulse_Duration* und ein Low-Pegel für die Dauer von *Pulse_Period – Pulse_Duration*.

Tabelle Measurement

Parameter	Datentyp
p_id	INTEGER PRIMARY KEY
Temperatur	INTEGER
value	INTEGER
Time	TIMESTAMP DEFAULT CURRENT_TIMESTAMP
f_DUTid	INTEGER FOREIGN KEY

Tab. 4.2.8.: Tabelle Measurement

In der Measurement Tabelle werden die Messwerte der einzelnen Prüfobjekte gespeichert. Der Primärschlüssel ist auch hier die Identifikationsnummer (*p_id*). Zu jedem Messeintrag gehört die aktuelle Temperatur (*Temperatur*), der gelesene Messwert (*value*) und die aktuelle Zeit (*Time*). Ein Messeintrag gehört immer zu einem Prüfobjekt. Über den Fremdschlüssel mit der Identifikationsnummer des Prüfobjektes (*f_DUTid*) wird diese Beziehung referenziert.

4.3. RS232 Protokoll

Das Protokoll für die Kommunikation über die RS232 Schnittstelle dient zum Austausch von Informationen innerhalb des Systems zwischen den Akteuren.

Folgende Kriterien sollen dabei erfüllt werden:

- Adressierung individueller Kommunikationspartner
- Senden verschiedener Befehle
- Variable Größe der Daten
- Sicherstellung der Validität der Übertragung
- Erweiterbar

Um eine hohe Zuverlässigkeit gewährleisten zu können, wird eine geringe Baudate von 2400 verwendet. Da nur geringe Datenmengen in großen Abständen auftreten, verhindert dass die Anfälligkeit der Übertragung gegenüber äußeren Einflüssen, ohne dabei die Performance merkbar zu beeinflussen.

Des Weiteren wird ein Paritätsbit für eine Paritätsprüfung eingesetzt. Dieses Bit gibt an, ob die Anzahl der Einsen des Datenblocks gerade oder ungerade ist. Der Wert, des verwendeten Even-Paritätsbits ist 1, wenn die Summe der Einsen gerade ist und 0 wenn sie ungerade ist. Dadurch können grobe Fehler bei der Übertragung ausgemacht und korrigiert werden.

4.3.1. Aufbau

1. Byte	2. Byte	3. Byte	4. Byte	5. Byte und folgend	Letztes Byte
Adresse & R/W	Länge	Befehl	Unterbefehl	Nutzdaten	Checksumme

Tab. 4.3.1.: Übertragungsrahmen

Ein Rahmen des Protokolls besteht aus 4 Steuerbytes, 1 Checksummenbyte und maximal 30 Bytes an Nutzdaten. Durch diesen Aufbau können die Anforderungen erfüllt werden. Im folgenden Abschnitt wird auf die Zusammensetzung und die Funktion der einzelnen Bytes eingegangen.

1. Byte: Adresse & Read/Write

7. Bit	6. Bit	5. Bit	4. Bit	3. Bit	2. Bit	1. Bit	0. Bit
R/W	Addr6	Addr5	Addr4	Addr3	Addr2	Addr1	Addr0

Tab. 4.3.2.: 1. Byte: Adresse & Read/Write

Das erste Byte des Übertragungsrahmens setzt sich aus 7 Adressbits und einem Lese-/Schreibbit zusammen. Die ersten 7 Bit (Addr0 - Addr6) bilden die Adresse des anzusteuernenden Empfängers. Daraus ergibt sich ein Adressraum von möglichen 128 Adressen, wobei Adresse 0 für neue Mess-Slaves zur einmaligen Anmeldung im System reserviert ist (siehe Abschnitt 4.2.1).

Das höchste Bit ist das Lese-/Schreibbit. Mithilfe dieses Bits wird unterschieden, ob ein Befehl als Lese- oder Schreibzugriff interpretiert werden soll.

R/W Bit	Beschreibung
0	Die Sender möchte schreiben
1	Die Sender möchte lesen

Tab. 4.3.3.: Read/Write

Ein Lesezugriff stellt dabei eine Anfrage dar. Dass heißt, dass keine Nutzdaten übertragen werden und eine Antwort des Empfängers mit den angeforderten Daten erwartet wird. Bei einem Schreibzugriff hingegen, werden immer Nutzdaten übertragen.

2. Byte: Rahmenlänge

7. Bit	6. Bit	5. Bit	4. Bit	3. Bit	2. Bit	1. Bit	0. Bit
Len7	Len6	Len5	Len4	Len3	Len2	Len1	Len0

Tab. 4.3.4.: 2. Byte: Rahmenlänge

Das zweite Byte gibt die Länge des gesamten Rahmens inklusive Steuerbytes, Nutzdaten und Checksumme an. Die minimale Rahmenlänge beträgt 5 Bytes. Dabei handelt es sich um eine Übertragung ohne Nutzdaten und es werden lediglich die 4 Steuerbytes und das Byte für die Checksumme übertragen. Dies geschieht beispielsweise bei einer Leseanfrage.

Die maximale Rahmenlänge beträgt 35 Byte. Hierbei werden zusätzlich zu den 4 Steuerbytes und dem Byte für die Checksumme auch die maximale Nutzlast von 30 Byte übertragen. Dieser Fall kann beispielsweise bei Schreibzugriffen auftreten.

Anhand der Länge kann eine erste Prüfung der Validität eines Rahmens durchgeführt werden.

Sollte die Zahl der empfangenen Bytes sich von der Rahmenlänge unterscheiden, kann von einem ungültigen Rahmen ausgegangen werden.

3. Byte: Befehl

7. Bit	6. Bit	5. Bit	4. Bit	3. Bit	2. Bit	1. Bit	0. Bit
Cmd7	Cmd6	Cmd5	Cmd4	Cmd3	Cmd2	Cmd1	Cmd0

Tab. 4.3.5.: 3. Byte: Befehl

Das dritte Byte repräsentiert den Befehl. Dieser gibt an, welche Aktion ausgeführt oder welcher Parameter angesprochen wird. Da insgesamt ein Byte für den Befehl zur Verfügung steht, sind bis zu 256 unterschiedliche Befehle möglich.

4. Byte: Unterbefehl

7. Bit	6. Bit	5. Bit	4. Bit	3. Bit	2. Bit	1. Bit	0. Bit
Scmd7	Scmd6	Scmd5	Scmd4	Scmd3	Scmd2	Scmd1	Scmd0

Tab. 4.3.6.: 4. Byte: Unterbefehl

Das vierte Byte ist der Unterbefehl. Damit ist es möglich einen Befehl genauer zu spezifizieren. So kann beispielsweise der Befehl zum Auslesen eines Messwertes durch den Unterbefehl genau auf einen der 64 Prüfobjekte präzisiert werden. Im späteren Verlauf wird in Abschnitt 4.3.2 näher auf die vorhandenen Befehle und Unterbefehle eingegangen.

5. Byte und folgend: Nutzdaten

7. Bit	6. Bit	5. Bit	4. Bit	3. Bit	2. Bit	1. Bit	0. Bit
Data7	Data6	Data5	Data4	Data3	Data2	Data1	Data0

Tab. 4.3.7.: 5. Byte und folgend: Nutzdaten

Das fünfte Byte und die darauf folgenden, tragen die Nutzdaten des Rahmens. Die Größe der Nutzdaten ist variabel und lässt sich von der Rahmenlänge ableiten. Bei 16 Bit Variablen wird immer im Big-Endian-Format, das höhere Byte zuerst, übertragen.

Letztes Byte: Checksumme

7. Bit	6. Bit	5. Bit	4. Bit	3. Bit	2. Bit	1. Bit	0. Bit
CKS7	CKS6	CKS5	CKS4	CKS3	CKS2	CKS1	CKS0

Tab. 4.3.8.: Letztes Byte: Checksumme

Das letzte Byte ist immer die Checksumme, um sicherzustellen, dass alle Daten komplett und fehlerfrei übertragen wurden.

Der Sender bildet dabei die Checksumme mittels einer XOR Verknüpfung aller Bytes eines Übertragungsrahmens. Beim Empfänger werden alle Bytes inklusive Checksumme erneut mit XOR Verknüpft, wobei ohne Fehler immer 0 das Ergebnis sein muss.

Beispiel

Sender:

Es wird angenommen das folgender Rahmen übertragen werden soll.

Adresse	Länge	Befehl	Unterbefehl
1	5	9	0

Aus dem Rahmen wird dann mittels XOR die Checksumme gebildet.

$$1 \oplus 4 \oplus 9 \oplus 0 = 13$$

Anschließend wird die Checksumme als letztes Byte an den Rahmen an gehangen.

Adresse	Länge	Befehl	Unterbefehl	Checksumme
1	5	9	0	13

Empfänger:

Sobald der Empfänger den Rahmen erhält, verknüpft er alle Bytes erneut mittels XOR um die Gültigkeit zu prüfen.

$$1 \oplus 4 \oplus 9 \oplus 0 \oplus 13 = 0$$

Das Ergebnis 0 repräsentiert einen gültigen Rahmen und die Checksummenprüfung war erfolgreich.

4.3.2. Befehle und Unterbefehle

Für die Funktion des Systems sind verschiedene Befehle notwendig. Sie dienen zur Kommunikation zwischen den Akteuren. Eine Übersicht über die implementierten Befehle der RS232 Schnittstelle findet sich in Tabelle 4.3.9.

Befehl	Code	Unterbefehl	Datenbytes
ADC-Value	0x00	MUX-Kanal (0..63)	2
Number Of Pulses	0x04	-	1
Pulsewidth and -period	0x05	Pulsnummer (1..20)	4
Perform Pulseupdate	0x06	-	0
DAC-Value	0x07	-	2
Temperature	0x08	-	1
LTT Name	0x09	-	1..30
Rs232-Address	0x0A	-	1
Measurement Intervall	0x0C	Intervallnummer (0..2)	4

Tab. 4.3.9.: RS232 Befehlsliste

ADC-Value

Liest den Messwert eines Sensors des Mess-Slaves. Mit dem Unterbefehl muss einer der 64 Empfänger spezifiziert werden. Ein Messwert hat die Größe von 2 Byte. Dabei wird das höhere Byte zuerst übertragen, also im Big-Endian-Format.

Number Of Pulses

Liest oder schreibt die Anzahl der Impulse im Pulsmuster des Mess-Slaves. Die maximale Anzahl ist 20 und sie wird als 1 Byte Wert übertragen.

Pulsewidth and -period

Liest oder schreibt ein Pulsbreiten-/Pulsperiodenpaar. Der Unterbefehl bestimmt die Position im Pulsmuster. Sowohl die Pulsbreite als auch die Pulsperiode sind jeweils 2 Byte groß. Die ersten 2 Byte repräsentieren die Pulsbreite und die zweiten 2 Byte die Pulsperiode. Die Übertragung erfolgt im Big-Endian-Format.

DAC-Value

Liest oder schreibt den Wert für die optische Leistung der Prüfobjekte. Sie ist 12 Bit groß und hat somit einen Wertebereich von 0 bis 4096. Die Übertragung erfolgt im Big-Endian-Format.

Temperatur

Der Befehl liest den Wert des Temperatursensors. Er ist nur zum Lesen des Temperaturwertes gedacht und kann nicht zum schreiben verwendet werden. Es handelt sich um einen signed 1 Byte Wert. Der Wertebereich geht von 127 bis -127.

LTT Name

Liest oder schreibt den Namen des Mess-Slaves. Die maximale Zeichenlänge beträgt 30 Zeichen. Das erste Zeichen wird auch zuerst übertragen.

RS232-Adresse

Liest oder schreibt die Adresse für die RS232 Kommunikation des Mess-Slaves. Es handelt sich um einen 7 Bit Wert. Der Wertebereich geht von 0 bis 127 was 128 Adressen entspricht.

Measurement Intervall

Liest oder schreibt die Konfiguration der Messintervalle. Der Unterbefehl gibt dabei den Zeitraum (0 bis 2) an. Die ersten 2 Byte enthalten das Intervall zwischen den Messungen und die zweiten 2 Byte den Zeitraum. Die Übertragung erfolgt im Big-Endian-Format.

5. Implementierung

Die Installation der Linux Software erfolgt mittels des Terminalemulators PuTTY [ubu]. Mit PuTTY wird eine Secure Shell (SSH) Verbindung zu dem BeagleBone Black aufgebaut, wodurch ein direkter Zugriff auf das Linux Terminal des BeagleBone Black möglich ist. Mit dem Linux Paketmanager kann bei aktiver Internetverbindung die benötigte Software installiert werden.

Bei der Implementierung der eigenen Software wird wie in Kapitel 4 erwähnt, die Qt C++ Klassenbibliothek in der Version 4.8.6 verwendet. Für die komfortable Programmierung eignet sich der Qt Creator als Entwicklungsumgebung sehr gut. Er bietet die Möglichkeit direkt in der Entwicklungsumgebung eine Toolchain einzurichten. Diese erlaubt eine Programmierung des Codes auf einem normalen Windows Rechner, wobei der Code anschließend automatisch übersetzt und auf dem Zielsystem über eine SSH Verbindung installiert wird. Verwendet wird dabei ein frei verfügbarer GNU-C-Compiler (GCC).

Die Steuerungssoftware teilt sich in einen sequentiellen Prozess für die Abfrage und Speicherung der Messwerte, sowie einen Event gesteuerten Prozess für die GUI und die externe Kommunikation für die Fernzugriffe. Die beiden Programmteile können unabhängig voneinander agieren und kommunizieren ausschließlich über Signale und Slots (siehe Abschnitt 2.2.1). Durch diese Kapitelung ist es möglich die beiden Programmteile durch andere Lösungen auszutauschen, welche lediglich die selben Schnittstellen in Form der Signale und Slots unterstützen müssen.

5.1. Systemkonfiguration

Beim Hochfahren des Mess-Servers sorgt der Linux Hintergrunddienst systemd für die Ausführung aller nötigen Prozesse. Systemd ist unter Linux der Prozess mit der ID 1. Er ist der Initialisierungsprozess und wird als erster Prozess gestartet. Des Weiteren übernimmt er den Start und die Überwachung aller anderen Prozesse die beim Systemstart ausgeführt werden. Diese Prozesse werden auch Service genannt. Um zusätzlich die Steuerungssoftware zu starten, wird ein neuer Service erstellt und dem systemd mitgeteilt.

```

[ Unit ]
2 Description=Boot Setup
[ Service ]
4 TimeoutStartSec=300
EnvironmentFile=/etc/Bootsetup.conf
6 ExecStart=/usr/bin/Bootsetup
Restart=always
8 RestartSec=30
[ Install ]
10 WantedBy=multi-user.target

```

Quellcode 5.1: Systemd Service

Im Quellcode 5.1 ist die Konfiguration des Services zu sehen. Er wird beim Systemstart automatisch gestartet und führt anschließend ein Script aus, das das System konfiguriert und die Qt Anwendung startet. Des Weiteren wird der Service automatisch erneut ausgeführt, falls ein Fehler, der die Qt Anwendung beendet, auftritt. Dadurch soll sichergestellt werden, dass das System jederzeit verfügbar ist. Die Datei mit dem Quellcode des Services wird im Verzeichnis des systemd abgelegt.

Das Systemstart Script initialisiert zunächst das RS232 Cape, das RTC Cape und die Systemuhr. Anschließend führt es die Qt Anwendung aus.

```

#!/bin/bash
2 echo BB-UART4 >/sys/devices/bone_capemgr.*/slots
echo BBB-RTC-01:00A1 >/sys/devices/bone_capemgr.*/slots
4 sleep 60
hwclock -s -f /dev/rtc1
6 sleep 60
cd /root/Progs
8 ./LTT_MeasMaster_GUI -qws

```

Quellcode 5.2: Systemstart Script

Im Quellcode 5.2 ist das Systemstart Script zu sehen. Die Initialisierung des RTC und RS232 Capes erfolgt über den für das BeagleBone Black verfügbaren Cape Manager (capemgr). Der Capemgr [Cap] dient zum einfachen Einbinden von Capes in das System. Er lädt unter anderem die benötigten Treiber und reserviert die nötigen General Purpose Input/Output (GPIO) Pins. Das Script befindet sich im Verzeichnis `/usr/bin/`.

Da das BeagleBone Black über keine persistente Uhr verfügt, wird nachdem die beiden Capes initialisiert sind die Systemuhr mithilfe der Uhrzeit des RTC Capes gestellt. Die Uhrzeit muss zu jeder Zeit korrekt sein, damit das System fehlerfrei funktioniert.

Nach der Konfiguration wird die Qt Steuerungssoftware ausgeführt. Sie wird dabei weiterhin vom systemd überwacht und im Fehlerfall neu gestartet.

Alle Log-Nachrichten werden dabei in die Logdatei des systemd geschrieben. Die dadurch anfallenden Datenmengen überschreiten jedoch schnell die Kapazitäten der Hardware des BeagleBone Black. Weshalb zur Organisation der Logdateien die im Linux Betriebssystem mitgelieferte Software logrotate [Log] eingesetzt wird. Logrotate übernimmt die automatische Rotation, Kompression und Löschung von Logdateien.

```
1 /var/log/daemon.log
2 {
3     size 100M
4     weekly
5     rotate 4
6     compress
7 }
```

Quellcode 5.3: Logrotate

Im Quellcode 5.3 ist die entsprechende Konfiguration zu sehen. Die maximale Größe der Logdateien ist dabei auf 100MB beschränkt um den internen Speicher von 4GB nicht zu sehr zu belasten. Wird diese Größe überschritten, wird die Logdatei rotiert. Dabei wird ein komprimiertes Backup der aktuellen Logdatei erstellt und eine neue angelegt. Des Weiteren wird die Logdatei einmal pro Woche unabhängig von der Größe rotiert. Bei der Rotation werden die letzten vier Logdateien gespeichert. Kommt eine Weitere dazu, wird die älteste gelöscht. Die maximale Größe von 100MB stellt dabei nur selten das Limit dar. So stehen die Logdateien für etwa 4 Wochen rückblickend zur Verfügung.

5.2. RS232 Kommunikation

Die Kommunikation von dem Mess-Server mit den Mess-Slaves erfolgt sequentiell. Der Mess-Server stellt eine Anfrage und wartet anschließend für eine festgelegte Zeit auf eine Antwort. Die Antwort enthält bei fehlerfreier Kommunikation das invertierte Read/Write-Bit, sowie die gleiche RS232 Adresse und den selben Befehl/Unterbefehl wie bei der Anfrage. Bei einer Leseanfrage enthält die Antwort zusätzlich noch die angeforderten Daten.

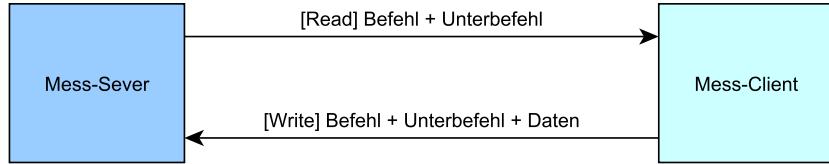


Abb. 5.2.1.: RS232 Kommunikation: Leseanfrage

Um Fehler bei der Kommunikation zu vermindern und die Erfolgschancen zu erhöhen, müssen fehlerhafte Übertragungen erkannt und wenn möglich korrigiert werden. Die Erkennung von Fehlern geschieht durch die Auswertung der Antwort eines Mess-Slaves.

Dabei können drei Szenarien auftreten:

- Keine Antwort: Innerhalb des Zeitlimits wurde keine Antwort empfangen.
- Falsche Antwort: Innerhalb des Zeitlimits wurde eine Antwort empfangen, jedoch entspricht sie nicht den Erwartungen.
- Prüfsummen-Fehler: Innerhalb des Zeitlimits wurde eine Antwort empfangen, jedoch ist die Prüfsumme nicht korrekt.

Zur Korrektur dieser Fehler werden Anfragen bis zu fünf mal in einem festgelegten zeitlichen Abstand wiederholt gesendet. Sollte die Übertragung weiterhin fehlerhaft sein, ist der Mess-Slave entweder nicht erreichbar oder es gibt einen Adresskonflikt auf dem RS232 Bus. Bei einem Adresskonflikt haben zwei oder mehr Mess-Slaves die selbe RS232 Adresse. Das führt dazu, dass beide Mess-Slaves auf die selben Anfragen reagieren und ihre Antworten sich auf dem RS232 Bus überlagern. Was die Übertragung unleserlich macht.

5.3. Datenbank

Die Datenbank muss aus dem Netzwerk erreichbar sein, damit die Messdaten von einem PC-Client abgerufen werden können. Bei einem MySQL Server ist dies jedoch nicht von Grund auf gegeben. Standardmäßig akzeptiert der MySQL Server nur Anfragen über die Localhost Adresse (IP-Adresse: 127.0.0.1). Deshalb müssen die Zugriffsrechte erst erteilt werden.

```
GRANT ALL ON *.* to root@ '%' IDENTIFIED BY 'toor';
```

Quellcode 5.4: MySQL Zugriffsrechte

Mit dem Befehl im Quellcode 5.4 erhalten alle Nutzer die sich mit dem Benutzernamen *root* und dem Passwort *toor* identifizieren, vollen Zugriff auf alle Datenbanken. Somit ist es möglich von überall aus eine Verbindung zum Datenbankserver aufzubauen und sämtliche Daten abzurufen.

Über einen langen Betriebszeitraum können sich große Mengen an Daten in der Datenbank ansammeln. Das hat zur Folge, dass die SQL-Abfragen nach bestimmten Daten sehr lange dauern können. Um dem vorzubeugen werden Indexe für häufig abgefragte Daten erstellt.

```
1 CREATE INDEX measurement_time_idx on Measurement(Time);
```

Quellcode 5.5: MySQL Index

Der Quellcode 5.5 zeigt die Erzeugung eines Indexes. Der Index mit dem Namen *measurement_time_idx* wird für die Spalte *Time* in der Tabelle *Measurement* angelegt. Vor Allem in der Tabelle *Measurement* können sehr schnell große Mengen an Daten anfallen.

Der Index für die Spalte *Time* ist besonders gut geeignet. Denn um herauszufinden wann die letzte Messung durchgeführt wurde, muss aus der Tabelle *Measurement* das Messergebnis mit dem aktuellsten Zeitstempel ermittelt werden. Diese Abfrage kann bei 2 Millionen Einträgen ohne Index bis zu 2 Minuten dauern. Mit dem Index kann sie auf 4 Sekunden reduziert werden.

Wenn ein Board gelöscht wird, werden alle dazugehörigen Datenbankeinträge aus den anderen Tabellen gelöscht. Das soll unnötigen Daten in der Datenbank vorbeugen. Dafür sind die Fremdschlüssel in allen Tabellen außerhalb der *D_Board* Tabelle mit dem Constraint *ON DELETE CASCADE* versehen. Es sorgt für eine Löschung des betroffenen Eintrages, falls das referenzierte Attribut gelöscht wird. Da die Tabellen *Setting* und *Pattern* keine eigenen Verweise zu der Tabelle *D_Board* haben, ist für diese Kaskadierung ein Trigger notwendig.

Ein Trigger ist die Reaktion auf ein Ereignis. Das Ereignis ist die Löschung eines Eintrags in der *D_Board* Tabelle. Daraufhin sollen die dazugehörigen Einträge in der *Setting* und der *Pattern* Tabelle gelöscht werden. Dafür wird ein Trigger erstellt (siehe Quellcode 5.6).

```
1 Create Trigger onBoardDelete before delete on D_Board
2   for each row
3     Begin
4       delete from Setting
5         where p_id = old.f_Settingid ;
6       delete from Pattern
7         where p_id = old.f_Patternid ;
8     end ;
```

Quellcode 5.6: onBoardDelete Trigger

5.4. Webinterface

Das Webinterface soll nach der Verbindung über den WLAN Hotspot ohne eine spezielle Adresse manuell im Webbrower aufrufen zu müssen erreichbar sein. Dafür wird der DNS Server dnsmasq so konfiguriert, dass alle Anfragen auf den Webserver umgeleitet werden.

```
1 interface=wlan0
2 dhcp-range=wlan0,192.168.10.10,192.168.10.50,4h
address=/#/192.168.10.1
```

Quellcode 5.7: DHCP/DNS Konfiguration

Im Quellcode 5.7 ist die Konfiguration des DNS/DHCP Servers zu sehen. Er verwaltet die Verbindungen über das *wlan0* Interface, was unter Linux dem WLAN Dongle entspricht. Das System selbst ist mit der IP Adresse 192.168.10.1 konfiguriert. Alle Anfragen werden von dem DNS Server auf diese Adresse umgeleitet. Für die IP-Adressvergabe des DHCP Servers ist der Bereich von 192.168.10.10 bis 192.168.10.50 vorgesehen.

In Abbildung 5.4.1 ist das Webinterface zu sehen. Es zeigt die Messdaten die für einen Mess-Slave aufgenommen wurden. Die Daten stellen dabei keine echten Messergebnisse dar. Sie dienen lediglich der Veranschaulichung.

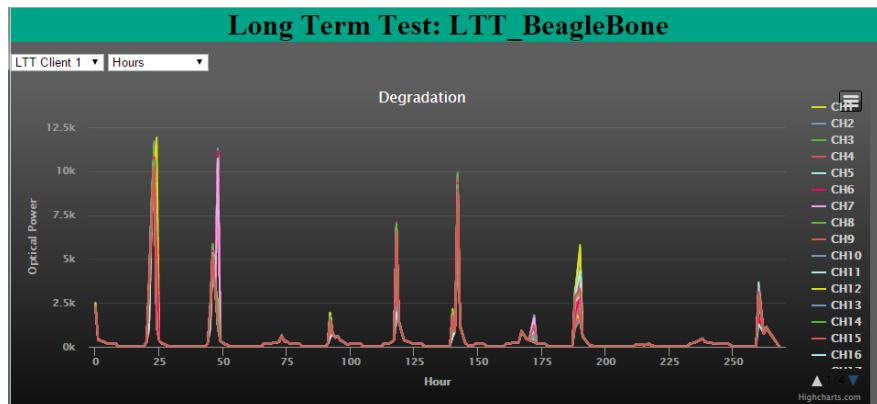


Abb. 5.4.1.: Webinterface

Die Daten für das Webinterface werden direkt aus der MySQL Datenbank bezogen und mittels des Javascript Diagramm Frameworks Highcharts [Hig] grafisch aufbereitet.

5.5. BeagleBone Black Benutzeroberfläche

Die GUI soll die Anforderung der Statusüberwachung aus Abschnitt 4.2.2 erfüllen und ist dafür in vier Tabs unterteilt. Am oberen Rand der GUI (siehe u.a. Abbildung 5.5.1) wird das aktuelle Datum und die aktuelle Uhrzeit angezeigt, sowie die derzeitige IP Adresse und der aktuelle Name. Am unteren Rand wird die derzeit ausgeführte Aktion in einer Statusnachricht ausgegeben.

Status Tab

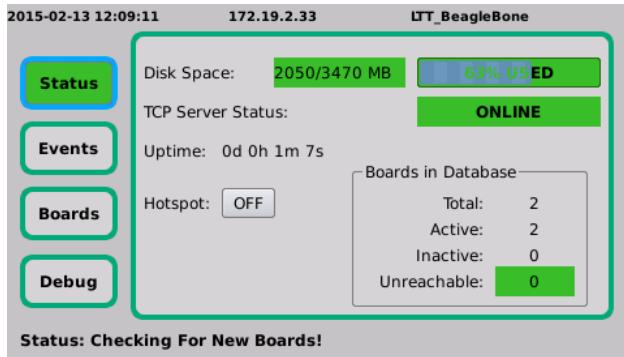


Abb. 5.5.1.: Mess-Server GUI: Status Tab

Das Erste ist das Status-Tab (siehe Abbildung 5.5.1). Es zeigt die wichtigsten Statusdaten wie verfügbarer Speicher, die aktuelle Laufzeit und TCP Server Status an. Um auf einen Blick den Status des Systems zu erkennen wird mittels der Farben grün und rot ein positiver bzw. negativer Status signalisiert. Außerdem kann hier der WLAN Hotspot aktiviert und deaktiviert werden.

Events Tab

ID	Label	Description
0	LTT Client 1	New Board Created! 2015-02-13 12:08:45
1	LTT Client 2	New Board Created! 2015-02-13 12:09:03

Abb. 5.5.2.: Mess-Server GUI: Events Tab

Im Events Tab (siehe Abbildung 5.5.2) werden wichtige Ereignisse dargestellt. Es sollen zwei Ereignisse dargestellt werden. Das bisher einzige Ereignis ist das Erkennen und Eingliedern eines neuen Mess-Slaves. Es wird ausgelöst, wenn ein Mess-Slave erfolgreich am System angemeldet und eingegliedert wurde.

Boards Tab

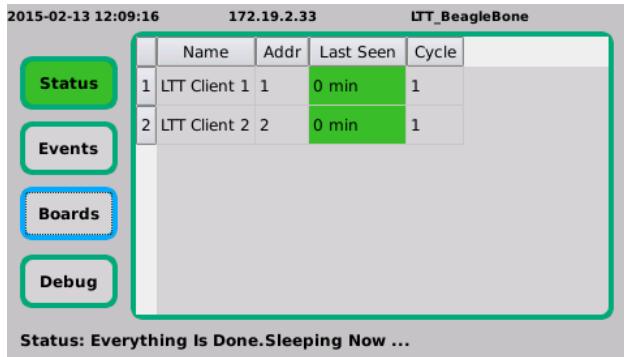


Abb. 5.5.3.: Mess-Server GUI: Boards Tab

Das Boards Tab (siehe Abbildung 5.5.3) zeigt die derzeit aktiven Mess-Slaves in einer Liste an. Als aktiv werden Mess-Slaves bezeichnet, die mit einer gültigen Adresse in der Datenbank eingetragen sind. Angezeigt werden der Name, die Adresse, die vergangene Zeit seit dem es das letzten mal erfolgreich kontaktiert wurde und die Anzahl der erfolgreich aufgenommenen Messzyklen. Sollte ein Mess-Slave über längere Zeit nicht ansprechbar sein, wird dem Nutzer durch eine rote Markierung das Problem signalisiert. Außerdem ist der Status der Mess-Slaves ablesbar.

Debug Tab



Abb. 5.5.4.: Mess-Server GUI: Debug Tab

Der Zweck des Debug Tabs (siehe Abbildung 5.5.4) ist es, die im Programmcode erzeugten Nachrichten anzuzeigen. Somit soll es möglich sein, Fehler einfacher zu erkennen. Die Anzeige dient dabei zur Vor-Ort-Analyse von Fehlern. Die selben Informationen werden auch intern auf dem Mess-Slave in einer Logdatei gespeichert.

6. Testen und Validieren

Da das System in einem Langzeit-Teststand eingesetzt wird, ist die Zuverlässigkeit und die Betriebsfähigkeit über lange Zeiträume besonders wichtig. Es darf keine großen Performanceeinbußen oder lange Ausfälle beim Betrieb geben. In diesem Kapitel wird auf die Tests zur Sicherstellung dieser Kriterien und die Grenzen des Systems eingegangen.

6.1. Speicherverwaltung

Ein häufiger Grund für Performanceeinbußen sind Fehler in der Speicherverwaltung. Meist treten dabei Speicherlecks (englisch: memory leak) auf. Speicherlecks sind Fehler in der Programmierung der Speicherverwaltung, wodurch Speicher belegt, aber ungenutzt ist und nicht wieder freigegeben wird. Dabei kommt es zu einer immer größer werdenden Speichernutzung, bis der gesamte Speicher des Systems ausgelastet ist und es sich stark verlangsamt oder sogar abstürzt.

Bei normalen Mikrocontrollern wird die Speicherverwaltung komplett vom Programmierer umgesetzt. Beim BeagleBone Black übernimmt das Linux-Betriebssystem einen großen Teil der Verwaltung. Jedoch kann es trotzdem noch zu Speicherlecks kommen. Der Hauptgrund ist auf dem Heap durch `malloc()` oder `new` reservierter Speicher der nicht wieder durch `free()` oder `delete` freigegeben wird.

Um Speicherlecks auszuschließen wird die Steuerungssoftware mittels Valgrind [Val] auf Speicherfehler überprüft. Valgrind ist ein Programm zu Speicherdiagnose. Es ist im Grunde eine virtuelle Maschine, in der das zu testende Programm ausgeführt wird. So läuft das zu testende Programm niemals auf der realen CPU, sondern lediglich in Valgrind. Dadurch können alle Aufrufe und Aktionen eines Programmes überwacht werden. Aus diesen Daten ermittelt Valgrind, ob Speicherfehler vorliegen.

Bei der Überprüfung der Steuerungssoftware kommt es lediglich zu einigen falsch positiven (englisch: false positive) Ergebnissen. Sie sind auf die Beschaffenheit der Qt C++ Klassenbibliothek zurückzuführen und können ignoriert werden. Somit kann davon ausgegangen werden, dass die Steuerungssoftware keine Fehler in der Speicherverwaltung aufweist.

6.2. Fehlerfälle

Ein Fehlerfall ist der Stromausfall. Nach einem Stromausfall muss das System binnen kürzester Zeit wieder einsatzbereit sein. Zur Sicherstellung dieser Eigenschaft wurden 20 Stromausfälle durch trennen und anschließendes Neu-verbinden der Stromversorgung simuliert. Bei 20 Versuchen ist das System ohne Probleme neu gestartet. Das BeagleBone Black fährt bei Anschluss einer Stromversorgung automatisch hoch. Auch die Uhrzeit und das Datum werden durch den Einsatz des RTC Capes beibehalten.

Ein weiteres Szenario, ist das Abziehen eines Mess-Slaves im laufenden Betrieb. Der Mess-Server erkennt diesen Fehlerfall und teilt den Fehler über die Benutzeroberfläche dem Administrator mit. Der Mess-Server versucht weiterhin den Mess-Slave in regelmäßigen Abständen zu erreichen. Der Versuch der Kontaktaufnahme erfolgt so lange, bis der Mess-Slave durch das Löschen der RS232 Adresse aus der Datenbank vom System abgemeldet , oder der Mess-Slave wieder angeschlossen wird und somit erreichbar ist. In diesem Fall werden wieder Messdaten in den konfigurierten Intervallen erfasst.

6.3. Testaufbau

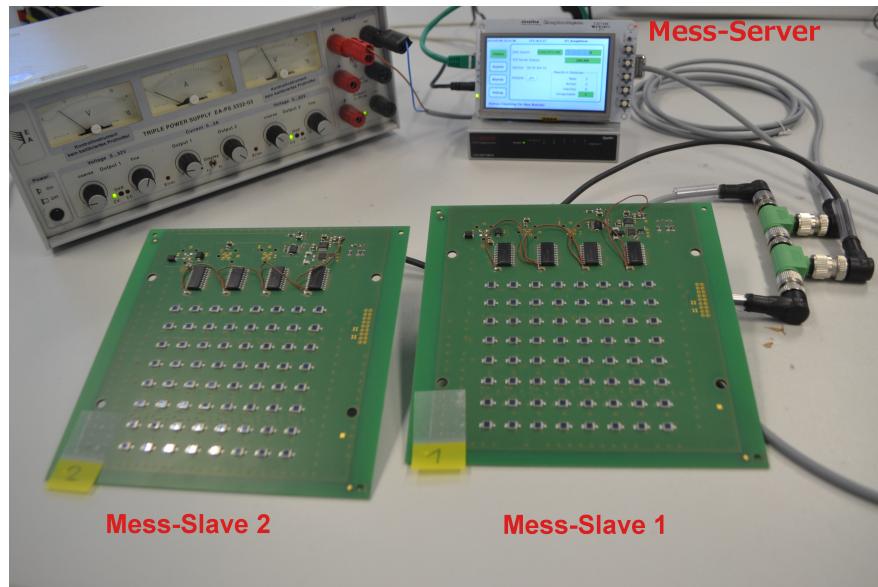


Abb. 6.3.1.: Testaufbau

In einem Testaufbau werden zwei Mess-Slaves an einen Mess-Server angeschlossen und in Betrieb genommen.

Beiden Mess-Slaves wurde vorher die RS232 Adresse 0 zugewiesen, damit sie sich bei dem Mess-Server anmelden können. Außerdem soll eine Messung pro Stunde durchgeführt werden. Die Mess-Slaves werden nach einander an den RS232 Bus angeschlossen und vom Mess-Server erkannt. Es ist dabei wichtig, zu warten bis der erste Mess-Slave erfolgreich angemeldet ist, bevor der zweite Mess-Slave angeschlossen ist. Würden beide Mess-Slaves gleichzeitig angeschlossen werden, gebe es einen Adresskonflikt, da beide Mess-Slaves die RS232 Adresse 0 besitzen. Es wäre keine Kommunikation möglich.

Nach der erfolgreichen Anmeldung am Mess-Server, werden in den eingestellten Messintervallen die Messdaten aufgezeichnet. Über die Desktop-Anwendung des PC-Clients kann auf die Mess-Slaves zugegriffen werden. Die Performance des Mess-Servers wird dabei nach durchgängigem Betrieb über 10 Tage nicht beeinflusst.

6.4. Grenzen

In der Theorie können 127 Mess-Slaves gleichzeitig an einem Mess-Server betrieben werden. Diese Grenze wird durch den RS232 Adressraum gegeben. Jedoch ergeben sich in der Praxis einige Einschränkungen. Die durchschnittliche Dauer für die Erfassung der Messdaten für einen Mess-Slave beträgt 4,64 Sekunden. Dieser Wert wurde aus 25 Messungen gemittelt.

Wenn 127 Mess-Slaves angeschlossen sind und für jeden eine Messungen durchgeführt werden soll, ergibt sich folgende Zeit für einen gesamten Durchlauf.

$$\text{AnzahlDerMessSlaves} * \text{DauerProMessung} = \text{Durchlaufzeit}$$

$$127 * 4,64s = 589,28s$$

Das bedeutet, dass bei 127 Mess-Slaves das minimal Messintervall 589 Sekunden beträgt.

Der typische Betrieb eines Mess-Slaves sieht ca. 365 Messungen pro Jahr vor. Bei 64 Prüfobjekten ergibt das 23.360 Messwerte in einem Jahr. Bei zwei Millionen Messwerten in der Datenbank, wird diese in der Performance spürbar negativ beeinflusst. Eine typische Abfrage dauert so bei zwei Millionen Einträgen bereits bis zu fünf Sekunden, im Vergleich zu 160 Millisekunden bei 35 Tausend Einträgen. Nehme man diese 2 Millionen als obere Grenze ergibt sich folgende maximale Anzahl von Betriebsjahren.

$$\text{MaxMesseinträge}/\text{MessungenProJahr} = \text{MaxBetriebsjahre}$$

$$2.000.000/23360 = 85,6$$

Jedes Prüfobjekt soll für 20.000 Stunden getestet werden, was 2,28 Jahren entspricht. Daraus kann die maximale Anzahl an Mess-Slaves ermittelt werden.

$$\text{MaxBetriebsjahre/MessDauer} = \text{AnzahlAnMessSlaves}$$

$$88,6/2,28 = 37,5$$

Es ergibt sich eine maximale Anzahl von 37 Slaves pro Mess-Server. Nach 37 abgeschlossenen Langzeittests müssen durch den Administrator alte Messdaten gelöscht werden um Raum für neue zu schaffen.

In der Praxis wird pro Temperaturschrank ein Mess-Server eingesetzt. In jedem dieser Temperaturschränke können bis zu zehn Mess-Slaves gleichzeitig betrieben werden. Daraus ergibt sich eine Betriebsdauer von 8,86 Jahren, bevor von einem Administrator in das System eingegriffen werden muss.

7. Zusammenfassung und Ausblick

Dieses abschließenden Kapitel bietet eine Zusammenfassung dieser Arbeit, sowie einen Ausblick für mögliche Verbesserungen und Erweiterungen für die Zukunft.

7.1. Zusammenfassung

Im Laufe dieser Arbeit wurde eine Steuereinheit für die Ansteuerung eines Langzeittests zur Erfassung des Degradationsverhaltens von LEDs entwickelt. Zur Lösung dieser Aufgabe wurde in Kapitel 2 auf die Grundlagen der Qt C++ Klassenbibliothek und einer Datenbank eingegangen. Außerdem wurde sich mit der Definition der Begriffe *Degradation* und *Embedded-Linux-System* auseinander gesetzt.

In Kapitel 3 wurde das Szenario, welches als Motivation für diese Arbeit gilt, näher beleuchtet. Des Weiteren wurden die Akteure des Systems analysiert und daraus die Anforderung abgeleitet.

Im folgenden Kapitel 4 wurde zunächst das BeagleBone Black mit seinen Erweiterungen als Hardwarekomponente vorgestellt. Anschließend wurde das Design der Softwarekomponenten erklärt. Die Softwarekomponenten teilen sich dabei in zwei Teile. Zum Einen die Steuerungssoftware, die in C++ unter Zuhilfenahme der Qt Klassenbibliothek entwickelt wurde. Sie beinhaltet die Messdatenerfassung, die Mess-Slave Verwaltung, einen TCP/UDP-Server und eine Statusüberwachung. Zum Anderen die Linux Softwarepakete, welche sich aus dem MySQL-Datenbankserver, dem lighttpd Webserver und dem hostapd WLAN Hotspot zusammensetzen.

Auf Implementierung des Designs wurde in Kapitel 5 eingegangen. Dabei ging es um die Konfiguration des Linux-Systems und der Datenbank, sowie die Umsetzung der RS232 Kommunikation. Außerdem wurde die Implementierung der Statusüberwachung in Form einer grafischen Benutzeroberfläche näher beschrieben. Des Weiteren wurde die Konfiguration des Webinterfaces gezeigt.

Zum Abschluss wurde in Kapitel 6 die Lösung auf Fehler und Funktion geprüft. Dabei wurden mögliche Fehlerfälle aufgezeigt und die Grenzen des Systems beschrieben.

7.2. Ausblick

In Zukunft könnte der Mess-Server in Funktionalität und Sicherheit erweitert werden. Der externen Zugriffe auf den MySQL Datenbankserver kann durch ein Webinterface abstrahiert werden. Das würde vor allem die Sicherheit des Datenbankservers erhöhen, da alle Zugriffe durch eine Webschnittstelle überwacht werden könnten.

Zur Erweiterung der Funktionen könnte eine lokale Ausgabe der Messdaten auf eine Micro-SD-Karte über den Micro-SD-Kartenslot ausgegeben werden. Des Weiteren wäre denkbar eine Logik zur Speicherung der Messdaten zu implementieren. Diese würde anstatt in festgelegten Intervallen, die Messwerte nur bei Erreichen vorher festgelegter Kriterien aufnehmen. Wodurch redundante Messdaten vermieden werden könnten.

Auch eine bessere Statusüberwachung wäre möglich. So könnte das System bei Ausfällen und Fehlern eine Nachricht per E-Mail an den Administrator senden. Denn im aktuellen Zustand muss der Administrator selbst aktiv den Status des Systems abrufen.

Literaturverzeichnis

- [Ben05] BENDER, K.: *Embedded Systems: Qualitätsorientierte Entwicklung*. Springer, 2005.
– ISBN 9783540273707
- [Bog08] BOGUS, A.: *Lighttpd*. Packt Publishing, Limited, 2008 (From Technologies to Solutions). – ISBN 9781847192110
- [Cap] CAPEMGR: *Capemgr*. <http://elinux.org/Capemgr>. – Zugriff am 02.03.2015
- [Hig] HIGHSOFT: *Highcharts*. <http://www.highcharts.com/>. – Zugriff am 06.03.2015
- [Log] LOGROTATE: *Logrotate*. <http://www.linux-praxis.de/lpic1/manpages/logrotate.html>. – Zugriff am 02.03.2015
- [QtP] QTPROJECT: *Qt*. <http://qt-project.org/>. – Zugriff am 02.03.2015
- [SGD09] SCHRÖDER, J. ; GOCKEL, T. ; DILLMANN, R.: *Embedded Linux*:. Springer, 2009 (X. systems. press Series). – ISBN 9783540786191
- [SSH10] SAAKE, G. ; SATTLER, K.U. ; HEUER, A.: *Datenbanken: Konzepte und Sprachen*. mitp, 2010 (Biber-Buch). – ISBN 9783826690570
- [ubu] UBUNTUUSERS: *PuTTY*. <http://wiki.ubuntuusers.de/PuTTY>. – Zugriff am 02.03.2015
- [Val] VALGRINDEVELOPERS: *Valgrind*. <http://valgrind.org/>. – Zugriff am 09.03.2015
- [Wik] WIKIPEDIA: *Qt Creator*. http://de.wikipedia.org/wiki/Qt_Creator. – Zugriff am 06.03.2015
- [Wir] WIRELESS, Linux: *hostapd Linux documentation page*. <https://wireless.wiki.kernel.org/en/users/documentation/hostapd>. – Zugriff am 25.02.2015

- [YMBYG08] YAGHMOUR, K. ; MASTERS, J. ; BEN-YOSSEF, G. ; GERUM, P.: *Building Embedded Linux Systems*. O'Reilly Media, 2008. – ISBN 9780596555054
- [ZSG11] ZHOU, Rensheng R. ; SERBAN, Nicoleta ; GEBRAEEL, Nagi: Degradation modeling applied to residual lifetime prediction using functional data analysis. In: *Ann. Appl. Stat.* 5 (2011), 06, Nr. 2B, 1586–1610. <http://dx.doi.org/10.1214/10-AOAS448>. – DOI 10.1214/10-AOAS448

Anhang A

Hardware

A.1. BeagleBone Black

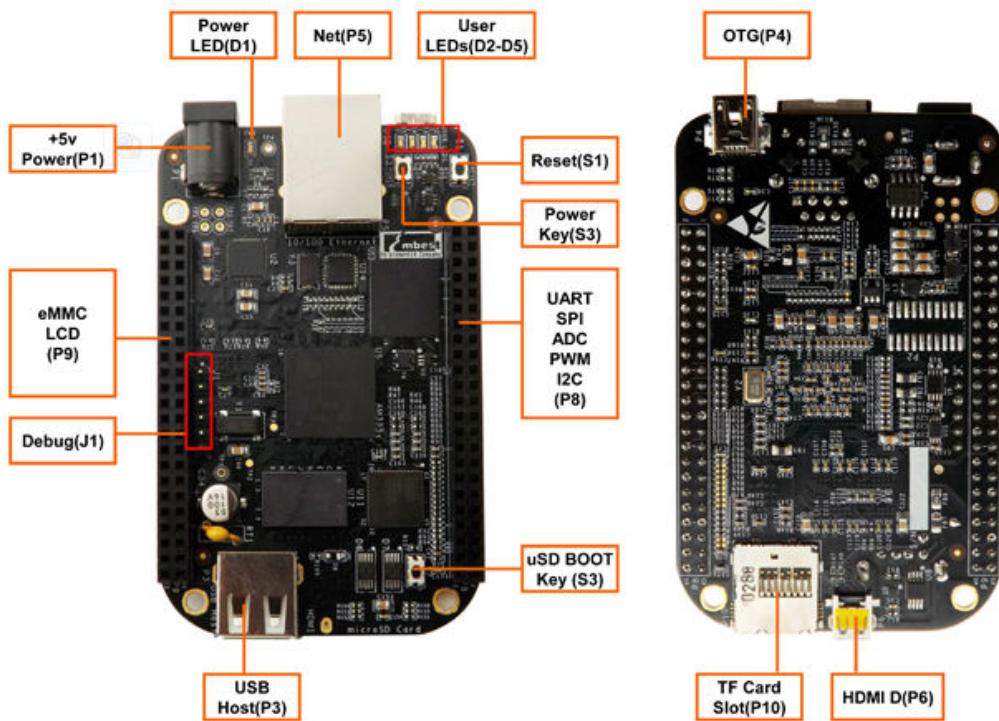
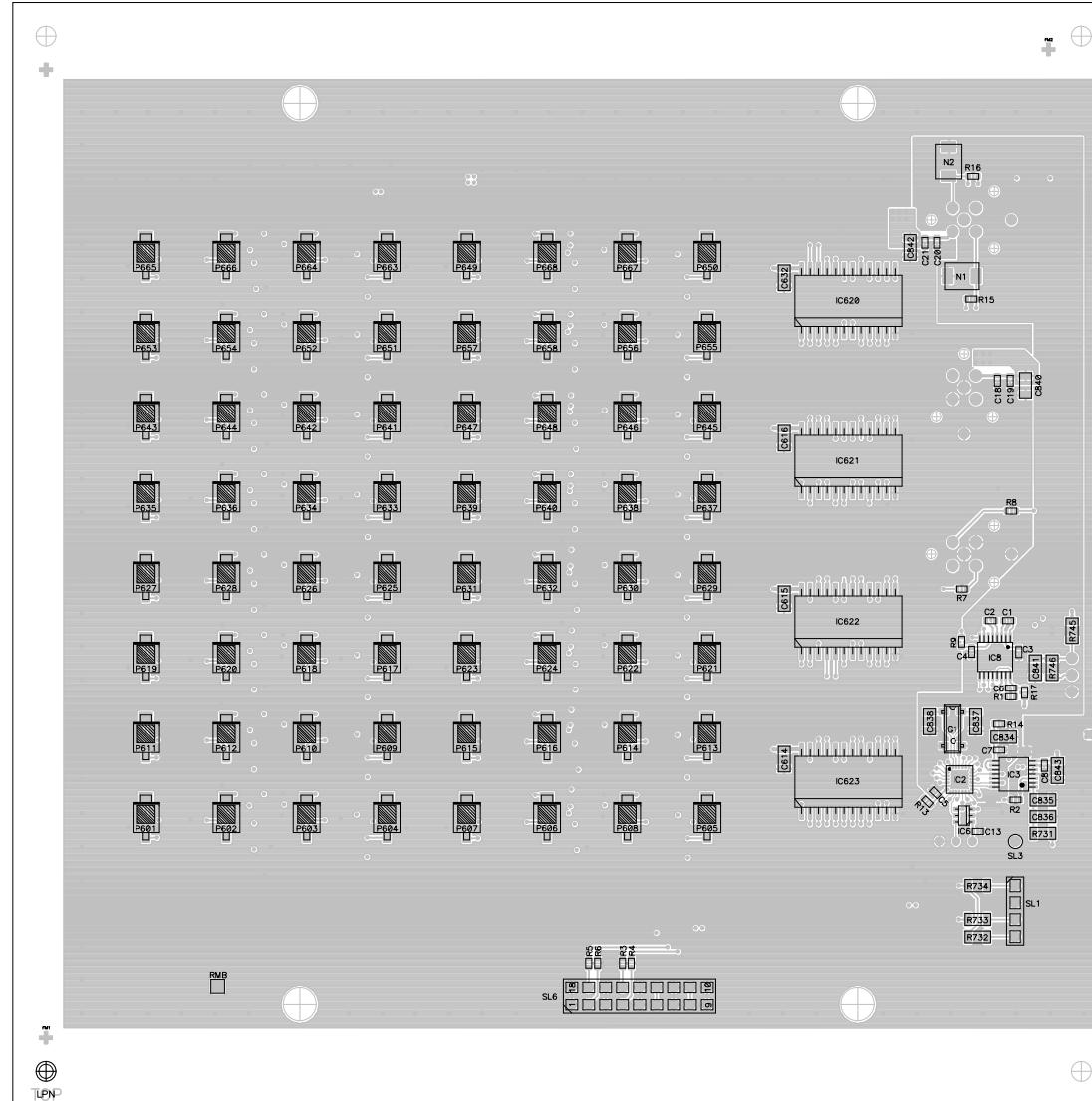


Abb. A.1.1.: BeagleBone Black

A.2. Mess-Slave



Confidential according to ISO 1

 PEPPERL+FUCHS

Only valid as long as released in EDM or with a valid production documentation

assembly plan top

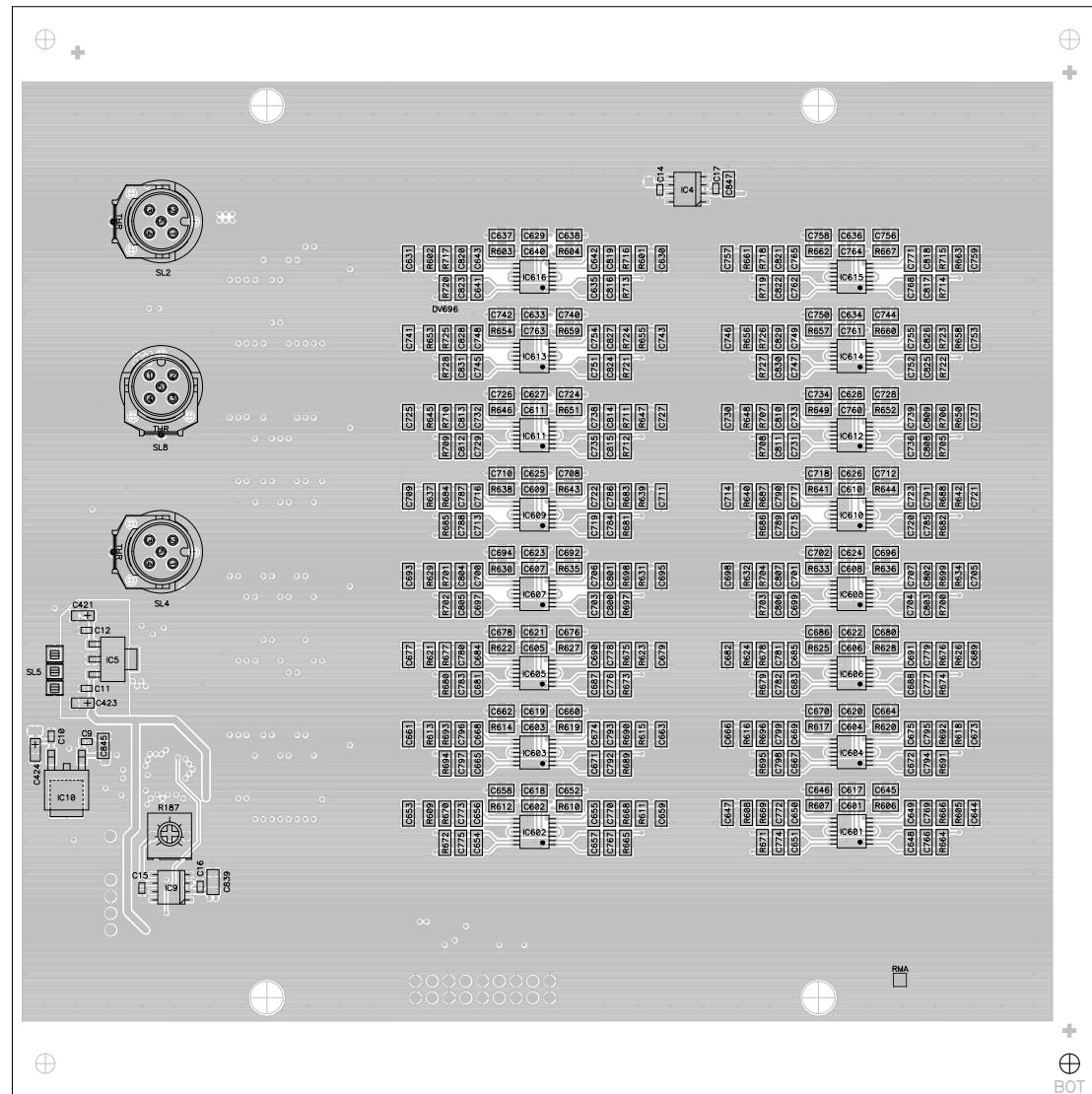
Degradation Testboard Receiver

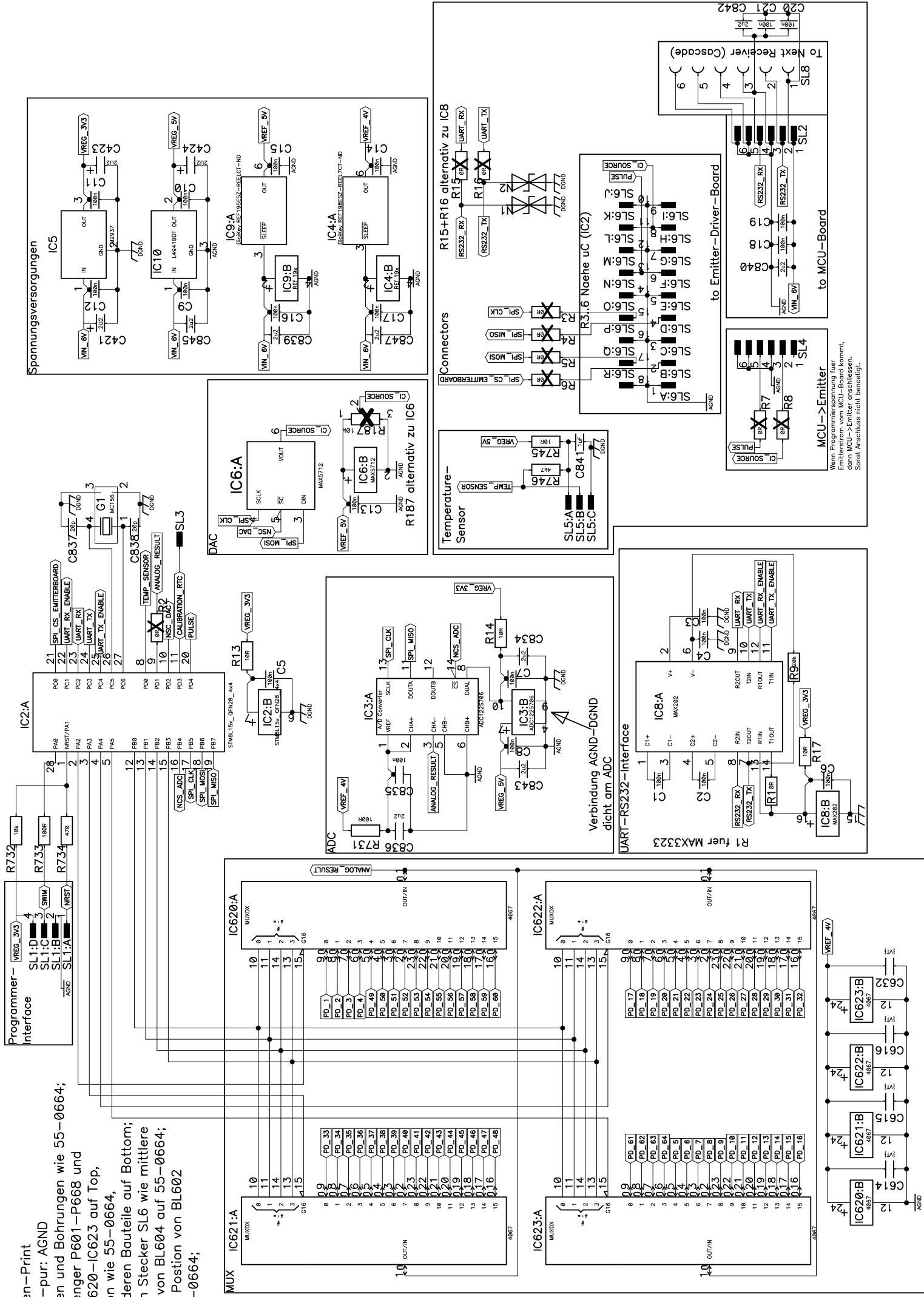
scale: --- date: 17.10.14

.TBO 03-AC-13A

-- 85 A605A

rm Released EDM sheet 1 of 2





Anhang B

Inhalt der beigefügten CD-ROM

Thesis Tim Nieter.pdf	Diese Arbeit als PDF Dokument
Hardware/	Enthält die Schaltungen und Layouts
Quellcode/Datenbank	Quellcode zum erzeugen der Datenbank
Quellcode/Mess-Slave	Quellcode des Mess-Slaves
Quellcode/Mess-Server	Quellcode des Mess-Servers
Quellcode/PC-Client	Quellcode des PC-Clients
Tools/	Bei der Entwicklung verwendete Werkzeuge

