Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Fachbereich: Ingenieurwissenschaften – Energie und Information
Studiengang: Computer Engineering

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 I

Tabellenverzeichnis II

Abkürzungsverzeichnis III

1 Einleitung 1
- 1.1 Motivation . . . 1
- 1.2 Aufbau der Arbeit . . . 1

2 Grundlagen 2
- 2.1 Degradation . . . 2
- 2.2 Qt . . . 2
- 2.3 Datenbank . . . 4

3 Anforderungsanalyse 5
- 3.1 Szenario . . . 5
- 3.2 Anforderungen . . . 6

4 Konzept 9
- 4.1 Teststand . . . 9
  - 4.1.1 Mess-Client . . . 11
  - 4.1.2 Mess-Server . . . 12
    - 4.1.2.1 Aufbau . . . 13
  - 4.1.3 PC-Client . . . 14
  - 4.1.4 RS232 Protokoll . . . 15
    - 4.1.4.1 Aufbau . . . 15
    - 4.1.4.2 Befehle und Unterbefehle . . . 18
- 4.2 Datenbank . . . 19

5 Testen und Validieren 21

Literaturverzeichnis i

Abbildungsverzeichnis

2.2.1 QtCreator Version 2.0.1 3

3.1.1 Szenario 5

4.1.1 Gesamt System 10
4.1.2 BeagleBone Black 12
4.1.3 Gesamt System 13
4.2.1 Entity Relationship Modell 19

Tabellenverzeichnis

3.2.1 Intervalle 6
3.2.2 Anforderungen 8

4.1.1 Übertragungsrahmen 15
4.1.2 1. Byte: Adresse & Read/Write 15
4.1.3 Read/Write 16
4.1.4 2. Byte: Rahmenlänge 16
4.1.5 3. Byte: Befehl 16
4.1.6 4. Byte: Unterbefehl 16
4.1.7 5. Byte und folgend: Nutzdaten 17
4.1.8 Letztes Byte: Checksumme 17
4.1.9 Befehlsliste 18
4.2.1 Tabelle Setting 20

Abkürzungsverzeichnis

SQL Structured Query Language 4
DB Datenbank 4
DBMS Datenbankmanagementsystem 4
DBS Datenbanksystem 4
ERM Entity-Relationship-Modell 19
DUT Device Under Test 6
GUI Graphical User Interface 2
IDE Integrated Developement Environment 2
SDK Software Developement Kit 2
LED Light-Emitting Diode 5
ADC Analog-Digital-Converter 17
UART Universal Asynchronous Receiver Transmitter 12

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 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

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.

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.

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.

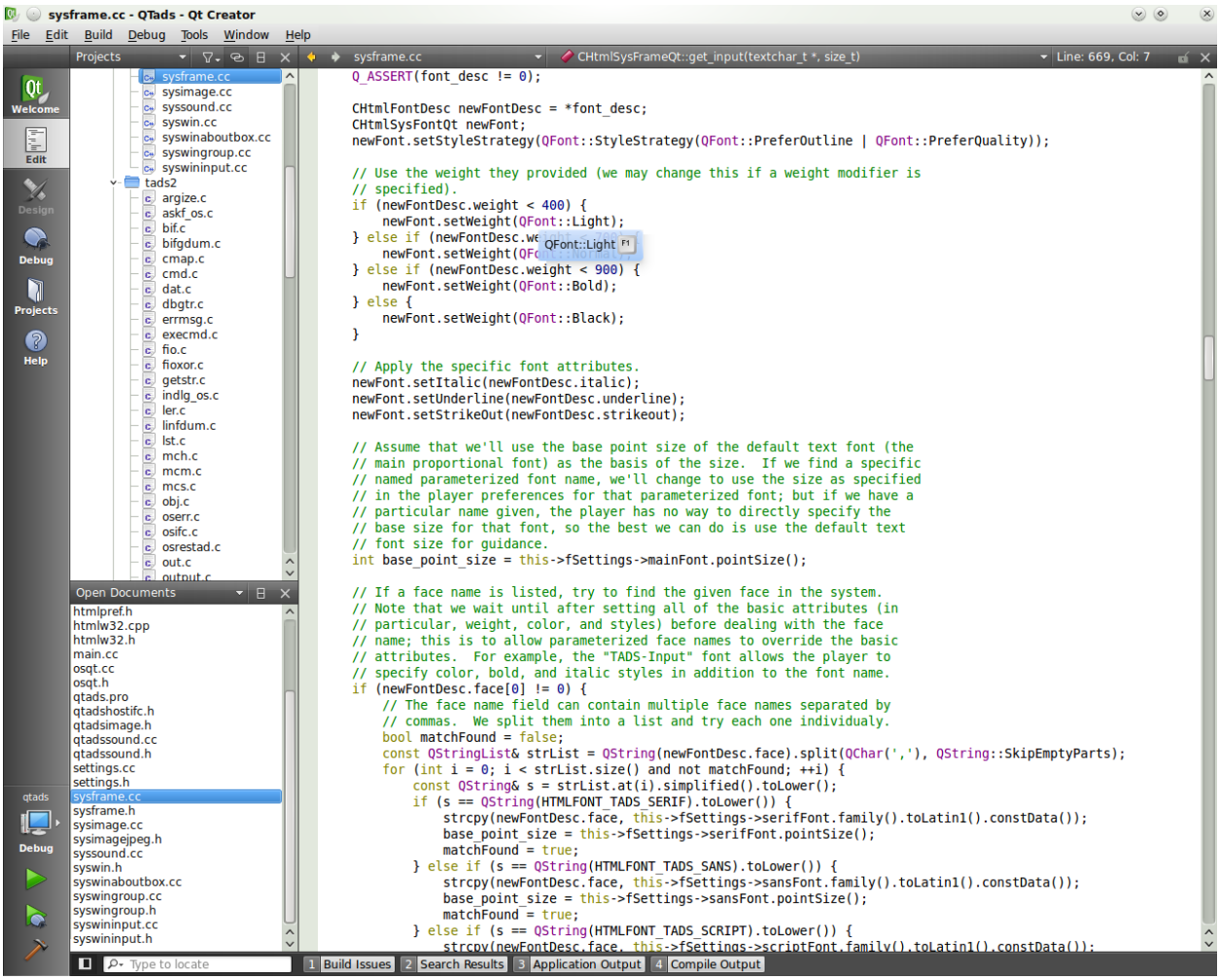

Abb. 2.2.1: QtCreator Version 2.0.1

2.3 Datenbank

Für das Speichern der Messdaten und Betriebsparameter wird eine Structured Query Language (SQL) Datenbank (DB) verwendet. Bei der Wahl des Datenbankmanagementsystem (DBMS), standen mehrere Optionen zur Auswahl und die Entscheidung wurde zwischen SQLite und MySQL gefällt. Beide System haben ihre Vor- und Nachteile.

SQLite ist ein SQL DBMS, welches ohne einen Server auskommt und operiert stattdessen in einer einzigen Datei. Es wird vor allem im Embedded Bereich eingesetzt, da kaum Konfigurationen oder Verwaltung notwendig ist. Deshalb eignet es sich ausgezeichnet für sich schnell weiterentwickelnde Applikationen.
Aufgrund dieser Eigenschaften existieren allerdings auch Nachteile. So unterstützt SQLite nur eingeschränkt mehrere Nutzer gleichzeitig. Da das gesamte Datenbanksystem (DBS) in einer einzigen Datei zusammengefasst ist, können mehrere zur selben Zeit durchgeführte Schreibzugriffe nicht unterstützt werden. Denn die einzige Sicherstellung der Datenintegrität erfolgt durch das Betriebssystem.
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 Rechte. Außerdem ist das System gut hinsichtlich Performance und Größe zu skalieren. Zusätzlich bietet MySQL viele Möglichkeiten für Performanceanpassungen, wie z.B. Query-Caching.
Jedoch gibt es auch hier Nachteile. So ist die Konfiguration wesentlich schwerer und komplexer. Durch die Notwendigkeit eines Servers benötigt MySQL mehr Ressourcen auf dem Host-System.

Die Wahl fällt auf das relationale DBMS MySQL. Auch wenn SQLite einige Vorteile vor allem im Embedded Bereich besitzt, ist die fehlende Unterstützen von mehreren Nutzern gleichzeitig ein Ausschlusskriterium.

[SSH10]

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

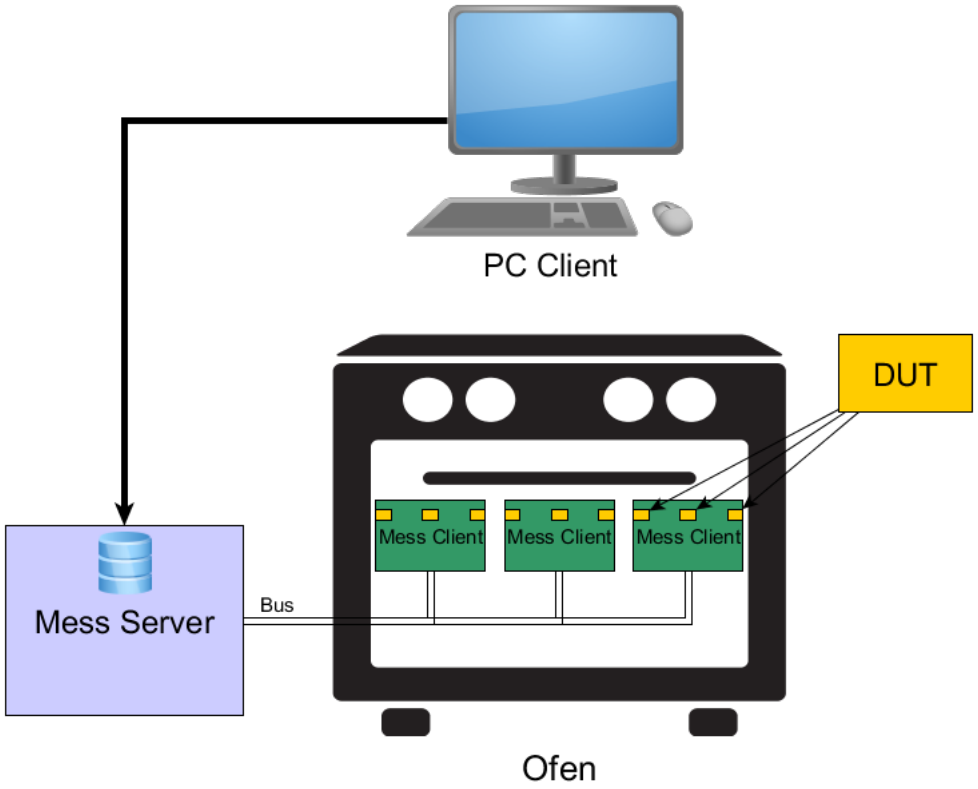

Abb. 3.1.1: Szenario

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 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. 3.2.1: 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 3.2.2 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. 3.2.2: Anforderungen

4 Konzept

Im folgenden Kapitel wird auf die Erstellung des Konzeptes eingegangen.

4.1 Teststand

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

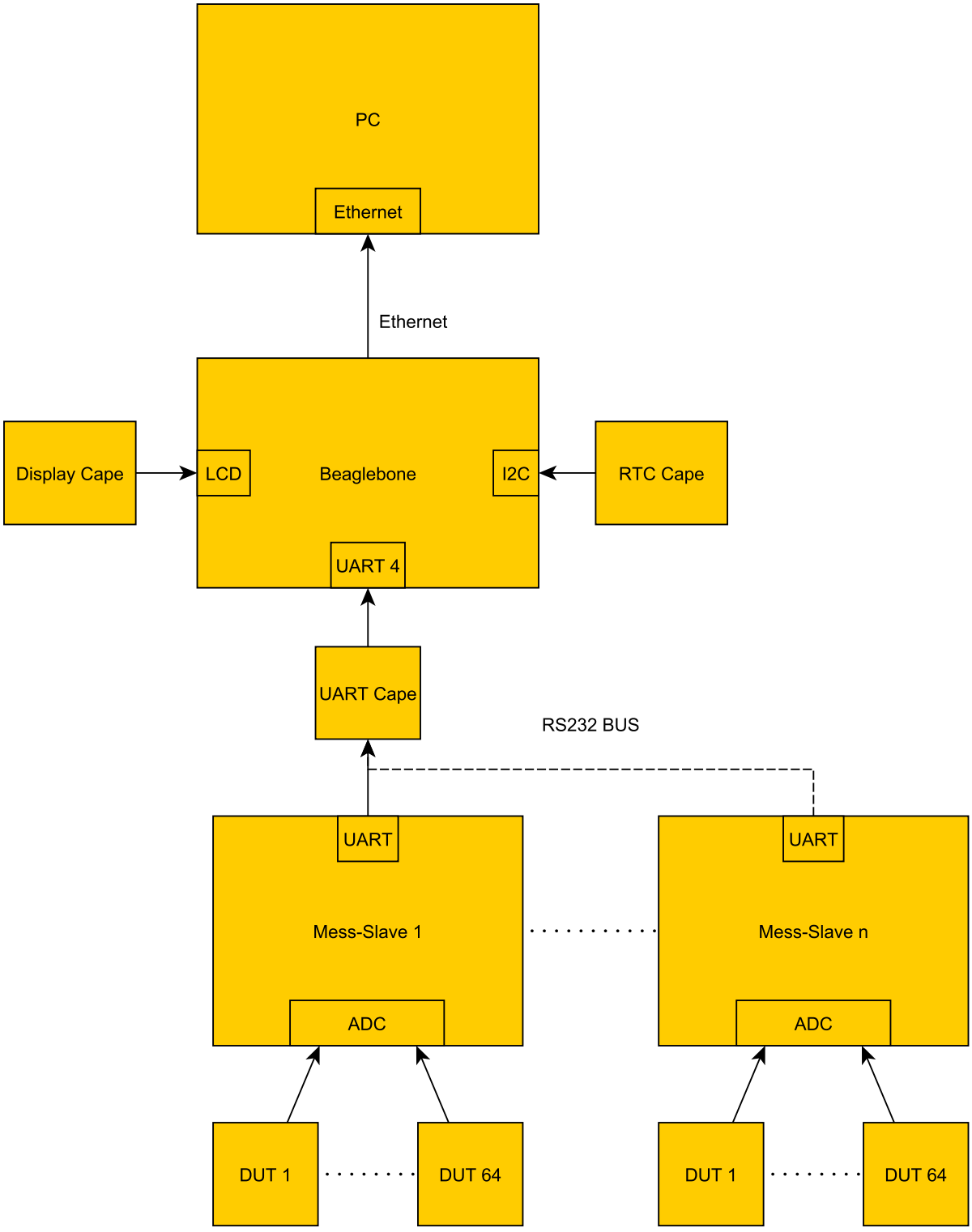

Abb. 4.1.1: Gesamt System

4.1.1 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 (siehe Abbildung **??**.
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 4.1.4).

4.1.2 Mess-Server

Als Mess-Server wird ein BeagleBone Black von Texas Instruments eingesetzt. Dabei handelt es sich um einen kostengünstigen Einplatinencomputer der mit einem AM335x 1GHz ARM® Cortex-A8 Prozessor arbeitet. Des Weiteren verfügt er über 512MB DDR3 RAM und 4GB 8-bit eMMC internen Flash Speicher.

Abb. 4.1.2: BeagleBone Black

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 Universal Asynchronous Receiver Transmitter (UART) Interface beispielsweise wie eine normale Datei beschrieben und gelesen werden.

4.1.2.1 Aufbau

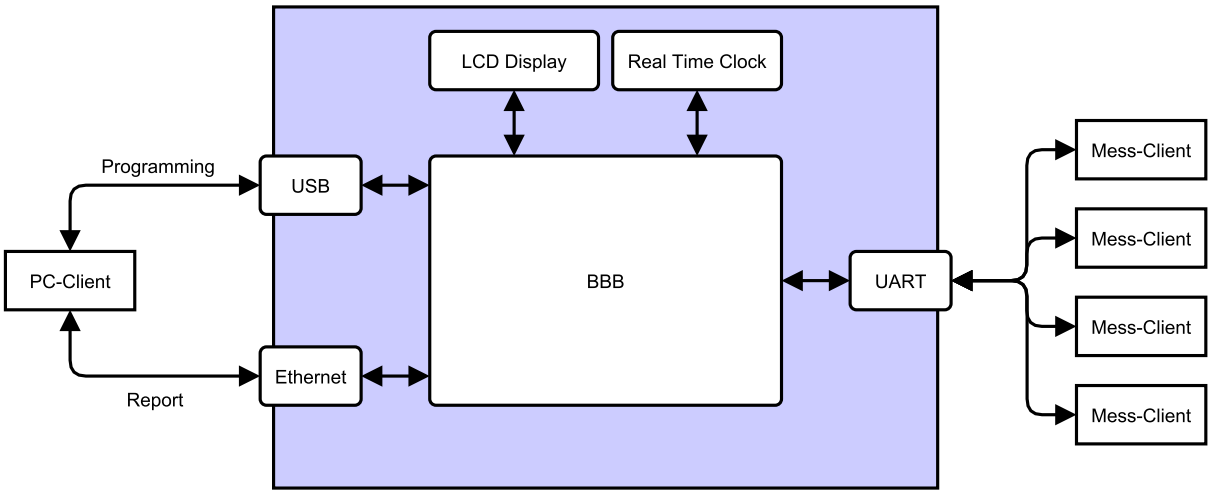

Abb. 4.1.3: Aufbau BeagleBone Black

Das BeagleBone Black bildet das Herzstück des Systems.

4.1.3 PC-Client

asdasad

4.1.4 RS232 Protokoll

Das Protokoll für die Kommunikation über die RS232 Schnittstelle ist nötig, um die Validität, Vollständigkeit und Zuverlässigkeit der Übertragungen sicherzustellen.

Folgende Kriterien sollen dabei erfüllt werden:

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

4.1.4.1 Aufbau

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

Tab. 4.1.1: 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. 4.1.2: 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 Mess-Clients. Daraus ergibt sich ein Adressraum von möglichen 128 Adressen, wobei Adresse 0 für neue Mess-Clients zur einmaligen Anmeldung im System reserviert ist.
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. 4.1.3: Read/Write

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. 4.1.4: 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. 4.1.5: 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. 4.1.6: 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 Analog-Digital-Converter (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. 4.1.7: 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 dich 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. 4.1.8: 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.

4.1.4.2 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 4.1.9.

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

Tab. 4.1.9: Befehlsliste

4.2 Datenbank

Aus den Anforderungen ergibt sich folgendes Entity-Relationship-Modell (ERM) (siehe Abbildung 4.2.1).

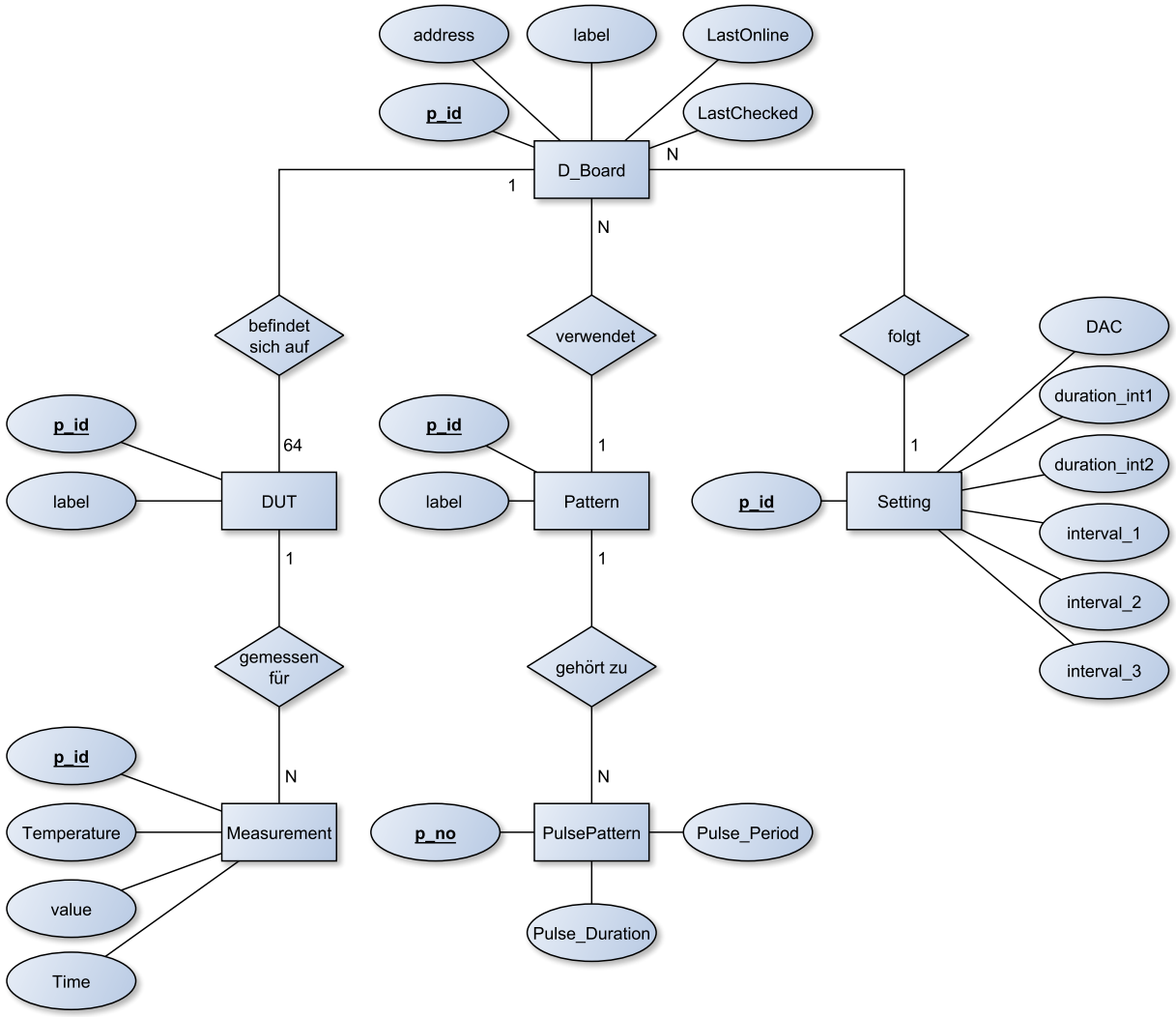

Abb. 4.2.1: Entity Relationship Modell

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. 4.2.1: Tabelle Setting

5 Testen und Validieren

TEST TEST 123

Literaturverzeichnis

[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

[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