



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

- Nicht öffentlich -

Erstbetreuer: Prof. Dr. F. Bauernöppel

Zweitbetreuer: G. 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)

Sperrvermerk

Die nachfolgende Bachelorarbeit mit dem Titel „Entwicklung eines vollautomatisierten Embedded-Linux-Systems zur Ansteuerung und Auswertung eines Langzeittests“ enthält vertrauliche Daten der Pepperl+Fuchs GmbH. Veröffentlichungen oder Vervielfältigungen der Arbeit – auch nur auszugsweise – sind ohne ausdrückliche Genehmigung der Pepperl+Fuchs GmbH nicht gestattet. Die Arbeit ist lediglich den Korrektoren sowie den Mitgliedern der Prüfungskommission zugänglich zu machen.

Vorwort

Die Netze sind in einer vernetzten Welt von netzartiger Bedeutung. Ohne Netze ist

Die vorliegende Arbeit ist auf Basis des Latex-Templates zu [1] erstellt worden.

[1] T. Gockel. Form der wissenschaftlichen Ausarbeitung. Springer-Verlag, Heidelberg, 2008. Begleitende Materialien unter <http://www.formbuch.de>.

Inhaltsverzeichnis

Abbildungsverzeichnis	II
Tabellenverzeichnis	III
Quellcodeverzeichnis	IV
Abkürzungsverzeichnis	V
1 Einleitung	1
1.1 Motivation	1
1.2 Zielsetzung	1
1.3 Aufbau der Arbeit	1
2 Grundlagen	2
2.1 Degradation	2
2.2 Qt	3
2.2.1 Signale und Slots	3
2.2.2 Entwicklungsumgebung	4
2.3 Datenbank	5
2.4 Embedded Linux System	6
3 Anforderungsanalyse	7
3.1 Szenario	7
3.2 Analyse	8
3.2.1 Mess-Client	8
3.2.2 Mess-Server	9
3.3 Anforderungen	9
4 Design	12
4.1 Übersicht	12
4.2 Hardware	14
4.3 Software	15
4.3.1 Messdatenerfassung	16

4.3.2	Benutzeroberfläche	18
4.3.3	RS232 Kommunikation	19
4.3.3.1	Ethernet Kommunikation	20
4.3.4	Webzugriff	20
4.3.5	Datenbank	20
4.4	RS232 Protokoll	22
4.4.1	Aufbau	23
4.4.2	Befehle und Unterbefehle	26
5	Testen und Validieren	27
6	Fazit und Ausblick	28
Literaturverzeichnis		i

Abbildungsverzeichnis

2.1.1	Beispiel Degradation	2
2.2.1	QtCreator Version 2.0.1	4
2.4.1	Links:Kommunikation zwischen Applikation und Hardware. Rechts: Beispiel Kommunikation zwischen Applikation und UART über den Kernel.	6
3.1.1	Szenario	7
3.2.1	Mess-Client Oberseite	8
3.2.2	Mess-Client Unterseite	8
3.2.3	BeagleBone Black	9
4.1.1	Gesamt System	13
4.2.1	Aufbau Mess-Server	14
4.2.2	RS232 Cape	14
4.2.3	RTC Cape	14
4.2.4	LCD Cape	14
4.2.5	Gestapelte Capes	15
4.3.1	Ablaufplan	17
4.3.2	Mess-Server GUI: Status Tab	18
4.3.3	Mess-Server GUI: Events Tab	18
4.3.4	Mess-Server GUI: Boards Tab	19
4.3.5	Mess-Server GUI: Debug Tab	19
4.3.6	Entity Relationship Modell	21

Tabellenverzeichnis

3.3.1	Intervalle	10
3.3.2	Anforderungen	11
4.3.1	Parameter der Messintervalle	16
4.3.2	Ethernet Befehlsliste	20
4.3.3	Tabelle Setting	21
4.4.1	Übertragungsrahmen	23
4.4.2	1. Byte: Adresse & Read/Write	23
4.4.3	Read/Write	23
4.4.4	2. Byte: Rahmenlänge	23
4.4.5	3. Byte: Befehl	24
4.4.6	4. Byte: Unterbefehl	24
4.4.7	5. Byte und folgend: Nutzdaten	24
4.4.8	Letztes Byte: Checksumme	25
4.4.9	RS232 Befehlsliste	26

Quellcodeverzeichnis

2.1	Qt <i>connect</i> -Statement	3
2.2	Qt <i>emit</i> -Statement	3

Abkürzungsverzeichnis

SQL	Structured Query Language	5
DB	Datenbank	5
DBMS	Datenbankmanagementsystem.....	5
DBS	Datenbanksystem	5
ERM	Entity-Relationship-Modell	20
DUT	Device Under Test	7
GUI	Graphical User Interface	3
IDE	Integrated Developement Environment	4
SDK	Software Developement Kit	4
LED	Light-Emitting Diode.....	7
ADC	Analog-Digital-Converter.....	16
UART	Universal Asynchronous Receiver Transmitter	14
RTC	Real Time Clock.....	15

1 Einleitung

In diesem ersten Kapitel wird auf die Motivationen und die Zielsetzung dieser Arbeit eingegangen.

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 erschwerten Bedingungen betrieben und ausgewertet. Nur wenn ein Baulement 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 Zielsetzung

Ziel der Arbeit ist es, ein Embedded-Linux-System zur Ansteuerung und Auswertung eines Langzeittests zu entwickeln.

1.3 Aufbau der Arbeit

2 Grundlagen

Bei der Entwicklung des Teststandes kommen verschiedene Systeme, Schnittstellen und Konzepte zum Einsatz. Dieses Kapitel befasst sich mit der C++ Klassenbibliothek Qt und der RS232 Schnittstelle. Des Weiteren wird auf das Konzept einer Datenbank und den Vergleich 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]:1586.

Die Schwierigkeit liegt in der Feststellung des mechanischen Versagen 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 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.

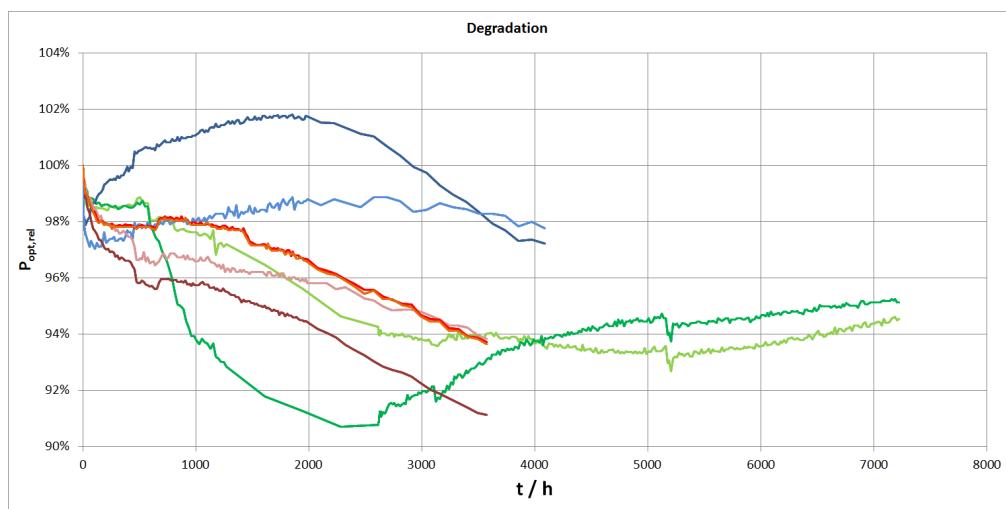


Abb. 2.1.1: Beispiel Degradation

2.2 Qt

Qt 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 Betriebssystem wie Windows, Linux, Android, iOS und einige mehr. Diese 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.

```
1 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();
}
```

Quellcode 2.2: Qt *emit*-Statement

Im Quellcode 2.2 wird das Signal *clicked()* mittels *emit*-Statement ausgelöst und der Slot des verbundenen Objekts aktiviert. Somit ist es möglich beispielsweise die GUI und den Hauptprogrammablauf von einander zu trennen und lediglich über Signale und Slots kommunizieren zu lassen. Dadurch agieren beide Teile 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 auf Linux, Windows als auch Mac OS X zur Verfügung steht. Er kommt mit einem Debugger, einem integrierten grafischen GUI Designer und einem Texteditor mit Funktionen wie Syntax-Hervorhebung und automatischer Vervollständigung.

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

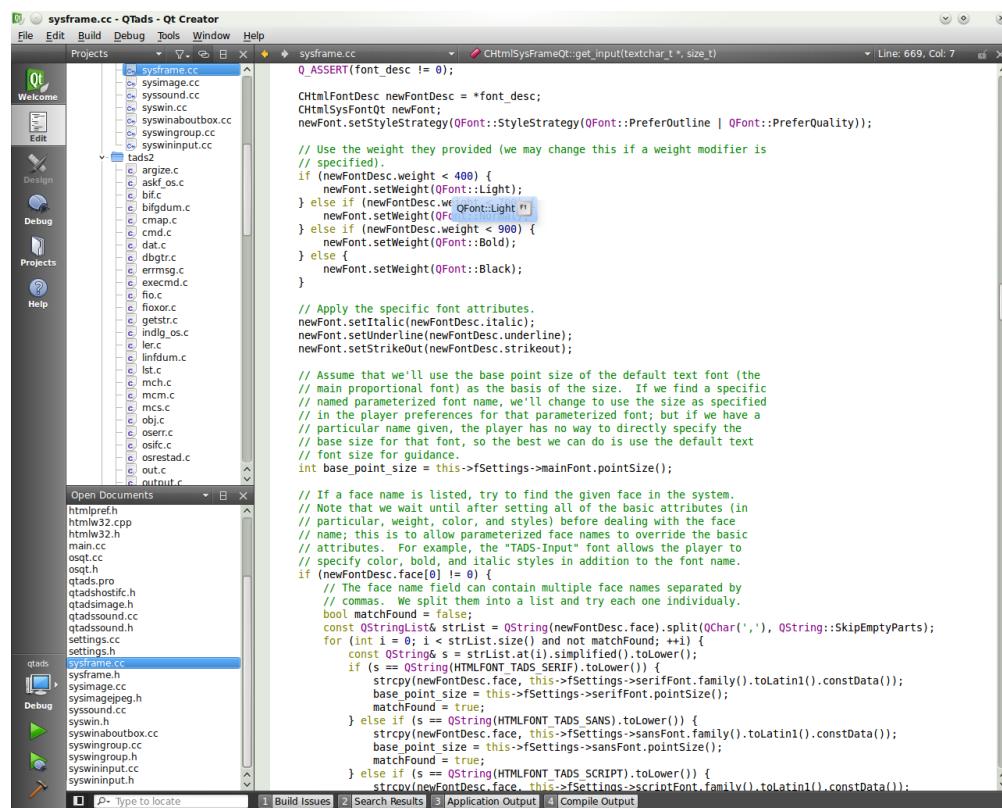


Abb. 2.2.1: QtCreator Version 2.0.1

2.3 Datenbank

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

Für das Speichern der Messdaten und Betriebsparameter 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 Betriebssystems, 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. Außerdem ist das System gut hinsichtlich Performance und Größe zu skalieren. Zusätzlich bietet MySQL viele Möglichkeiten für Performanceanpassungen, 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 das gegebene Projekt eignet sich das relationale DBMS MySQL besser. Auch wenn SQLite einige Vorteile vor allem im eingebetteten Bereich besitzt, ist die fehlende Unterstützung von mehreren Nutzern gleichzeitig ein Ausschlusskriterium.

2.4 Embedded Linux System

Ein eingebettetes System (englisch: embedded system) ist laut [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 Regelaufgaben.

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. [YMBYG08] sagt, dass Embedded Linux dabei typischerweise für ein Komplettsystem, bzw. ein Betriebssystem für ein spezifisches eingebettetes Gerät steht. Außerdem verwende man dabei den normalen Linux-Kernel, der sich lediglich in betriebsbedingten Eigenschaften des Zielsystems unterscheide.

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.

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 kennen (siehe Abbildung 2.4.1).

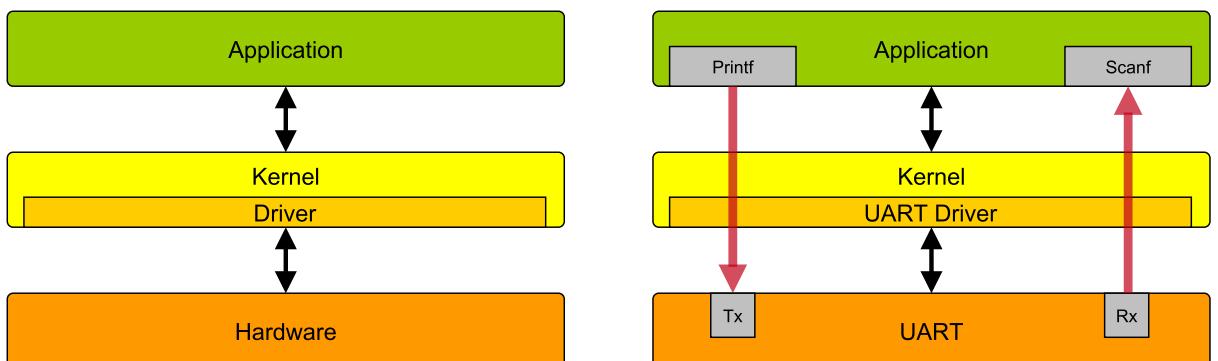


Abb. 2.4.1: Links: Kommunikation zwischen Applikation und Hardware.
Rechts: Beispiel Kommunikation zwischen Applikation und UART über den Kernel.

3 Anforderungsanalyse

Das Ziel dieser Arbeit ist es, einen Teststand entwickeln, der über die Messung der Degradation von optoelektronischen Sendern. Dadurch soll eine Qualifizierung der Bauelemente erfolgen. In diesem Kapitel wird zunächst das Szenario näher beschrieben und anschließend die sich daraus entwickelnden Anforderungen definiert.

3.1 Szenario

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

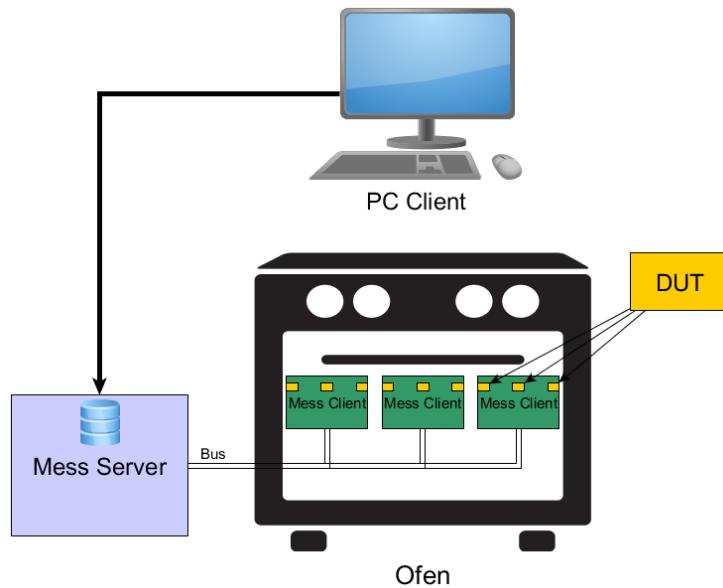


Abb. 3.1.1: Szenario

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

sind über einen Bus mit dem Mess-Server verbunden, welcher alle anfallenden Daten speichert. Von einem PC-Client wird dann auf den Mess-Server zugegriffen um die Daten abzugreifen und grafisch auszuwerten.

3.2 Analyse

Die Akteure des Systems sind der Mess-Server, Mess-Client 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-Client

Das Herzstück des Mess-Clients bildet ein STM8 8-Bit Mikrocontroller der Firma STMicroelectronics. Auf jedem Mess-Client sind 64 DUTs befestigt. In zyklischen Abständen werden die Messdaten der Prüfobjekte aufgenommen und über eine RS232-Schnittstelle zur Verfügung gestellt.

Dieses System war zum großen Teil bereits gegeben, so dass lediglich die Übertragung der RS232 Schnittstelle geregelt werden musste. Dazu wurde ein Protokoll für die Kommunikation entworfen (sieht Abschnitt 4.4).

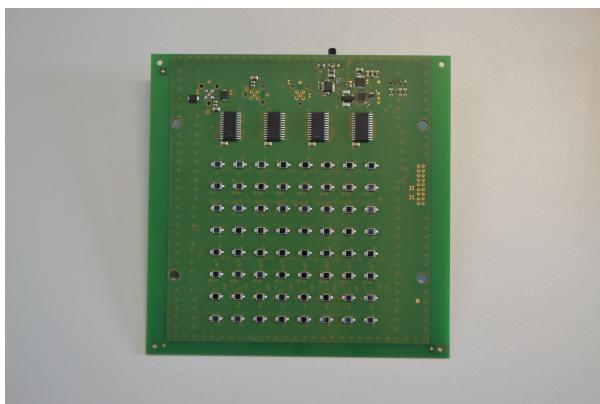


Abb. 3.2.1: Mess-Client Oberseite

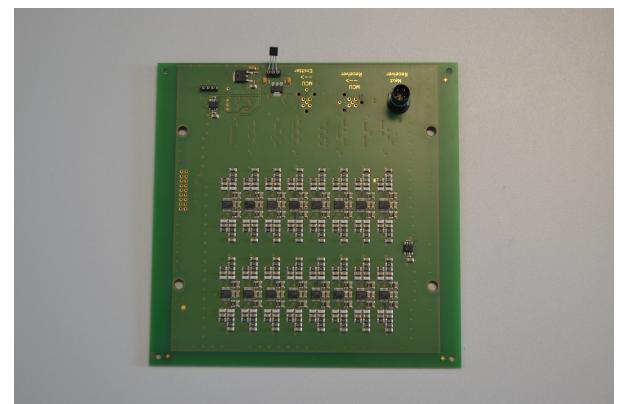


Abb. 3.2.2: Mess-Client Unterseite

—- Besondere RS232 chips, können RX hochohmig schalten. —-

3.2.2 Mess-Server



Abb. 3.2.3: BeagleBone Black

Als Mess-Server wird ein BeagleBone Black von Texas Instruments eingesetzt (siehe Abbildung 3.2.3). Dabei handelt es sich um einen kostengünstigen Einplatinencomputer mit offener Hardware. Damit ist es möglich, das BeagleBone Black auf individuelle Anforderungen anzupassen und selbst herzustellen. Auch gibt es eine große Community, die ständig die Entwicklung vorantreibt. Er 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 Debian-GNU/Linux Betriebssystem zum Einsatz. Damit ist es möglich die umfangreichen Debian-Funktionen wie die Paketverwaltung zu nutzen. Es bietet auch den Vorteil, dass eine große Ähnlichkeit zu PC-Distributionen wie Ubuntu besteht und somit einfacher nutzbar ist (vgl. [SGD09]).

Außerdem sind Linux Mechanismen einfach nutzbar. So kann das RS232 Interface beispielsweise wie eine normale Datei beschrieben und gelesen werden.

3.3 Anforderungen

Folgende Anforderungen werden dabei an das System gestellt:

- Individuelle Parametrierung der DUTs
- Automatische Erfassung der Messdaten
- Fernzugriff auf den Mess-Server

Individuelle Parametrierung der DUTs

Immer 64 DUTs befinden sich auf einem Mess-Client. Dabei sollen verschiedene Parameter für die DUTs berücksichtigt werden. Zum einen sollen die Intervalle in denen Messwerte aufgenommen werden konfigurierbar sein. Dies soll in mindestens 3 verschiedenen Intervallen möglich sein.

Beispiel:

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

Tab. 3.3.1: Intervalle

Des Weiteren soll für jeden Mess-Client ein Pulsepattern definierbar sein. Dieses Pulsepattern wird dann als Versorgungssignal für die DUTs verwendet.

Automatische Erfassung der Messdaten

Die Messdaten der DUTs sollen zyklisch erfasst werden. Es soll die derzeitige Temperatur im Ofen, der gemessene Wert des Sensors 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 diese in einer SQL Datenbank abgelegt werden. Dafür ist eine Kommunikationsschnittstelle zwischen der Datenbank und den Mess-Clients erforderlich, welcher über den Mess-Server realisiert werden soll.

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 dann grafisch auf dem PC-Client zur Auswertung aufbereitet werden. Auch soll ein Tunnelmodus direkten Zugriff von einem PC-Client auf einem Mess-Client ermöglichen. Dabei sollen die Parameter des Mess-Clients verändert werden können.

Diese Basis-Anforderungen und einige zusätzliche Anforderungen 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 DUT aufnehmen	Hauptanforderung des Systems. Messdaten sollen zu einem DUT gesammelt werden, um den Grad der Degradation bestimmen zu können.
Funktional	Neue Mess-Clients sollen am PC-Client parametrierbar sein.	Parameter sollen auf dem Mess-Client und in einer Datenbank gespeichert werden.
Funktional	Neue bereits parametrierte Mess-Clients sollen automatisch in das System integrierbar sein.	Ein parametrierter Mess-Client kann direkt an den Bus im Ofen 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 auf die Erstellung des Konzeptes eingegangen.

4.1 Übersicht

Das Gesamtsystem setzt sich aus drei Untersystemen zusammen:

- Mess-Client
 - Übernimmt die lokale Ansteuerung der DUTs
 - Nimmt Messdaten auf und stellt sie zur Verfügung
- Mess-Server
 - Verwaltet angeschlossene Mess-Clients
 - Speichert alle Messdaten
 - Bildet das Bindeglied zwischen Mess-Client und dem PC-Client
- PC-Client
 - Parametriert die Mess-Clients
 - Wertet Messdaten aus und stellt sie leserlich da

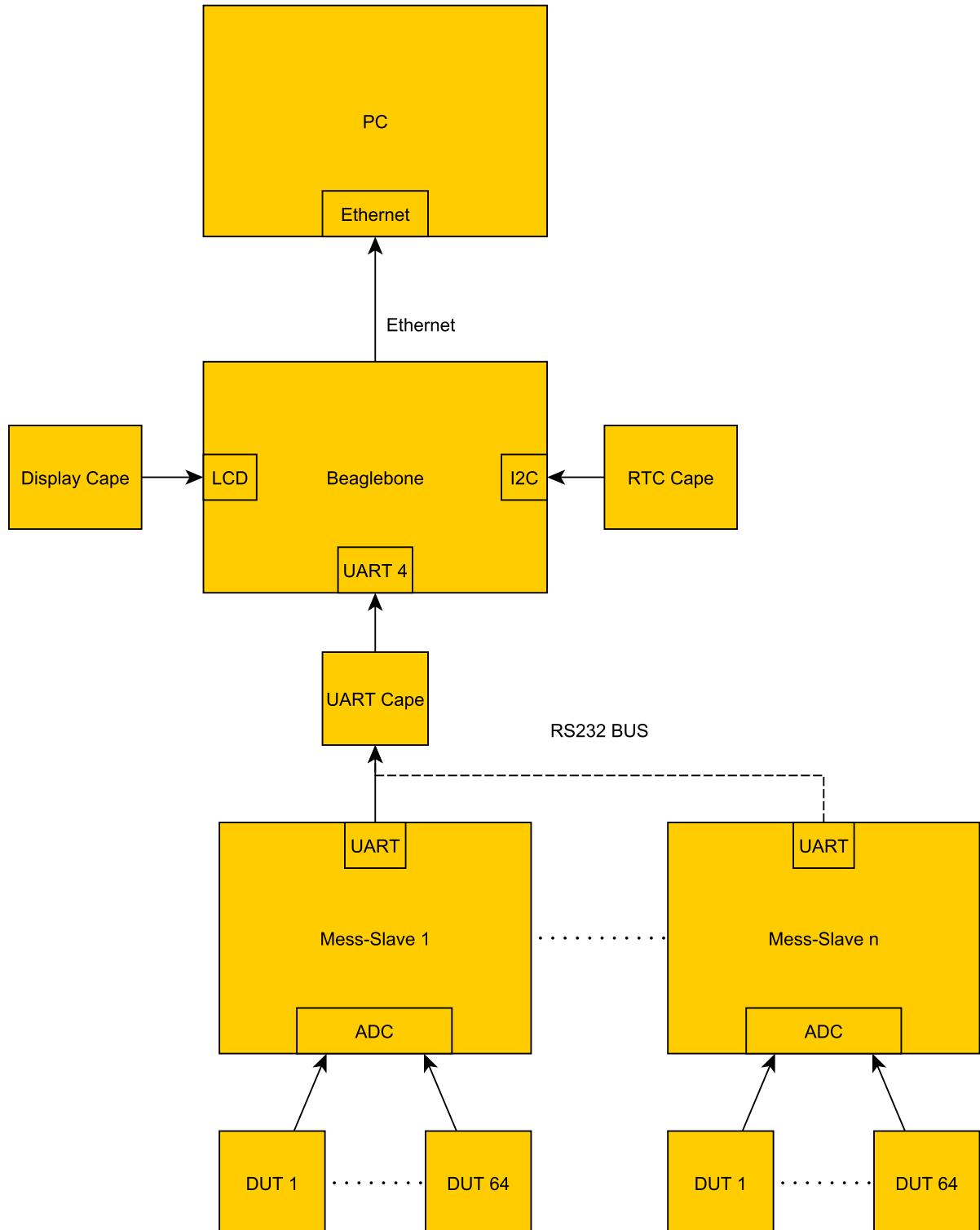


Abb. 4.1.1: Gesamt System

4.2 Hardware

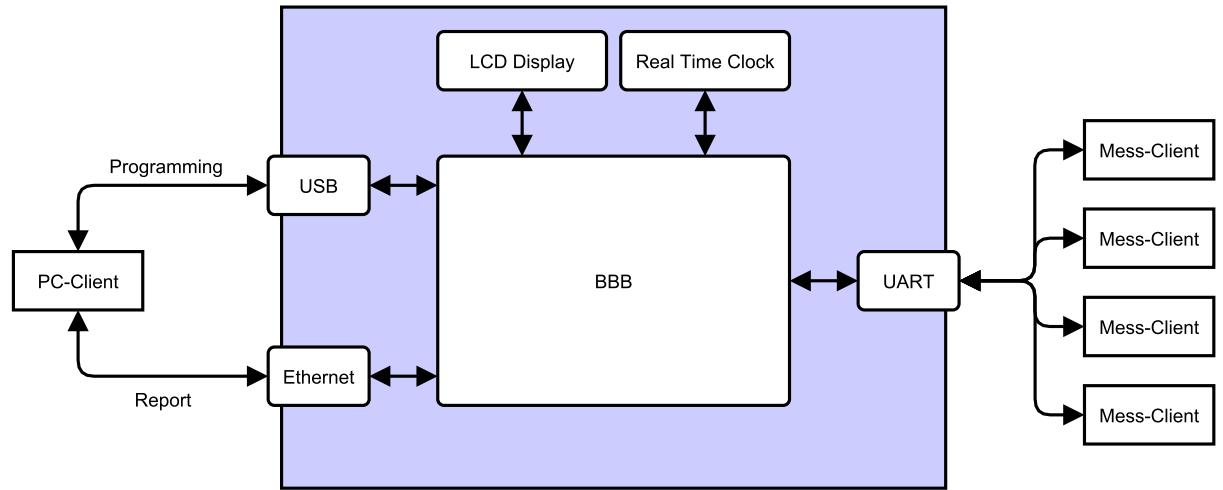


Abb. 4.2.1: Aufbau Mess-Server

Das BeagleBone selbst ist bereits sehr Leistungsstark. Um jedoch weiter Funktionen und Schnittstellen hinzuzufügen, existieren 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 komfortable.

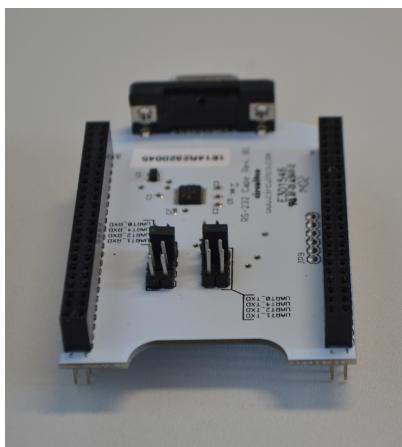


Abb. 4.2.2: RS232 Cape

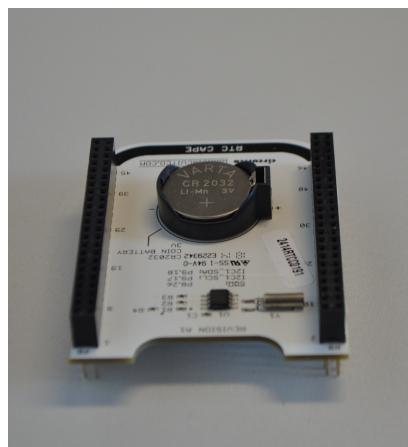


Abb. 4.2.3: RTC Cape



Abb. 4.2.4: LCD Cape

Zur Kommunikation mit den Mess-Clients wird die Universal Asynchronous Receiver Transmitter (UART) Schnittstelle des BeagleBone Black verwendet. Dafür wird ein RS232 Cape eingesetzt (siehe Abbildung 4.2.2). 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 Jumper 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 4.2.3). 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 4.2.4) . Dadurch ist es möglich, das BeagleBone Black trotz den Erweiterungen kompakt zu halten. Denn die Capes sind untereinander stapelbar (siehe Abbildung 4.2.5).



Abb. 4.2.5: 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.

4.3 Software

Die Programmierung der Software erfolgt in C++ unter Verwendung der Klassenbibliothek Qt (siehe Abschnitt 2.2).

Das Softwaredesign teilt sich in einen sequentiellen Teil für die Abfrage und Speicherung der Messwerte, sowie einen Event gesteuerten Teil für die GUI und die externe Kommunikation für

die Fernzugriffe. Die beiden Programmteile können unabhängig von einander agieren und kommunizieren ausschließlich über Signale und Slots (siehe 2.2.1). Durch diese Kapselung ist es möglich die beiden Programmteile durch andere Lösungen auszutauschen, welche lediglich die selben Signale und Slots unterstützen müssen.

4.3.1 Messdatenerfassung

Der Hauptzyklus der Software ruft kontinuierlich die Messdaten von den Mess-Clients 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 der Tabelle 4.3.1 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. 4.3.1: 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-Client verfügbar ist. Dazu wird eine Namensanfrage über die RS232 Schnittstelle verschickt. Bei einer positiven Antwort wird dann eine Messung durchgeführt. Dabei werden alle Analog-Digital-Converters (ADCs) der 64 möglichen DUTs mittels *ADC-Value*-Befehls (siehe Tabelle ??) 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.

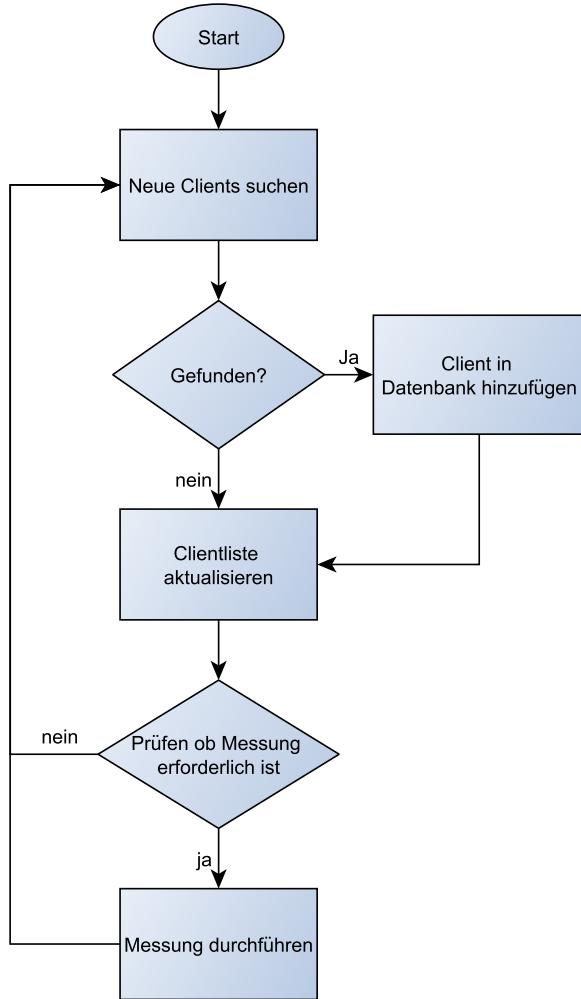


Abb. 4.3.1: Ablaufplan

4.3.2 Benutzeroberfläche

Um die Anforderung der Statusüberwachnung zu erfüllen, verfügt das BeagleBone über eine GUI. Sie bietet einen einfachen Überblick über die aktuellen Vorgänge und soll auf einen Blick den Status des Mess-Servers wiedergeben.

Am oberen Rand der GUI (siehe u.a. Abbildung 4.3.2) wird das aktuelle Datum und die aktuelle Uhrzeit angezeigt, sowie die derzeitige IP Adresse und der aktuelle Name. Am unteren Rand wird die derzeit ausführende Aktion in einer Statusnachricht ausgegeben. Die GUI ist dabei in vier Tabs unterteilt.

Status Tab

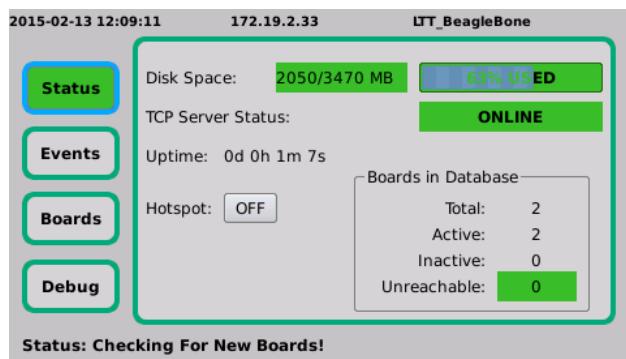


Abb. 4.3.2: Mess-Server GUI: Status Tab

Das Erste ist das Status-Tab (siehe Abbildung 4.3.2). 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.

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. 4.3.3: Mess-Server GUI: Events Tab

Im Events Tab werden wichtige Ereignisse dargestellt.

Boards Tab

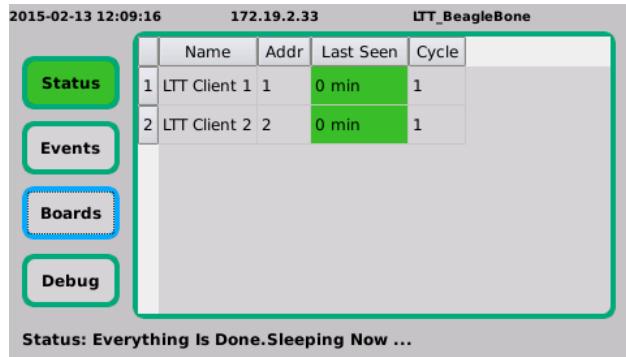


Abb. 4.3.4: Mess-Server GUI: Boards Tab

Das Boards Tab zeigt die derzeit aktiven Mess-Clients in einer Liste an. Als aktiv werden Mess-Clients 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 erfolgreichen aufgenommenen Messzyklen.

Debug Tab



Abb. 4.3.5: Mess-Server GUI: Debug Tab

Der Zweck des Debug Tabs ist es die im Programmcode erzeugten Nachrichten anzuzeigen. Somit soll es möglich sein Fehler einfacher zu erkennen.

4.3.3 RS232 Kommunikation

Die RS232 Schnittstelle wird ausschließlich zur Kommunikation mit den Mess-Clients verwendet. Um die Erfolgschancen der Anfragen zu erhöhen, werden Fehlschläge erkannt und durch den Versuch des erneuten Sendens minimiert. Zwischen jedem Sendeversuch befindet sich eine kurze Verzögerung.

client anmeldung

client abfrage der werte

4.3.3.1 Ethernet Kommunikation

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 im Netzwerk. Dabei antwortet der Mess-Server auf Broadcast-Nachrichten mit seiner IP-Adresse und seinem Namen. Dies ermöglicht die einfache Eingliederung der Mess-Server in ein Netzwerk. Der TCP-Server nimmt RS232- und SQL-Befehle an und leitet diese weiter. Dies ermöglicht den Fernzugriff auf die einzelnen Mess-Clients über ein Netzwerk.

Command	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	RS232 Rahmen	Das BeagleBone sende den empfangenen Rahmen über seine RS232 Schnittstelle
SQLCMD:	TCP	SQL Anfrage	Das BeagleBone führt die SQL Anfrage aus

Tab. 4.3.2: Ethernet Befehlsliste

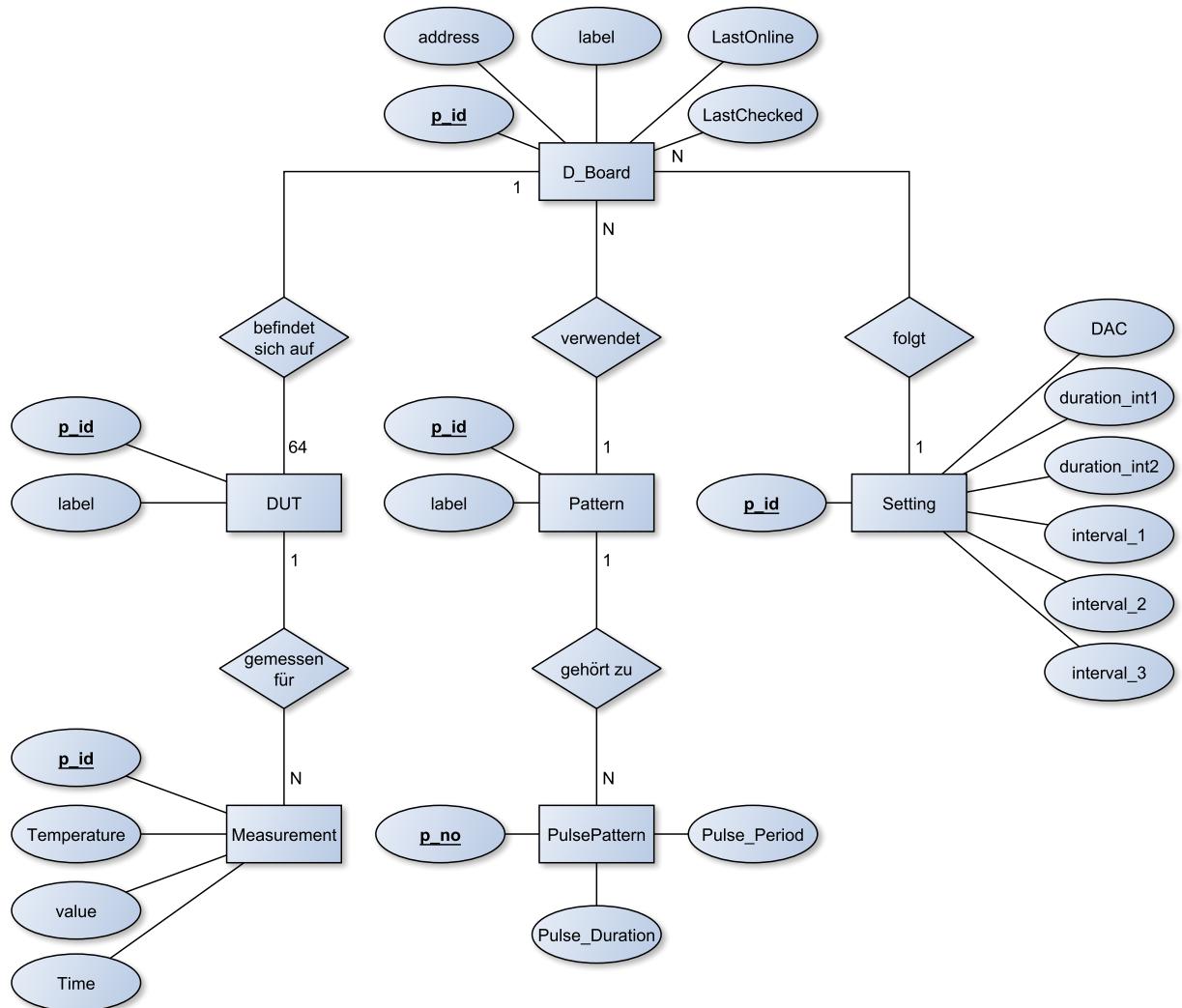
4.3.4 Webzugriff

Um eine Übersicht über die gesammelten Daten auf einem Mess-Server zu erhalten, ist ein lighttpd Webserver installiert.

4.3.5 Datenbank

Auf dem BeagleBone kommt ein MySQL Datenbankserver zum Einsatz.

Aus den Anforderungen ergibt sich folgendes Entity-Relationship-Modell (ERM) (siehe Abbildung 4.3.6).

**Abb. 4.3.6:** Entity Relationship Modell

Die Datenbank muss folgende Daten für die Parameter der Mess-Clients aufnehmen:

Parameter	Beschreibung
DAC	Vorverstärkung
duration_int1	Dauer der Zeit die Werte im 1.Interval aufgenommen werden in Tagen
duration_int2	Dauer der Zeit die Werte im 2.Interval aufgenommen werden in Tagen
interval_1	Abstand zwischen den Messungen im 1. Interval in Minuten
interval_2	Abstand zwischen den Messungen im 2. Interval in Minuten
interval_3	Abstand zwischen den Messungen nach dem 2. Interval in Minuten

Tab. 4.3.3: Tabelle Setting

4.4 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 Baudrate von 2400 verwendet. Da nur geringe Datenmengen in großen Abständen auftreten, vermindert 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 verwendeten Even-Paritätsbit 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.

4.4.1 Aufbau

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

Tab. 4.4.1: Übertragungsrahmen

Ein Rahmen des Protokolls besteht aus 4 Steuerbytes, 1 Checksummenbyte und maximal 30 Datenbytes. 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. 4.4.2: 1. Byte: Adresse & Read/Write

Das erste Byte des Übertragungsrahmens setzt sich aus 7 Adressbits und einem Lese-/Schreibbit zusammen. Die ersten 7 Bits (Addr0 - Addr6) bilden die Adresse des anzusteuernden Empfänger. Daraus ergibt sich ein Adressraum von möglichen 128 Adressen, wobei Adresse 0 für neue Mess-Clients zur einmaligen Anmeldung im System reserviert ist (siehe Abschnitt 4.3.3).

Das höchste Bit ist das Lese-/Schreibbit. Der Mess-Client unterscheidet mithilfe dieses Bits, ob ein Befehl als Lese- oder Schreibzugriff interpretiert werden soll.

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

Tab. 4.4.3: Read/Write

Ein Lesezugriff bedeutet immer, dass keine Daten übertragen werden.

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. 4.4.4: 2. Byte: Rahmenlänge

Das zweite Byte gibt die Länge des gesamten Rahmens inklusive Steuerbytes, Nutzdaten und Checksumme an.

Die minimale Rahmenlänge beträgt 5 Byte. 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 des 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. 4.4.5: 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 zulässig.

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. 4.4.6: 4. Byte: Unterbefehl

Das vierte Byte ist der Unterbefehl. Damit ist es möglich einen Befehl genauer zu definieren. So kann beispielsweise der Befehl zum Auslesen eines ADC Wertes durch den Unterbefehl genau auf einen von 64 ADCs präzisiert werden.

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. 4.4.7: 5. Byte und folgend: Nutzdaten

Das fünfte Byte und die darauf folgenden, tragen die Nutzdaten des Rahmens. Die Größe der Nutzdaten ist variable und lässt sich auf der Rahmenlänge ableiten. Bei zwei Byte Daten wird immer 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. 4.4.8: 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 wird.

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.

4.4.2 Befehle und Unterbefehle

Für die Funktion des Systems sind verschiedene Befehle notwendig. Sie dienen zu Kommunikation zwischen den Akteuren. Eine Übersicht über die vorhandenen Befehle findet sich in Tabelle 4.4.9.

Command	Code	Subcommand	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
LTU Name	0x09	-	1..30
Rs232-Address	0x0A	-	1
Error	0x0B	MUX-Channel (0..63)	2
Measurement Intervall	0x0C	Intervallnummer (0..2)	4

Tab. 4.4.9: RS232 Befehlsliste

5 Testen und Validieren

TEST TEST 123

6 Fazit und Ausblick

Literaturverzeichnis

- [Ben05] BENDER, K.: *Embedded Systems: Qualitätsorientierte Entwicklung.* Springer, 2005 <https://books.google.de/books?id=JoMfBAAQBAJ>. – ISBN 9783540273707
- [SGD09] SCHRÖDER, J. ; GOCKEL, T. ; DILLMANN, R.: *Embedded Linux::* Springer, 2009 (X. systems. press Series). <https://books.google.de/books?id=vpgrxY4hcHIC>. – ISBN 9783540786191
- [SSH10] SAAKE, G. ; SATTLER, K.U. ; HEUER, A.: *Datenbanken: Konzepte und Sprachen.* mitp, 2010 (Biber-Buch). – ISBN 9783826690570
- [YMBYG08] YAGHMOUR, K. ; MASTERS, J. ; BEN-YOSSEF, G. ; GERUM, P.: *Building Embedded Linux Systems.* O'Reilly Media, 2008 <https://books.google.de/books?id=I-hVqkX6A9cc>. – 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

