technische universität
dortmund

Bachelorarbeit

Eine Experimentierumgebung zur Durchführung und Auswertung von Energiemessungen mit MIMOSA

David Tondorf
November 2014

Gutachter:
Prof. Dr.-Ing. Olaf Spinczyk
Dipl.-Inf. Markus Buschhoff

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl Informatik 12
http://ls12-www.cs.tu-dortmund.de

ERKLÄRUNG

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 16. November 2014

David Tondorf

Inhaltsverzeichnis

Abstract

Diese Bachelorarbeit beschreibt die Entwicklung einer Experimentierumgebung, mit deren Hilfe Messungen mit MIMOSA durchgeführt werden können, wobei gleichzeitig eine Ansteuerung des zu vermessenden Gerätes möglich ist.
Den Ausgang der Entwicklung bildete dabei eine Analyse des aktuellen Messvorgangs. Darauf aufbauend wurde ein optimierter Messvorgang entworfen. Um diesen umzusetzen wurde anschließend eine Anforderungsanalyse durchgeführt, die ergab, dass ein auf verschiedenen Komponenten basierendes modulares System die beste Lösung für die gewünschte Funktionalität darstellt.
Zur Umsetzung der Anforderungen wurde ein Konzept erstellt und implementiert.
Abschließend wurden verschiedene Experimente mit Hilfe der neuen Software durchgeführt. So konnte gezeigt werden, dass die Experimentierumgebung eine erhebliche Erleichterung bei der Durchführung von Experimenten darstellt und die gelieferten Ergebnisse korrekt sind.

1. Einleitung

Im Forschungsbereich der eingebetteten Systeme ist es oftmals nötig, den Energieverbrauch einzelner Komponenten oder ganzer Systeme zu bestimmen. Um ausreichend genaue Messungen durchführen zu können sind hier in der Regel spezielle Messgeräte nötig, da diese sogenannten „Low Power“ Systeme besondere Anforderungen an die Messtechnik stellen. Eines dieser Messgeräte ist MIMOSA[1]. Bei computergestützter Messtechnik wird außerdem auch noch die passende Software benötigt um die Messungen durchzuführen und auszuwerten.
Dieses Kapitel beschreibt einleitend das Messgerät MIMOSA, um diese Arbeit in einen Kontext zu setzen und die Grundlage zum Verständnis der restlichen Arbeit zu schaffen. Anschließend wird basierend auf der aktuellen Situation die Motivation dieser Arbeit erläutert. Den Kapitelabschluss bildet eine kurze Übersicht über den Aufbau der Arbeit.

1.1. MIMOSA

Bei MIMOSA handelt es sich um ein von Markus Buschhoff und Christian Günter entwickeltes Messgerät zur hochpräzisen Messung des Energieverbrauchs von eingebetteten Systemen (siehe [8]). Es wurde mit dem Ziel entwickelt, zuverlässige Energiemodelle von Systemen erstellen zu können, bei denen der Stromfluss im Bereich von unter einem Milliampere liegt.

1.1.1. Grundlagen und Funktion

Um zu erklären, wie sich MIMOSA von anderen Messgeräten unterscheidet, müssen zunächst einige Grundlagen der Elektrotechnik erläutert werden.

$$U = R * I \tag{1}$$

$$Q = \int_{t_0}^{t_1} I(t)dt \tag{2}$$

Das Ohmsche Gesetz (1) beschreibt den Zusammenhang zwischen dem durch einen Widerstand (R) fließenden Strom (I) und der daran abfallenden Spannung (U).
Strommessgeräte machen sich diesen Zusammenhang zunutze. Da es nicht möglich ist,

[1] Messgerät zur integrativen Messung ohne Spannungsabfall

den Strom direkt zu messen, wird statt dessen der Spannungsabfall über einen bekannten, hochpräzisen ohmschen Widerstand, genannt *Shunt*, gemessen. Daraus lässt sich dann über (1) der durch den Widerstand fließende Strom berechnen. Dieser Ansatz bringt allerdings auch Probleme mit sich. Wird der Shunt Widerstand zu klein gewählt, so kommt es zu keinem ausreichend großen Spannungsabfall und eine genaue Messung kann nicht durchgeführt werden. Wird der Shunt hingegen zu groß gewählt, so ist auch der Spannungsabfall sehr groß, was dazu führen kann, dass die Betriebsspannung nicht mehr ausreicht um das Device Under Test (DUT) fehlerfrei zu betreiben. MIMOSA umgeht dieses Problem, indem eine Schaltung zur Kompensation des Spannungsabfalls verwendet wird (Abbildung 1.1[2]).

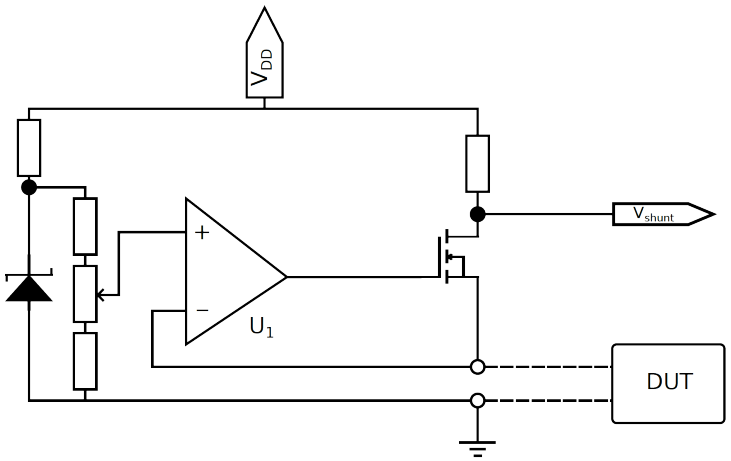


Abbildung 1.1.: Vereinfachte Darstellung der Schaltung zur Kompensation des Spannungsabfalls

Eine präzise Referenzspannungsquelle, in der Abbildung als Z-Diode dargestellt, liefert die Eingangsspannung für den Operationsverstärker (U1). Dieser ist als Spannungsfolger geschaltet. Er kopiert die Eingangsspannung (+) an den Ausgang. Der Ausgang wiederum ist mit dem Minus-Eingang (-) des Operationsverstärkers verbunden und erzeugt so eine Gegenkopplung. Fällt die Spannung V_{shunt} ab, so bedeutet dies auch ein Abfallen der Spannung am Minus-Eingang. Da die Spannung des Plus-Eingangs konstant ist, entsteht also eine Spannungsdifferenz zwischen beiden Eingängen des Operationsverstärkers. Dieser erhöht daraufhin die Ausgangsspannung, bis an beiden Eingängen wieder die gleiche Spannung anliegt.
Durch diese Kompensation des Spannungsabfalls ist es möglich, einen großen Shunt Widerstand zu verwenden, um eine genaue Messung des Spannungsabfalls zu ermöglichen, sowie gleichzeitig die Versorgung des DUT mit einer konstanten Betriebsspannung sicherzustellen.

[2] Abbildung 1.1 entnommen aus [8]

Eine weitere Besonderheit von MIMOSA stellt die Art der Messwerterfassung dar. Hierzu werden analoge Integratoren verwendet. Die über den oben beschriebenen Shunt abgefallene Spannung erzeugt einen Stromfluss, der einen Kondensator auflädt. MIMOSA arbeitet mit einer Samplingrate von 100 kHz, was bedeutet, dass der Strom jeweils über eine Periode von einer Hunderttausendstel Sekunde integriert wird. Nach (2) ist dies die Ladung Q. Im nächsten Schritt wird der analoge Ladungswert von einem A/D-Wandler in einen digitalen 16-bit Wert gewandelt. Zuletzt muss der Kondensator wieder entladen werden. Um sicherzustellen, dass es keine Unterbrechungen der Messung gibt, verwendet MIMOSA drei Integratoren. Diese führen jeweils nacheinander die eben beschriebenen drei Schritte aus: Während der erste Integrator den Strom der aktuellen Periode t_0 integriert, wird der Wert des zweiten Integrators, also das Ergebnis der Integration aus der vorherigen Periode t_{-1}, vom A/D-Wandler gesampelt. Der dritte Integrator, der in der Periode t_{-2} für die Integration verantwortlich war, wird derweil zurückgesetzt. Diese Art der analogen Integration hat außerdem den Vorteil, dass auch sehr kurze Anstiege im Energieverbrauch (*Peaks*), deren Dauer unterhalb der Dauer einer Samplingperiode von MIMOSA liegt, erfasst werden können.

1.1.2. Format des Datenstroms

MIMOSA verfügt über eine USB-Schnittstelle, über welche die Messwerte an einen PC übertragen werden können. Dabei werden neben dem 16-bit Wert des A/D-Wandlers auch noch weitere Informationen übertragen. Hierzu werden 3 Byte (24 bit) breite Worte verwendet, die folgendes Format haben (Bit 0 ist das Least Significant Bit (LSB)):

Bit(s)	Bezeichnung
0 - 1	I0 - I1
2	RSV
3	BUZ
4 - 19	D0 - D15
20 - 23	Prefix

Tabelle 1.1.: Format eines MIMOSA-Wortes

Die Bits I0 und I1 geben hierbei die Nummer des Integrierers an, der den aktuellen Messwert geliefert hat. Dadurch ist es zum Beispiel möglich, die von den einzelnen Integrierern gelieferten Werte zu vergleichen, um Abweichungen zu erkennen und zu beseitigen (siehe Abschnitt 3.3.1.4). Bit 2 (RSV) wird nicht verwendet und ist immer 0. Bit 3 (BUZ) zeigt den Wert des Buzzer-Eingangs von MIMOSA zum Zeitpunkt der Messung. Der Buzzer-Eingang stellt einen Rückkanal vom DUT zu MIMOSA dar, der z.B. verwendet werden kann, um Markierungen innerhalb des Datenstroms zu setzen. So kann beispielsweise durch Setzen dieses Bits auf 1 angezeigt werden, dass das DUT gerade eine Operation ausführt,

für die der Energieverbrauch bestimmt werden soll. D0 - D15 enthalten den eigentlichen 16-bit Messwert. Die vier höchstwertigsten Bits dienen als Prefix zur Unterscheidung, ob es sich bei dem Wort um ein Daten- oder ein Syncronisationswort handelt. Datenworte sind Worte nach dem eben beschriebenen Muster. Sie lassen sich an dem Prefix „0000“ erkennen. Syncronisationsworte werden bei einem Reset von MIMOSA gesendet, um den Anfang des Datenstroms anzuzeigen. Des Weiteren wird ein Syncronisationswort übertragen, wenn es zu einem Datenverlust durch einen Pufferüberlauf bei MIMOSA gekommen ist.

1.1.3. Umrechnen des A/D-Werts

Der von MIMOSA gelieferte Datenwert, die Bits D0 - D15, ist die in einen digitalen 16-bit Wert gewandelte Ladung des jeweiligen Kondensators zum Zeitpunkt der Abtastung. Da es sich um einen 16-bit Wert handelt, ist dies ein Wert zwischen 0 und 65535. Dieser Wert muss anschließend noch mit Hilfe der folgenden Formeln in eine reale physikalische Größe umgerechnet werden:

$$I_{MessberMax} = \frac{U_{eCoaxMax}}{R_{Shunt}} \tag{3}$$

$$I_{Schritt} = \frac{I_{MessberMax}}{65535 - Offset} \tag{4}$$

$$I_{DUT} = AD * I_{Schritt} \tag{5}$$

$$P_{DUT} = AD * I_{Schritt} * U_{DUT} \tag{6}$$

$$Q_{DUT} = AD * I_{Schritt} * t_{sample} \tag{7}$$

$$E_{DUT} = AD * I_{Schritt} * U_{DUT} * t_{sample} \tag{8}$$

(3) berechnet den maximalen Messbereich von MIMOSA. Dieser ist abhängig vom verwendeten Shunt und der Spannung $U_{eCoaxMax}$, die nur von den Bauteilen der Platine abhängig ist und daher für diese Arbeit als konstant angesehen werden kann. Die genaue Berechnung ist in Anhang A.2 zu finden.
(4) gibt den Strom an, den ein A/D Wert von 1 repräsentiert. Dieser Wert ist nötig, um den Messwert mit Hilfe der Formeln (5) bis (8) in physikalische Größen umzurechnen. t_{sample} bezeichnet hierbei die Dauer einer Samplingperiode, also eine Hunderttausendstel Sekunde. Die Formeln basieren auf [8] und Anhang A.2.

1.2. Motivation

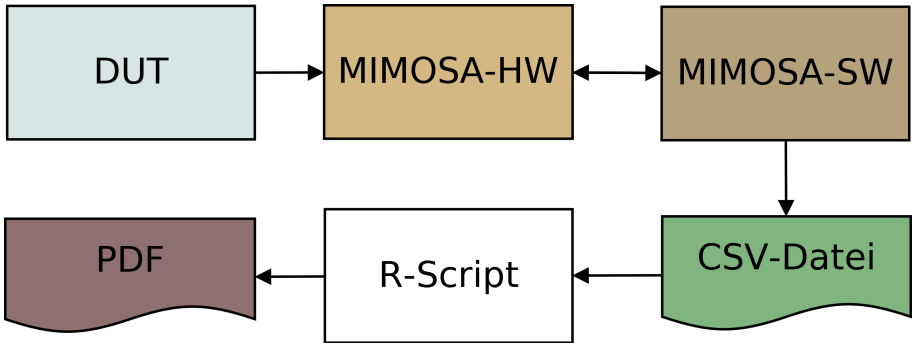


Abbildung 1.2.: Messvorgang aktuell

Möchte man aktuell eine Messung mit MIMOSA durchführen und auswerten, so sieht der Ablauf aus wie in Abbildung 1.2 dargestellt. Die MIMOSA Hardware misst den aktuellen Energieverbrauch des DUT und sendet die Daten über die USB-Schnittstelle an den PC, wo sie von der MIMOSA Software verarbeitet werden. Diese besteht momentan aus einem in C geschriebenen Programm, das den Datenstrom zerlegt und die relevanten Daten in eine CSV-Datei (Comma-seperated Values) speichert. Dabei wird für jeden Messwert eine Zeile geschrieben, die aus einem fortlaufenden Index, der Nummer des Integrierers und dem Messwert besteht. Diese Daten können nun mit einem anderen Programm oder Script (z.B. einem R-Script[3]) weiterverarbeitet werden um z.B. Grafiken zu erstellen.

Der aktuelle Ablauf hat jedoch einige Nachteile. So gibt es beispielsweise keine Live-Darstellung, d.h. der Benutzer kann nicht während des Messvorgangs überprüfen, ob die Messung richtig verläuft. Dies ist erst nach Abschluss der Messung und Verarbeitung der Daten möglich. Zur Verarbeitung der Daten sind wie vorhin beschrieben weitere Software und möglicherweise Programmierkenntnisse nötig, was einen weiteren Nachteil darstellt. Des Weiteren macht das verwendete CSV-Dateiformat MIMOSA ungeeignet für Langzeitmessungen, denn auf Grund der Abtastrate von 100 kHz fallen hier große Datenmenge an (ca. 2,3 MB pro Sekunde). Auch wenn ausreichend Festplattenspeicherplatz heutzutage in der Regel kein Problem mehr darstellt, so verlangsamen große Dateien mit vielen Einträgen doch die Weiterverarbeitung z.B. in R. Allgemein kann der in Abbildung 1.2 dargestellte Prozess daher als langsam und unpraktisch beschrieben werden.
Ein großes Problem stellt auch die fehlende Reproduzierbarkeit von Messungen dar. Der Benutzer muss bisher sowohl die Messung, als auch die zu testende Hardware manuell bedienen und starten, was eine Wiederholung der gleichen Messung zu einem späteren

[3] R ist eine freie Programmiersprache zur statistischen Verarbeitung von Daten und Erstellung von Grafiken, http://www.r-project.org/

Zeitpunkt erschwert. Dies ist der Fall, weil es keine Möglichkeit gibt, aus der MIMOSA Software Einfluss auf das DUT zu nehmen.

1.3. Ziel der Arbeit

Der letzte Abschnitt hat einige Probleme bei der Durchführung von Messungen mit MIMOSA aufgezeigt. Dieser Abschnitt wird nun basierend darauf das Ziel dieser Arbeit erläutern.

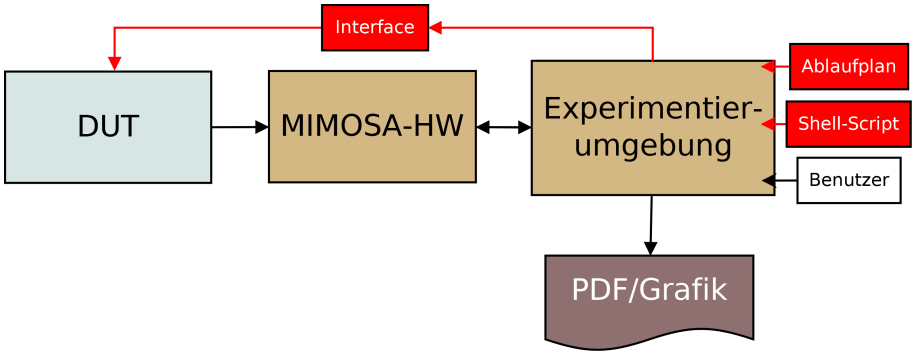


Abbildung 1.3.: Messvorgang gewünscht

Durch eine einfache Überarbeitung des vorhandenen Programms wäre es möglich einige der angesprochenen Probleme zu lösen, indem beispielsweise eine grafische Benutzeroberfläche (GUI) implementiert wird. Ziel dieser Arbeit ist es allerdings nicht, die bestehende Lösung nur zu vereinfachen, sondern der komplette Messablauf soll erweitert und verbessert werden. Insbesondere soll es möglich sein, mit dem DUT zu kommunizieren. Dadurch werden einfache Messungen zu wiederholbaren Experimenten. Abbildung 1.3 zeigt den gewünschten, neuen Ablauf.
Wie bisher misst MIMOSA den Energieverbrauch des DUT und liefert die Daten über die USB-Schnittstelle an den PC. Dort gibt es jetzt allerdings als zentrale Komponente die Experimentierumgebung als Schnittstelle zwischen dem Benutzer und der Hardware. Deutlich zu erkennen ist der nun vorhandene Rückkanal (rot), der es erlaubt direkt Einfluss auf das DUT zu nehmen. Die Experimentierumgebung kann hierbei vom Benutzer direkt bedient werden. Um jedoch die Reproduzierbarkeit von Experimenten zu gewährleisten, ist es außerdem Ziel der Arbeit, ein System zur Automatisation zu schaffen. In der Grafik wird dies durch die Komponenten „Ablaufplan“ und „Shell-Script“ dargestellt.

1.4. Aufbau der Arbeit

Dieses Kapitel stellt die Einleitung der Arbeit dar. Im nächsten Kapitel werden die Anforderungen an die Experimentierumgebung untersucht und es wird ein Konzept zur Umsetzung aufgestellt, dessen Implementierung in Kapitel 3 beschrieben wird. Kapitel 4 beschreibt anschließend die Validierung der Experimentierumgebung. Zu diesem Zweck wurden verschiedenen Experimente durchgeführt, deren Planung, Durchführung und Auswertung dieses vorletzte Kapitel beschreibt. Den Abschluss der Arbeit bildet eine Zusammenfassung der gesamten Arbeit in Kapitel 5.

2. Anforderungsanalyse und Konzept

Dieses Kapitel beschäftigt sich mit den Anforderungen, die an die zu entwickelnde Experimentierumgebung gestellt werden. Basierend auf den Anforderungen in Abschnitt 2.1 wird in Abschnitt 2.2 ein Konzept erstellt, welches die verschiedenen Komponenten der Software erläutert.

2.1. Anforderungsanalyse

Dieser Abschnitt beschreibt die Anforderungen an die Experimentierumgebung, die erforderlich sind um das in 1.3 beschriebene Ziel zu erreichen.

2.1.1. GUI

Um die Bedienung möglichst einfach zu gestalten muss es eine grafische Benutzeroberfläche (GUI) geben. Diese stellt die primäre Interaktionsschnittstelle zwischen dem Benutzer und MIMOSA dar. Sie muss daher alle Funktionen bereitstellen um Messungen und Experimente durchzuführen und darzustellen. Insbesondere sollen die Messdaten live angezeigt werden, um den Verlauf des Experiments beobachten zu können.
Es gibt grundsätzlich zwei verschiedene Arten, wie MIMOSA verwendet werden kann: Messungen und Experimente. Bei einer Messung werden die Messdaten von MIMOSA empfangen und in Echtzeit auf dem Bildschirm als Kurvendiagramm dargestellt. Hierbei verhält sich die Experimentierumgebung rein passiv, d.h. es wird kein Einfluss auf das DUT genommen.
Experimente in der GUI bestehen aus einer Messung erweitert um einen Zeitplan, der den Ablauf des Experiments beschreibt. Dabei wird aktiv Einfluss auf die getestete Hardware genommen, indem zum Beispiel über eine serielle Schnittstelle Befehle gesendet werden. Um Experimente realisieren zu können bedarf es grafischer Elemente, mit Hilfe derer der Benutzer den Zeitplan erstellen und verwalten kann. Nach Abschluss eines Experiments oder einer Messung muss das Ergebnis gespeichert werden können, so dass es zu jedem Zeitpunkt wieder geladen werden kann. Dazu muss es entsprechende Optionen innerhalb der Benutzeroberfläche geben.

Zur Kommunikation mit der Testhardware können verschiedene Schnittstellen verwendet werden. Die GUI muss daher die Möglichkeit bieten eine der verfügbaren Schnittstellen

auszuwählen und diese zu konfigurieren. Der Dialog zur Konfiguration muss sich dabei der gewählten Schnittstelle anpassen und passende Felder für alle jeweils möglichen Parameter bieten.

Nachdem eine Messung oder ein Experiment durchgeführt wurde, muss der Anwender die Möglichkeit bekommen diese(s) mit Hilfe des grafischen Plots der Messdaten auszuwerten. Daraus ergeben sich folgende Anforderungen:

Skalieren / Zoomen

Die Achsen des Plots müssen sich wahlweise unabhängig voneinander oder zusammen skalieren lassen. Dadurch wird es möglich, den angezeigten Ausschnitt der Messdaten zu wählen und die Darstellungsgenauigkeit zu bestimmen.

Scrollen

Der Benutzer soll durch horizontales oder vertikales Scrollen den angezeigten Ausschnitt verschieben können. Dazu sind verschiedene Interaktionsmöglichkeiten vorstellbar, etwa das Mausrad, "Klicken und Ziehen" oder Schaltflächen neben und unter dem Plot.

Bereiche markieren

Es soll dem Nutzer möglich sein, Bereiche auf der horizontalen Achse auswählen zu können. Dazu kann beispielsweise die Maus bei gedrückter linker Maustaste verwendet werden. Wurde ein Bereich markiert, so sollen dazu verschiedene statistische Kennzahlen, wie der minimale, maximale und durchschnittliche Messwert, sowie der gesamte Energieverbrauch, angezeigt werden. Außerdem sollte es möglich sein, die Achsenskalierung automatisch so anzupassen, dass der markierte Bereich die komplette Anzeige füllt.

Anmerkungen hinzufügen und bearbeiten

Der Benutzer soll an beliebigen Stellen im Plot textuelle Anmerkungen erstellen können, die auch in Form von farblichen Markierungen an der entsprechenden Stelle dargestellt werden. Die bereits erstellten Anmerkungen sollen im Nachhinein bearbeitet oder wieder gelöscht werden können.

Umrechnen der Messwerte

Die Experimentierumgebung soll für eine möglichst große Anzahl von Experimenten geeignet sein. Da es je nach Anwendungsfall erforderlich sein kann, unterschiedliche physikalische Größen zu verwenden, muss es möglich sein, diese Umrechnung automatisch und unkom-

pliziert durchzuführen. Hierbei sollen mindestens die Größen Energie [Joule, J], Ladung [Coulomb, C], Strom [Ampere, A] und Leistung [Watt, W] unterstützt werden.

Kalibrieren

Wie in Abschnitt 1.1 beschrieben, kommen bei MIMOSA drei Integratoren zum Einsatz, die nacheinander die Messwerte liefern. Hierbei kann es zu Abweichungen der Werte der einzelnen Integratoren untereinander, sowie vom Sollwert geben. Um diese zu korrigieren soll es innerhalb der GUI die Möglichkeit zur Kalibrierung geben. Dazu ist es nötig, die Werte der Integratoren getrennt voneinander anzuzeigen, sowie alle relevanten Parameter für die Umrechnung in physikalische Größen einstellen zu können. Des Weiteren sollte der Benutzer einen Sollwert eingeben können, der gleichzeitig mit den realen Messwerten angezeigt wird, um so die Kalibrierung zu vereinfachen.

2.1.2. CLI

Die GUI stellt eine einfache Möglichkeit dar Experimente durchzuführen und gleichzeitig auszuwerten. Hierzu sind allerdings immer Benutzereingaben nötig (z.B. ein Klick auf „Start“), was eine grafische Benutzeroberfläche ungeeignet für Automatisierung macht. Dazu ist eine Kommandozeilenschnittstelle, oder Command Line Interface (CLI), besser geeignet, da sie sowohl vom Benutzer direkt, als auch von anderen Programmen und Skripten gesteuert werden kann. Um ein möglichst breites Anwendungsspektrum abzudecken, muss die Experimentierumgebung daher neben der GUI auch ein CLI enthalten.
Dieses muss genau wie die grafische Benutzeroberfläche alle Funktionen enthalten um Messungen und Experimente durchzuführen. Dazu zählen das Starten und Stoppen der Messung, die Kommunikation mit dem DUT, die Konfiguration der verwendeten Schnittstelle, sowie das Speichern der Daten.

2.1.3. Kommunikation

Eines der Hauptprobleme der bisherigen Messsoftware ist der fehlende Rückkanal zum DUT. Die neue Experimentierumgebung muss hierzu in der Lage sein. Da noch nicht alle Szenarien absehbar sind, in denen MIMOSA jemals eingesetzt wird, muss die Software mit jeglicher Art von Schnittstelle umgehen können. Hierbei ist mindestens das Senden und Empfangen von beliebigen Daten, sowie die Konfiguration der jeweiligen Schnittstelle erforderlich.

2.1.3.1. Erweiterbarkeit

Auch wenn die Experimentierumgebung mit dem Ziel entwickelt wird Messungen und Experimente aller Art zu ermöglichen, so kann es doch in Zukunft vorkommen, dass spezielle

Anwendungsfälle nicht abgedeckt werden. Daher ist es wichtig, dass die Software bei Bedarf erweiterbar ist. Damit es zum Beispiel möglich ist spezialisierte Benutzeroberflächen zu schaffen, müssen einheitliche Schnittstellen geschaffen werden, die eine einfache Anbindung an MIMOSA erlauben.

2.1.4. Speichern und Laden

Um Experimente jederzeit wiederholen zu können oder zu einem späteren Zeitpunkt auszuwerten, muss es möglich sein diese zu speichern und zu laden. Hierbei ist zu beachten, dass der von MIMOSA gelieferte Datenwert alleine noch keinen verwendbaren Wert darstellt, da es sich lediglich um den von MIMOSAs A/D-Wandler gelieferten Wert handelt. Zur Umrechnung in einen brauchbaren Wert ist außerdem die Spannung der Vorplatine, der verwendete Shunt-Widerstand, der Leerlaufoffset von MIMOSA, sowie die Spannung $U_{coaxMax}$ nötig (siehe Abschnitt 1.1). All diese Werte müssen zusammen mit den eigentlichen Nutzdaten abgespeichert werden, um das Ergebnis eines Experiments zu einem späteren Zeitpunkt wieder laden zu können. Weiterhin müssen auch die vom Benutzer hinzugefügten Anmerkungen gesichert werden. Wurde ein Experiment durchgeführt, d.h. es existiert ein Zeitplan, der der Ablauf des Experiments beschreibt, so muss dieser ebenfalls gespeichert werden.

2.1.5. Export

Um die Ergebnisse von Messungen und Experimenten auch unabhängig von der MIMOSA-Software darstellen und verarbeiten zu können ist es nötig, die Daten exportieren zu können. Hier gilt es zwei Anwendungsfälle zu unterscheiden:

Export zur Weiterverarbeitung der Daten

Es kann vorkommen, dass die Funktionen der MIMOSA-Software nicht ausreichen um ein Experiment hinreichend auszuwerten. Daher muss es eine Möglichkeit geben, die Messdaten in einem Format zu exportieren, dass es anderen Anwendungen erlaubt diese einzulesen und zu verarbeiten. Hierzu eignet sich beispielsweise das CSV-Format (CSV, Comma Seperated Value).

Export als Grafik

Werden die Messdaten zur Verwendung in einer Arbeit oder einer Präsentation benötigt, so wird in der Regel eine grafische Darstellung der Messkurve gewünscht. Daher muss es möglich sein diese in einem üblichen Grafikformat wie JPG oder PNG zu exportieren. Die Grafik sollte wahlweise die komplette Messkurve oder nur einen Ausschnitt beinhalten. Der Benutzer muss außerdem die Möglichkeit haben, die Abmessungen der Grafik festzulegen.

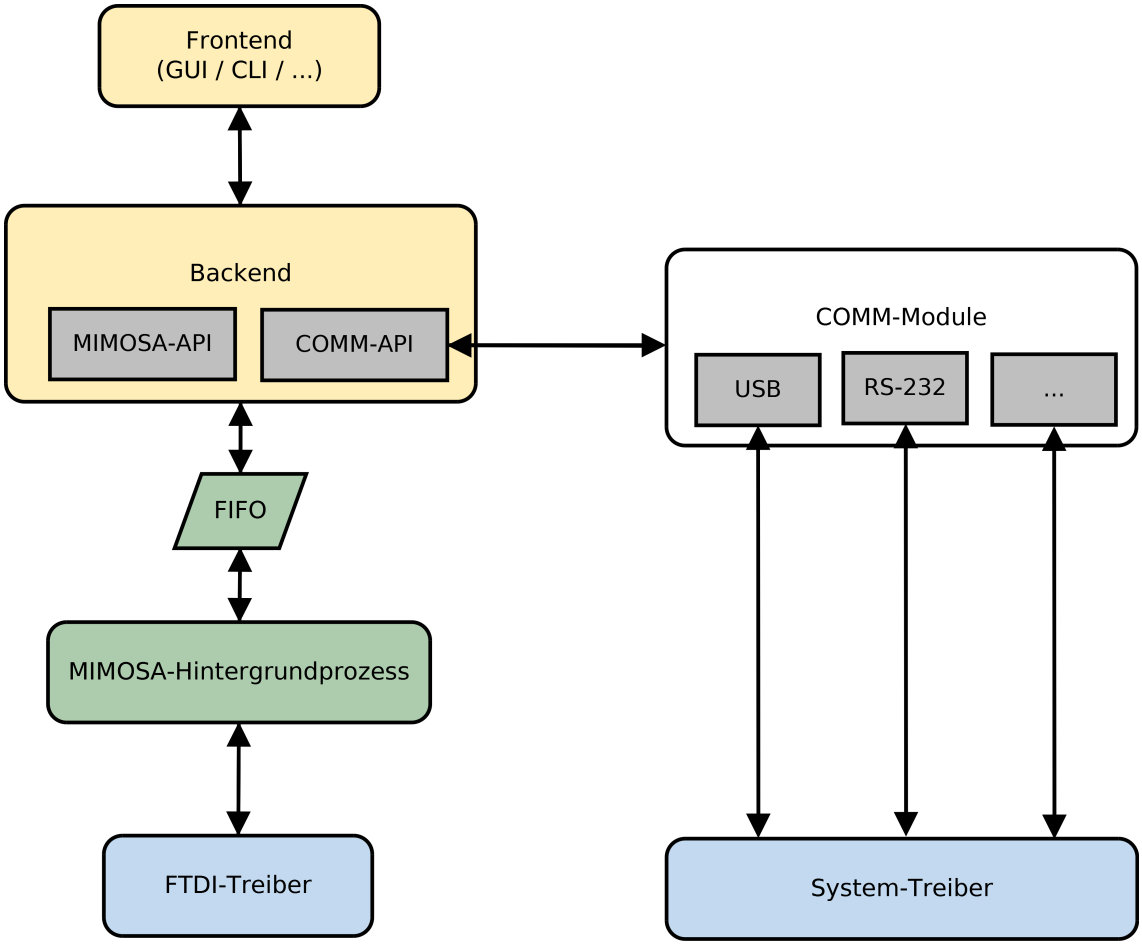


Abbildung 2.1.: Konzept

2.1.6. Integrität des Datenstroms

MIMOSA arbeitet mit einer Samplingrate von 100 kHz, was bedeutet, dass 100000 Worte pro Sekunde von MIMOSA über die USB Schnittstelle übertragen werden. Es muss sichergestellt werden, dass all diese Daten empfangen werden und es nicht zu Unterbrechungen im Datenstrom kommt.

2.1.7. Datenkompression

Um den Speicherbedarf der Ergebnisse von Messungen und Experimenten möglichst gering zu halten sollten die Daten komprimiert abgespeichert werden. Wichtig ist hierbei, dass die Daten nach der Dekompression wieder exakt den Originaldaten entsprechen. Daher ist ein verlustfreies Kompressionsverfahren zu wählen.

2.2. Konzept

In den vorherigen Abschnitten wurden die Anforderungen untersucht, die die neue Experimentierumgebung erfüllen muss. Darauf aufbauend wird nun ein Konzept entwickelt, welches diese umsetzt.

Ein zentraler Aspekt bei der Entwicklung des Konzepts ist die Entwicklung im Hinblick auf die Erweiterbarkeit der Software. Um diese zu gewährleisten, ist es sinnvoll, ein modulares System zu entwickeln, bei dem jedes Modul eine bestimmte Aufgabe übernimmt. Dies verbessert die Übersichtlichkeit und erleichtert somit die Wartung der Software. Außerdem wird die Umsetzung so erleichtert, da jede Komponente unabhängig von den anderen implementiert und getestet werden kann.
Abbildung 2.1 zeigt die verschiedenen Module der Experimentierumgebung. Auf unterster Ebene befinden sich die Treiber. Im Falle von MIMOSA sind dies die Treiber für den FTDI USB-Chip[1]. Diese werden, genau wie die Treiber für die verschiedenen Kommunikationsschnittstellen (Modul „Systemtreiber“), als gegeben angesehen und sind nicht Teil dieser Arbeit, weshalb sie hier nicht weiter behandelt werden.

2.2.1. MIMOSA-Hintergrundprozess

Das erste hier relevante Modul ist der MIMOSA-Hintergrundprozess. Dieser kommuniziert über die FDTI-Treiber direkt mit MIMOSA und liest die Daten aus dem Puffer des FTDI-Chips aus. Dieser ist 384 Bytes groß [7], was bedeutet, dass bei einer Wortgröße von 3 Byte maximal 128 Worte zwischengespeichert werden können. Bei der von MIMOSA verwendeten Samplingrate von 100 kHz ist der Puffer daher nach maximal 1,28 ms voll. Dauert es bis zum nächsten Auslesen des Puffers mehr als diese 1,28 ms, so kommt es zum Datenverlust. Der Hintergrundprozess muss daher so ausgelegt werden, dass diese Zeitschranke eingehalten wird. Eine Möglichkeit dies zu erreichen, ist eine Echtzeitschedulingstrategie zu verwenden. Eine geeignete Strategie ist First In - First Out (FIFO). Wird diese in Verbindung mit der höchsten Prozesspriorität verwendet, so ist sichergestellt, dass der Prozess niemals verdrängt wird, was zur Verletzung der Zeitschranke führen könnte.

2.2.2. FIFO

Die vom Hintergrundprozess empfangenen Daten müssen den höheren Schichten der Experimentierumgebung bereitgestellt werden. Damit die Echtzeitanforderungen auf den Hintergrundprozess beschränkt bleiben, muss dies über einen Puffer geschehen, in den der Hintergrundprozess schreiben kann und aus dem die Daten später ausgelesen werden können. Dieser Puffer muss dabei groß genug sein um einige Sekunden lang Daten aufnehmen

[1] MIMOSA verwendet einen FT245BL Chip von FTDI

zu können, damit keine Daten verloren gehen, falls es auf den höheren Ebenen zu kurzzeitigen Verzögerungen kommt (z.B. durch eine hohe Auslastung der CPU). Als beste Option für diesen Zweck stellte sich eine sogenannte „Named Pipe“ oder FIFO (kurz für First In - First Out) heraus. FIFOs sind eine spezielle Art der Interprozesskommunikation. Dabei wird eine spezielle Datei variabler Größe im Dateisystem angelegt, auf die alle Prozesse mit ausreichenden Rechten zugreifen können. Die Abkürzung FIFO geht dabei auf die Funktionsweise dieser Dateien zurück: Die Daten werden wieder in der Reihenfolge ausgelesen, in der sie ursprünglich in die FIFO geschrieben wurden. Die maximale Größe der FIFO lag auf allen getesteten Systemen bei 1048576 Bytes, was ausreichend groß für den angestrebten Zweck ist. Auf Grund dieser Funktionalität ist eine Named Pipe bestens geeignet um die Daten von MIMOSA dem Backend zur Verfügung zur stellen.

2.2.3. Backend

Das Backend stellt Funktionen bereit, die essentiell für die Experimentierumgebung sind und es dem Frontend beispielsweise ermöglichen auf die Daten von MIMOSA zuzugreifen. Das Backend ist dabei in zwei Teile gegliedert: Die MIMOSA-API und die COMM-API. Die MIMOSA-API stellt alle Funktionen bereit um die Verbindung mit MIMOSA herzustellen, auf die Daten zuzugreifen und diese zu verarbeiten. Außerdem stellt die MIMOSA-API eine einheitliche Schnittstelle zum Speichern und Laden von Messdaten und Experimenten zur Verfügung, so dass es zum Beispiel möglich ist das Ergebnis eines Experiments, das mit dem CLI durchgeführt wurde, im GUI Frontend zu laden und auszuwerten.
Die COMM-API stellt die Verbindung zwischen dem Frontend und den Kommunikationsinterfaces dar. Hier gibt es allgemeine Funktionen zum Senden und Empfangen von Daten, die vom Frontend genutzt werden können, ohne das genaue Informationen über das jeweils verwendete Interface nötig sind. Auch die Konfiguration der Interfaces ist über die COMM-API möglich. Des Weiteren ermöglicht die COMM-API eine Übersicht über alle verfügbaren COMM-Module.

2.2.4. Frontend

Das Frontend ist der Teil der Experimentierumgebung, mit dem der Benutzer direkt in Kontakt kommt. Hier wird die Funktionalität des Backends genutzt und gegebenenfalls erweitert um spezielle Aufgaben durchführen zu können (beispielsweise eine grafische Darstellung der Messwerte im GUI Frontend). Eine genauere Betrachtung der Aufgaben und Anforderungen an die Frontendkomponenten wurde bereits in den Abschnitten 2.1.1 und 2.1.2 durchgeführt.

2.2.5. COMM-Module

Die COMM-Module setzten die allgemeinen Funktionen der COMM-API zum Senden, Empfangen und Konfigurieren in spezifischer Weise um. Sie dienen also als „Wrapper“, für die Treiberfunktionen des jeweiligen Interfaces und erlauben so die Anbindung an die Experimentierumgebung. Da der Benutzer die Comm-Module später über das Frontend konfigurieren können soll, muss es eine Möglichkeit geben, eine Übersicht über die Verfügbaren Parameter zu bekommen. Dazu gibt es für jedes COMM-Modul eine XML-Datei, die dieses definiert und die Parameter, sowie die jeweils gültigen Werte, festlegt. In Abschnitt 3.4.1 wird dies detaillierter beschrieben.

3. Implementierung

In diesem Kapitel wird die Implementierung der Experimentierumgebung beschrieben. Dabei beschäftigt sich jeder Abschnitt mit einem Modul des Konzepts aus Abschnitt 2.2. Den Anfang macht hier das Modul „Hintergrundprozess“. Darauf aufbauend werden die anderen Module der Reihe nach beschrieben. Den Kapitelabschluss bildet ein Abschnitt über das verwendete Dateiformat sowie die Komprimierung.

3.1. Hintergrundprozess und FIFO

Der Hintergrundprozess stellt die Verbindung zu MIMOSA her, liest die Daten aus dessen Puffer aus und schreibt sie anschließend in die FIFO, um sie den höheren Schichten der Experimentierumgebung bereitzustellen. Das Verbinden mit MIMOSA und Auslesen der Daten ist bereits in der alten MIMOSA Software implementiert. Der neue Hintergrundprozess baut auf diesem Programm auf und ergänzt es um die erforderliche Funktionalität, die für die Anbindung an die Experimentierumgebung nötig ist.

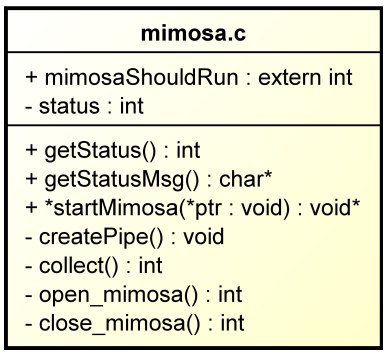


Abbildung 3.1.: mimosa.c

Die Modifikationen der existierenden Software beschränken sich auf die Datei mimosa.c. Abbildung 3.1 gibt einen Überblick über die Funktionen und Variablen des Programms. Aus der alten Software übernommen wurden hier die Funktionen *open_mimosa()*, *close_mimosa()*, sowie Teile der Funktion *collect()*.
Um später über die MIMOSA-API feststellen zu können ob z.B. gerade eine Verbindung mit MIMOSA besteht, wurde die Variable *status*, sowie Funktionen zum Anfragen dieser

Variablen eingeführt. Des Weiteren lässt sich der Hintergrundprozess über die Variable *mimosaShouldRun* beenden, wie die Abbildungen 3.2 und 3.3 zeigen. In Abbildung 3.2 ist der Ablauf des Hintergrundprozesses dargestellt. Nach dem Starten wird zunächst die Verbindung mit MIMOSA hergestellt. Bei erfolgreichem Verbinden wird dies durch ein Setzen der Variable *status* kenntlich gemacht. Anschließend wird die FIFO unter dem Namen */tmp/-mimosafifo* erstellt. In der Funktion *collect()* erfolgt daraufhin das eigentliche Empfangen und Verarbeiten der Daten von MIMOSA. Im Normalfall sollte die Funktion *collect()* erst verlassen werden, wenn mit Hilfe der MIMOSA-API die Variable *mimosaShouldRun* auf 0 gesetzt wird. Da es aber auch zum Beispiel durch einen Fehler bei der Verbindung zum Verlassen der Funktion kommen kann, befindet sich der gesamte eben beschriebene Ablauf in einer Schleife, die dafür sogt, dass die Verbindung direkt wieder hergestellt wird, sofern der Abbruch nicht gewollt war.

Das Aktivitätsdiagramm 3.3 zeigt die Funktion *collect()*. Nach dem Öffnen der FIFO und Setzen der Statusvariable befindet sich die Funktion in einer Schleife, die nur durch das Setzen von *mimosaShouldRun* auf 0, oder einen Fehler bei der Verbindung durch MIMOSA, unterbrochen wird. Hier werden zunächst die Daten aus dem Puffer von MIMOSA gelesen. Die gelesenen Daten werden daraufhin in das richtige Wortformat umgewandelt, wofür ein Parser verantwortlich ist. Dieser war bereits Teil der alten MIMOSA Software, weshalb hier kurz auf seine Arbeitsweise eingegangen wird: Das Parsen des Datenstroms erfolgt in zwei Schritten. Im ersten Schritt wird der Anfang des ersten Datenworts ermittelt. Dieses ist daran zu erkennen, dass es sich nach dem ersten Syncronisationswort (siehe Abschnitt 1.1.2) befindet. Schritt Zwei des Parsers besteht anschließend darin, jeweils 3 Byte nach dem Synchronisationswort zu einem Datenwort vom Typ *word_ t* zusammenzufassen.
Anhand der Anzahl der erfolgreich gelesenen Worte wird in der Funktion *collect()* überprüft, ob es zu einem Fehler beim Lesen gekommen ist. Ist dies der Fall, was sich daran erkennen lässt, dass kein vollständiges Wort gelesen wurde, so wird ein Errorwort, *0xA000000*, in die FIFO geschrieben. Durch dieses Wort können Fehler später im Back- oder Frontend erkannt und behandelt werden. War das Lesen von MIMOSA hingegen erfolgreich, so werden alle gelesenen Worte in die FIFO geschrieben und der Zyklus beginnt von vorne.

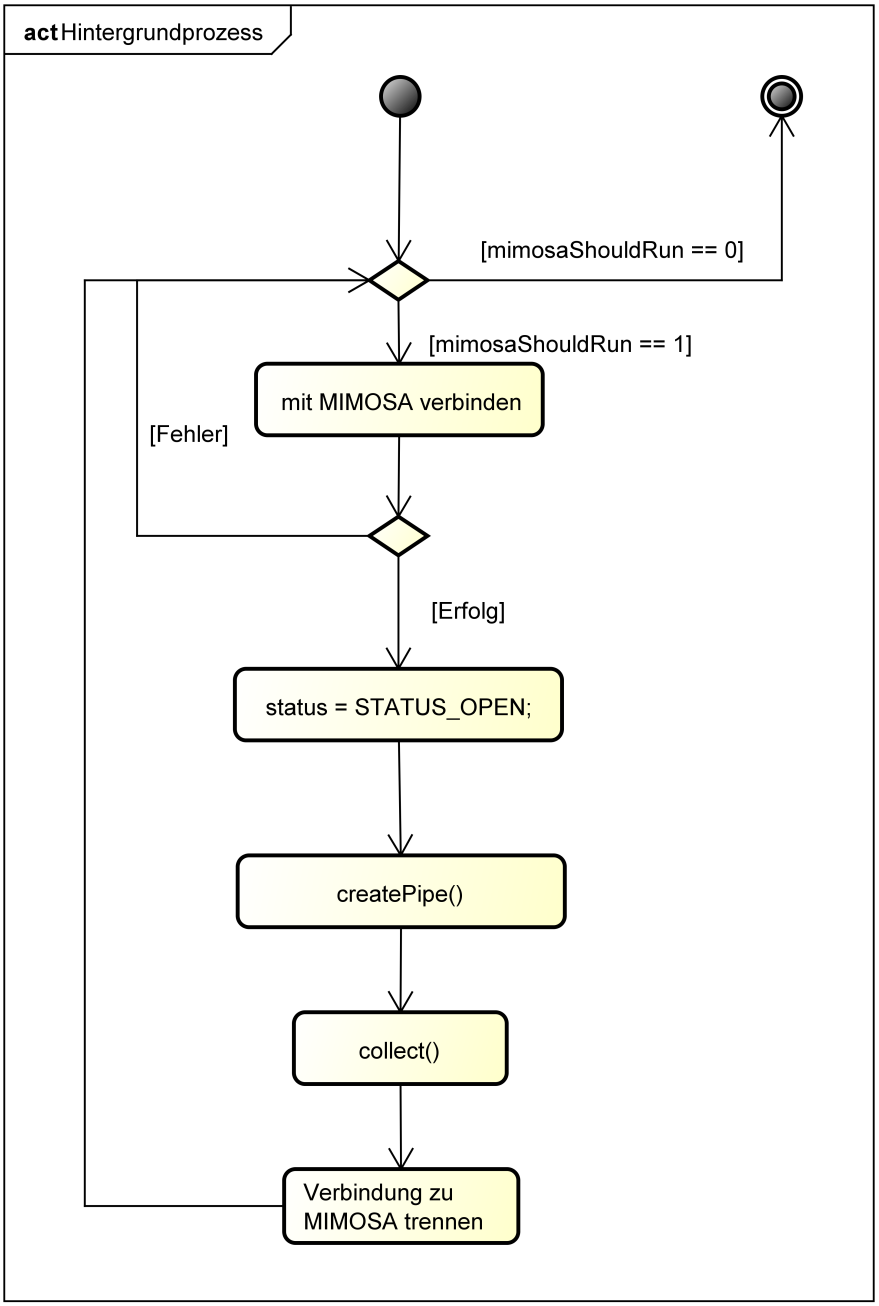


Abbildung 3.2.: Ablauf des Hintergrundprozesses

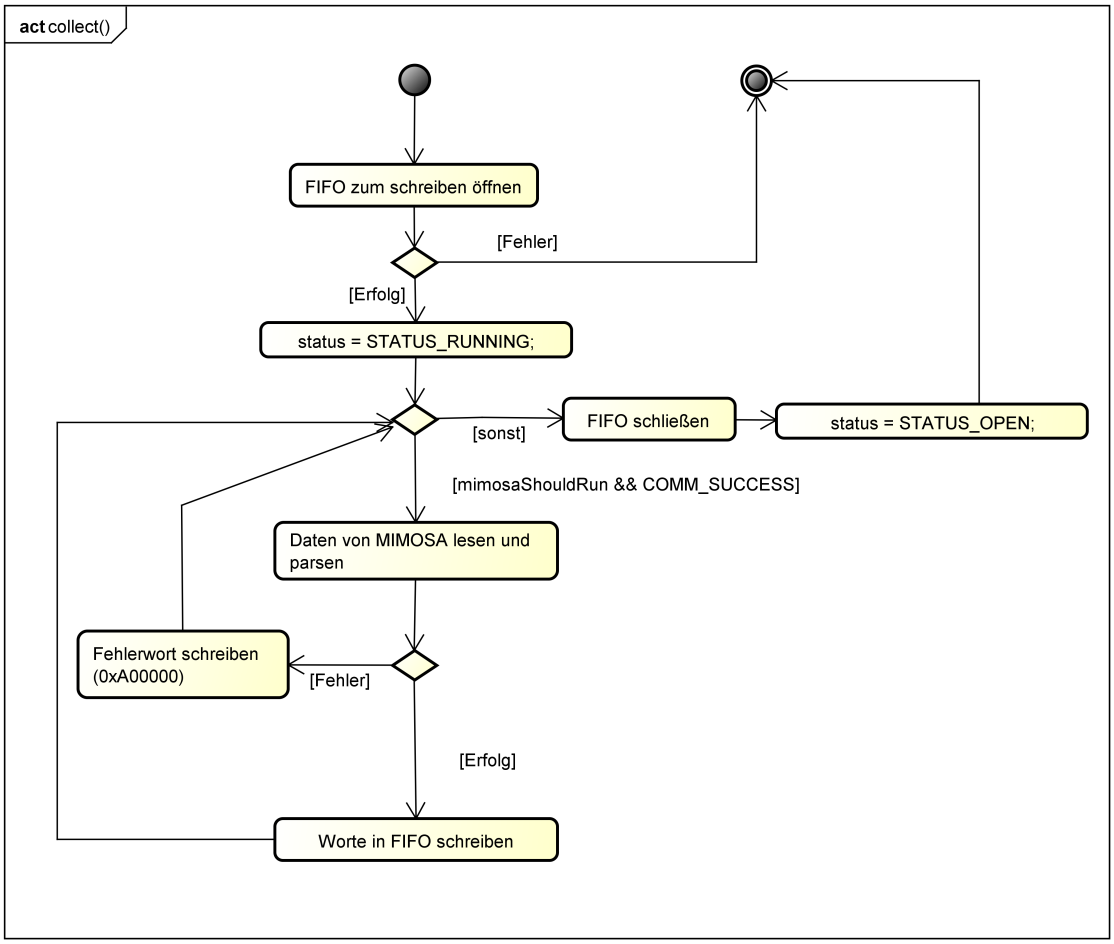


Abbildung 3.3.: collect() Funktion des Hintergrundprozesses

3.2. Backend

Das Backend stellt über die in den Abbildungen 3.4 und 3.6 zu sehenden Klasse Funktionen für den Zugriff und die Verarbeitung der Daten von MIMOSA bereit. Es wird in C++ als dynamische Bibliothek („shared library“) implementiert. Diese Bibliothek kann von anderen Programmen, wie dem GUI- und dem CLI-Frontend, eingebunden werden.

3.2.1. Die Klasse *MimosaAPI*

Die Klasse MimosaAPI ist für das Starten und Stoppen des Hintergrundprozesses verantwortlich. In der Funktion *start()* wird dazu ein neuer Thread gestartet, in dem das Programm aus 3.1 ausgeführt wird. Die Schedulingstrategie für diesen Thread wird auf FIFO gesetzt. Dabei handelt es sich um eine Echtzeitschedulingstrategie, die sicherstellt, dass der Thread nicht verdrängt wird, was zur Verletzung der in 2.2.1 beschriebenen Zeitschranke führen könnte.
Eine weitere wichtige Aufgabe dieser Klasse ist das Ermöglichen des Zugriffs auf die Daten von MIMOSA. Hierzu wird die Funktion *int readData(word_t *buf, int count)* bereitgestellt, die bis zu *count* MIMOSA-Worte aus der FIFO ausliest und sie in den vom Aufrufer (in der Regel das Frontend) bereitgestellten Speicherbereich **buf* kopiert.
Über die Funktionen *isConnected()*, *getStatusCode()* und *getStatusString()* kann der aktuelle Status von MIMOSA abgefragt werden und so z.B. im Falle eines Fehlers eine entsprechende Fehlermeldung ausgegeben werden.

3.2.2. Die Klasse *MimosaFileManager*

Die Klasse *MimosaFileManager* dient der Verwaltung aller Daten und Dateien der Experimentierumgebung. Sie stellt Funktionen zum Speichern, Laden und Exportieren von Experimenten und Ergebnissen bereit. Beim Speichern sind zwei Fälle zu unterscheiden: persistentes Speichen und temporäres Speichern. Mit persistentem Speichern wird dabei der Vorgang bezeichnet, der vom Benutzer eingeleitet wird, um das Ergebnis eines Experiments, inklusive aller zugehöriger Daten, auf der Festplatte zu speichern, so dass es später wieder geladen werden kann. Hierfür ist die Funktion *save()* verantwortlich. In ihr werden alle temporären Dateien zu einem Archiv zusammengefasst und komprimiert gespeichert. Während ein Experiment läuft werden temporäre Dateien angelegt, die alle relevanten Messdaten enthalten. Für das Speichern der von MIMOSA empfangenen Daten sind die Funktionen *addWord()*, und *addWords()* verantwortlich. Des Weiteren legen diese Funktionen sogenannte *Skalierungsdateien* an. Diese dienen folgendem Zweck: bei der Durchführung von Experimenten entstehen auf Grund der hohen Abtastrate von MIMOSA schnell sehr viele Datenpunkte. Sollen diese nun im GUI-Frontend angezeigt werden, so wird es die Leistung des Systems stark beeinträchtigen, wenn Millionen von Messwerten gleichzeitig

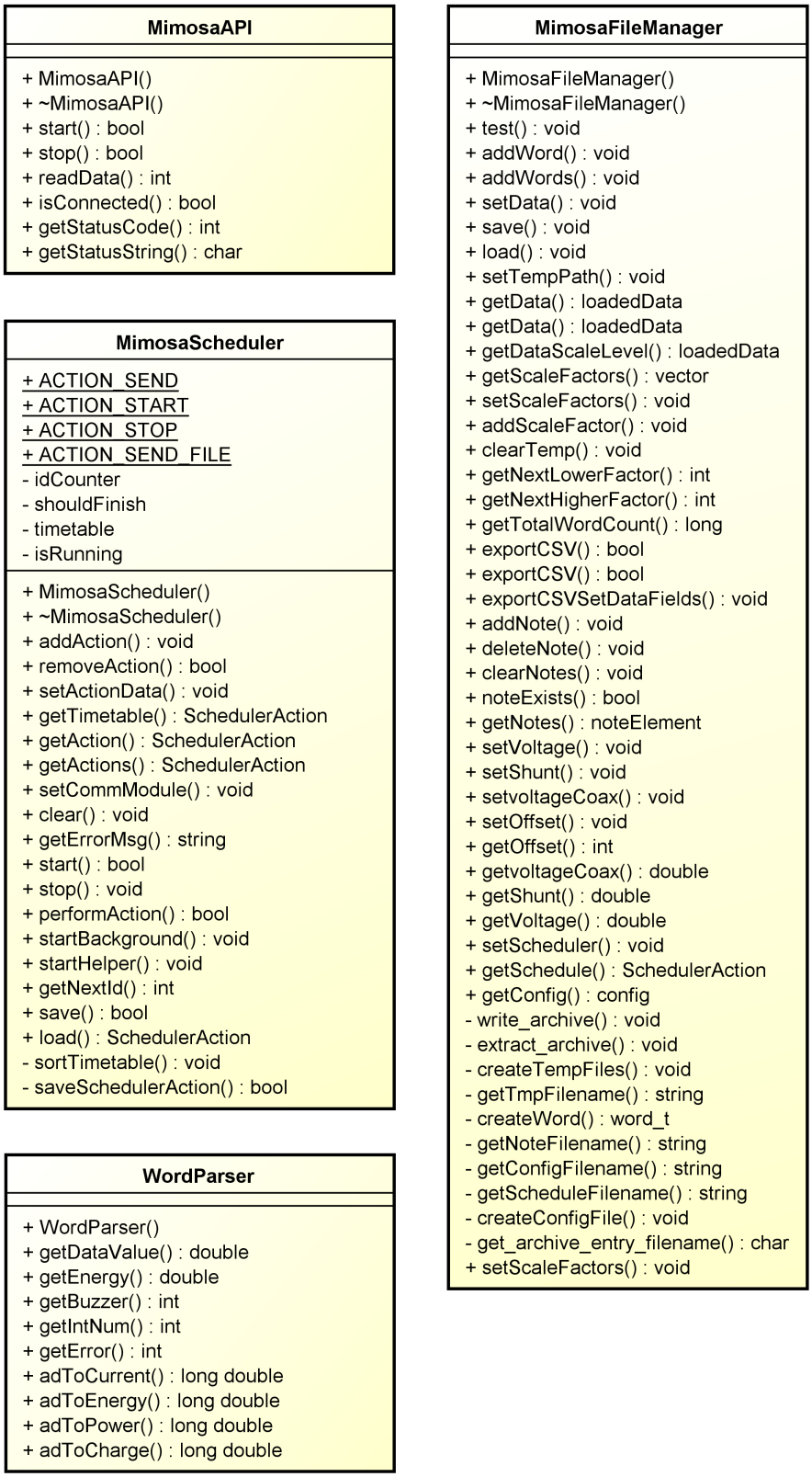


Abbildung 3.4.: Klassen der MIMOSA-API

dargestellt werden müssen. Um dies zu verhindern existieren die Skalierungsdateien. Hier werden mehrere Messwerte zu einem Wert zusammengefasst, um so die Anzahl der darzustellenden Werte zu verringern. Beispielsweise werden für die Skalierungsstufe 10 immer 10 MIMOSA-Worte zu einem neuen Skalierungswort zusammengefasst. Der Datenwert dieses Wortes ergibt sich dabei als Mittelwert aller 10 Datenwerte. Das Bit für den Buzzereingang von MIMOSA wird im neuen Wort 1 sein, sofern es in mindestens einem der 10 Worte 1 war. Die Bits für die Nummer des verwendeten Integrierers sind in einem Skalierungswort immer 0, da es hier keinen Sinn machen würde diesen Wert zu mitteln.
Die Klasse MimosaFileManager verwaltet für jede Skalierungsstufe eine Struktur vom Typ *struct scalelevel*. In dieser wird der Zwischenstand einer Skalierungsdatei gespeichert. Werden beispielsweise mit der Funktion *addWords()* 15 Worte hinzugefügt, so werden alle 15 Worte in die Datei für die Stufe 1 geschrieben. In die Datei für Stufe 10 wird ein Wort geschrieben und die Daten der restlichen fünf Worte müssen zwischengespeichert werden, da sie kein komplettes Skalierungswort ergeben.
Die einfachste Möglichkeit dieses System zu implementieren, ist jeweils nur ein neues Wort hinzuzufügen, was in der Funktion *addWord()* geschieht. So kann bei jedem Aufruf überprüft werden, ob sich aus den zwischengespeicherten Daten und dem neu hinzugefügten Wort ein komplettes Wort für die jeweilige Skalierungsstufe erzeugen lässt, welches anschließend auf die Festplatte geschrieben werden kann. Diese Methode sogt allerdings dafür, dass bei jedem Aufruf auf die Festplatte geschrieben wird, da die Skalierungsdatei für Stufe 1 alle Worte von MIMOSA enthalten muss. Da Festplattenzugriffe vergleichweise langsam sind, hat diese Methode einen negativen Einfluss auf die Leistung des Systems. Um die Anzahl der Festplattenzugriffe zu reduzieren wurde die Funktion *addWords()* implementiert. Diese bekommt eine Menge von Worten übergeben und führt daraufhin für jede Skalierungsstufe den in Abbildung 3.5 vereinfacht dargestellten Ablauf aus.
Dabei sind zwei grundlegende Fälle zu unterscheiden. Im ersten Fall reichen die hinzugefügten Worte nicht aus um ein komplettes Wort für die entsprechende Skalierungsstufe zu erzeugen. In diesem Fall werden die Informationen der Worte nur zwischengespeichert.
Lässt sich allerdings mindestens ein komplettes Wort erzeugen, so wird dieses erzeugt und in einem Puffer zwischengespeichert. Lassen sich aus den verbleibenden Worten noch weitere komplette Skalierungsworte erzeugen, so werden auch diese erzeugt und gepuffert. Die Daten aller eventuell verbleibenden Worte werden danach in dem entsprechenden *struct scaleLevel* zwischengespeichert, da sich daraus kein komplettes Wort mehr erzeugen lässt. Anschließend wird der gesamte Puffer mit kompletten Worten auf die Festplatte geschrieben. Zur Verdeutlichung dieses Vorgangs soll ein kleines Beispiel für die Skalierungsstufe 10 dienen:
Zuerst erfolgt der Aufruf *addWords(wordbuffer, 5)*. Es sollen also fünf Worte hinzugefügt werden, die in dem Puffer wordbuffer stehen. Da fünf Worte kein komplettes Skalierungswort für die Stufe 10 ergeben, werden die Daten nur im entsprechenden *struct scalelevel*

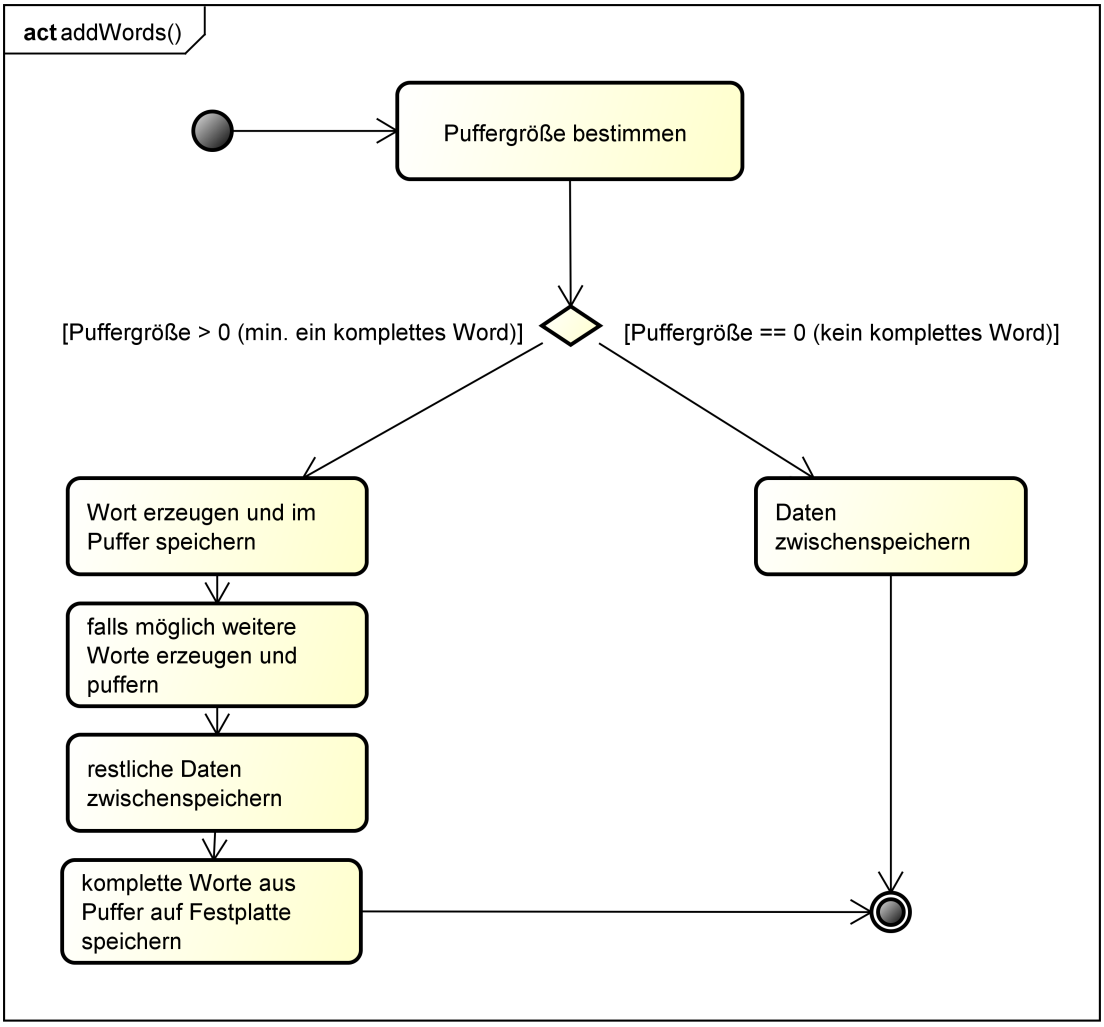


Abbildung 3.5.: Vereinfachter Ablauf der Funktion addWords()

zwischengespeichert. Anschließend werden mit dem Aufruf *addWords(wordbuffer, 30)* 30 weitere Worte hinzugefügt. Aus den Daten der fünf Worte des ersten Aufrufs und der ersten fünf Worte dieses Aufrufs wird nun ein komplettes Wort erstellt und in einem neuen Puffer gespeichert. Aus den verbleibenden 25 Worten lassen sich zwei weitere komplette Worte für die Skalierungsstufe 10 erstellen, welche ebenfalls im Puffer gespeichert werden. Es verbleiben fünf Worte, deren Daten wieder im *struct scaleLevel* zwischengespeichert werden. Der komplette Puffer, der nun 3 Worte enthält, kann jetzt auf einmal auf die Festplatte geschrieben werde. Dadurch wurden in diesem Beispiel zwei Festplattenzugriffe gegenüber der in *addWord()* verwendeten Methode gespart.

3.2.3. Die Klasse *Wordparser*

Die Klasse Wordparser besitzt Funktionen um das von MIMOSA verwendete Datenwortformat auszuwerten. So können alle in einem Datenwort enthaltenen Informationen (Datenwert, Nummer des Integrierers und Wert des Buzzersignals) einzeln extrahiert werden. Dieser Vorgang wird im Folgenden beispielhaft an der Funktion *getDataValue()* erläutert.

```
1  double WordParser::getDataValue(word_t word)
2  {
3      if(word == SYNCWORD){
4          return ERROR_SYNCWORD;
5      }
6      if(getError(word) == ERROR){
7          return ERROR;
8      }
9      return (((word) & 0x0FFFF0) >> 4);
10 }
```

Listing 3.1: Die Funktion Wordparser::getDataValue()

Die ersten beiden Abfragen dienen der Fehlerbehandlung. Das eigentliche Extrahieren des Datenwerts findet in Zeile 9 statt. Die Bits 4-19 eines Wortes enthalten den Datenwert. Um diesen zu erhalten werden zunächst durch eine bitweise UND-Verknüpfung mit der Maske 0x0FFFF0 alle nicht relevanten Bits des Datenworts ausgeblendet. Abschließend müssen alle Bits noch um vier Stellen nach rechts verschoben werden, so dass das Datenbit D0 an der Stelle des LSB des Wortes steht.
Die Funktionen zum Extrahieren der Integrierernummer und des Buzzersignals arbeiten nach dem gleichen Prinzip.
Darüber hinaus gibt es in der Klasse Wordparser Funktionen um den extrahierten Datenwert gemäß den Formeln in 1.1.3 in physikalische Größen umzurechnen.

3.2.4. Die Klasse *MimosaScheduler*

Die Klasse *MimosaScheduler* erlaubt das Erstellen und Ausführen von einfachen Zeitplänen, wie sie für Experimente mit der GUI nötig sind. Unterstützt werden das Starten und Stoppen des Hintergrundprozesses, sowie das Senden von beliebigen Daten oder ganzen Dateien.
Um einen Zeitplan zu erstellen müssen zunächst Aktionen vom Typ *struct SchedulerAction* erstellt werden (siehe Listing 3.2). Diese definieren die Art der Aktion, sowie ihren Ausführungszeitpunkt. Der Ausführungszeitpunkt wird hier in Sekunden relativ zum Start der Ausführung angegeben. Die Zeiteinheit Sekunden wurde gewählt, da bei einer genaueren Unterteilung, wie z.B. Millisekunden, nicht garantiert werden könnte, dass diese auch

eingehalten wird. Bei einer Angabe in Sekunden ist der exakte Ausführungszeitpunkt hingegen nicht von ganz so großer Bedeutung, da der Benutzer keine auf die Millisekunde genaue Ausführung erwartet.

```
1 struct SchedulerAction{
2     int id;
3     int timeInSec = 0;
4     int action = 1;
5     void *data = NULL;
6     long dataSize = 0;
7     string filename;
8 }
```

Listing 3.2: Das struct SchedulerAction

Einem Zeitplan können beliebig viele Aktionen hinzugefügt werden. Die Funktion *start()* startet die Ausführung des Zeitplans im Hintergrund. Dazu werden zunächst alle Aktionen aufsteigend nach der Zeit sortiert und anschließend in dieser Reihenfolge ausgeführt. Es ist möglich mehrere Aktionen mit dem gleichen Ausführungszeitpunkt hinzuzufügen, in diesem Fall kann allerdings die Ausführungsreihenfolge der betroffenen Aktionen nicht garantiert werden.

3.2.5. Die Klasse *MimosaCommModuleManager*

Die Klasse *MimosaCommModulemanager* gehört zum COMM-API Teil des Backends. Sie verwaltet einen Vektor *vector<CommModule> modules* der alle vorhandenen Comm-Module enthällt. Die get-Funktionen ermöglichen dem Frontend einen Zugriff auf diese Module.

3.2.6. Die Klasse *CommModule*

Die Klasse *CommModule* ist eine abstrakte Klasse, die als Basisklasse für die Implementierung von Comm-Modulen dient. Jedes Comm-Module hat einen Namen (*moduleName*), eine ID (*moduleId*), eine zugewiesene Config-Datei (siehe Abschnitt 3.4.1), sowie Funktionen um diese zu setzen und abzufragen. Darüber hinaus definiert die Klasse rein virtuelle Funktionen (kursiv im Klassendiagramm), die bei der Implementierung der Comm-Module überschrieben werden müssen. Dadurch wird sichergestellt, dass alle Comm-Module alle erforderlichen Funktionen haben. Die genaue Implementierung und Verwendung von Comm-Modulen wird in Abschnitt 3.4 beschriebene.

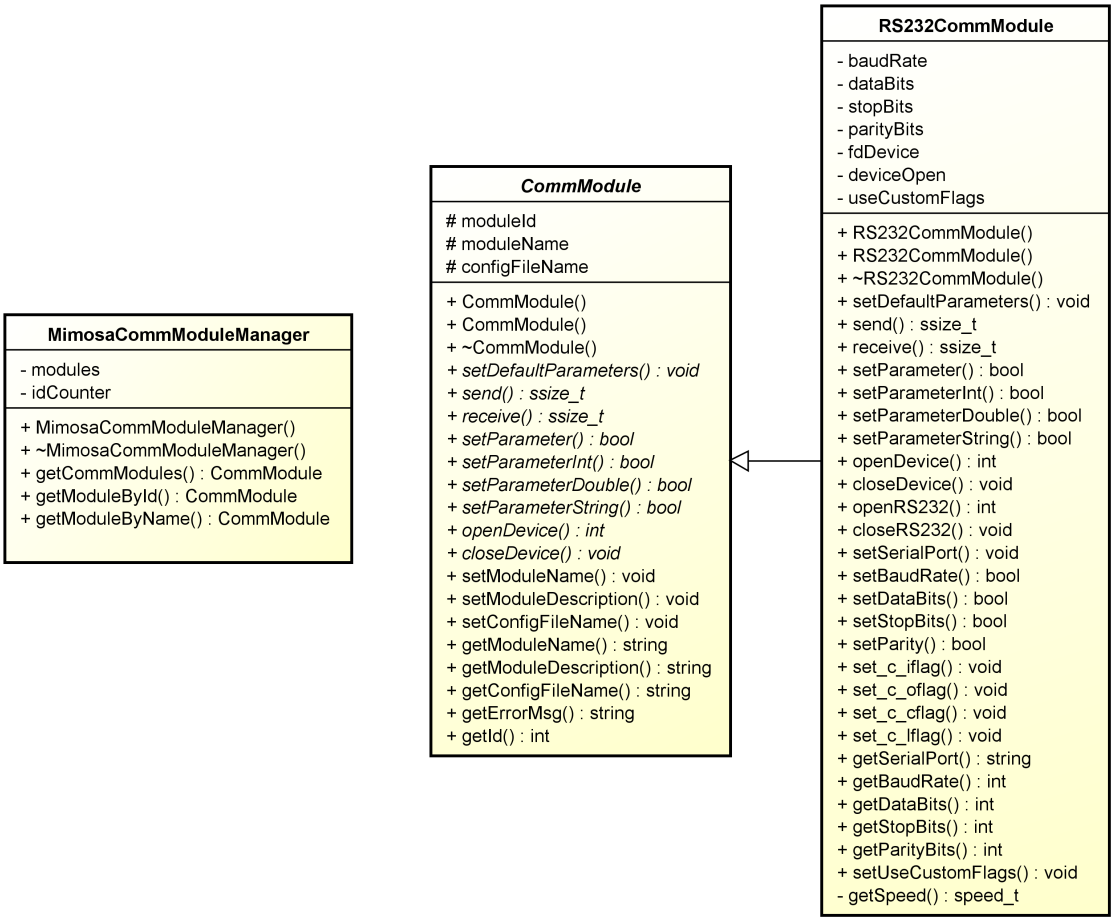


Abbildung 3.6.: Klassen der COMM-API und eine Beispielimplementierung für ein RS232 Comm-Modul

3.3. Frontend

3.3.1. GUI

Das GUI-Frontend stellt dem Benutzer durch grafische Elemente alle Funktionen der Experimentierumgebung zur Verfügung. Es greift dazu auf die Funktionalität der im letzten Abschnitt beschriebenen APIs zurück und ist daher ebenfalls in C++ implementiert. Dabei kommt Qt 5.3 [5] als Grafikbibliothek zum Einsatz. Für die Darstellung der Messwerte wird die Bibliothek QCustomPlot 1.2.1 verwendet [4]. Letztere wurde für diese Arbeit wie in A.3 beschrieben modifiziert um den Anforderungen dieser Arbeit gerecht zu werden.

Die Oberfläche ist in 9 Bereiche unterteilt, wie der Screenshot in Abbildung 3.7 zeigt. Der erste Bereich ist die Menüleiste (1), über die sich beispielsweise Experimente speichern und laden lassen.
Bereich 2 ziegt die Buttons zum Starten und Stoppen eines Experiments. Das Drücken des Buttons „Start“ löst eine Funktion aus, die kontinuierlich mit Hilfe der Klasse *MimosaAPI* Daten von MIMOSA liest. Die gelesenen Daten werden an eine Instanz der Klasse *MimosaFileManager* übergeben und wie in Abschnitt 3.2.2 beschrieben gespeichert. Gleichzeitig werden die Werte im Bereich 8 live dargestellt. Da eine Darstellung aller Daten bei einer Abtastrate von 100 kHz die Leistung des Systems zu stark beeinträchtigt, wird für einen Datenpunkt der Live-Darstellung immer der Mittelwert aus jeweils 1000 Werten gebildet. Die Y-Achse (in der Grafik mit „Strom [nA]“ bezeichnet), passt sich dabei automatisch dem sichtbaren Wertebereich an. Die Zeit-Achse zeigt jeweils die Werte der letzten 6 Sekunden. Wurde das Experiment mit dem Button „Stop“ angehalten, so werden die zuvor gespeicherten Daten geladen und die komplette Messkurve wird in Bereich 9 angezeigt. Dabei wird automatisch die entsprechende Skalierungsdatei gewählt, so dass maximal 1000000 Werte geladen werden. Dieser Wert hat sich bei Tests als bester Kompromiss zwischen Leistung (Ladezeit und Darstellungszeit im Graph) und Genauigkeit herausgestellt.
Im Bereich 3 wird der aktuell gewählte Messwert, sowie der Mittelwert, das Minimum und das Maximum aller gewählten Werte angezeigt. Dies erleichtert das Auswerten von Experimenten. Außerdem befindet sich in diesem Bereich das Feld „Gesamtverbrauch“, das den gesamten Energieverbrauch des gewählten Bereichs oder der kompletten Messkurve anzeigt. Diese Funktion ermöglicht es z.B. verschiedene Bereiche der Messkurve miteinander zu vergleichen um so Energiemodelle aufzustellen. Um dies weiter zu vereinfachen gibt es außerdem die Möglichkeit die Werte der Messkurve zu filtern, so dass beispielsweise nur die Messwerte zum Gesamtverbrauch zählen, bei denen das Buzzersignal 1 ist.
In Bereich 6 kann der Benutzer ein Comm-Interface wählen, sowie einen Zeitplan für das Experiment zusammenstellen. Dazu wird die Klasse *MimosaScheduler* verwendet (siehe Abschnitt 3.2.4).

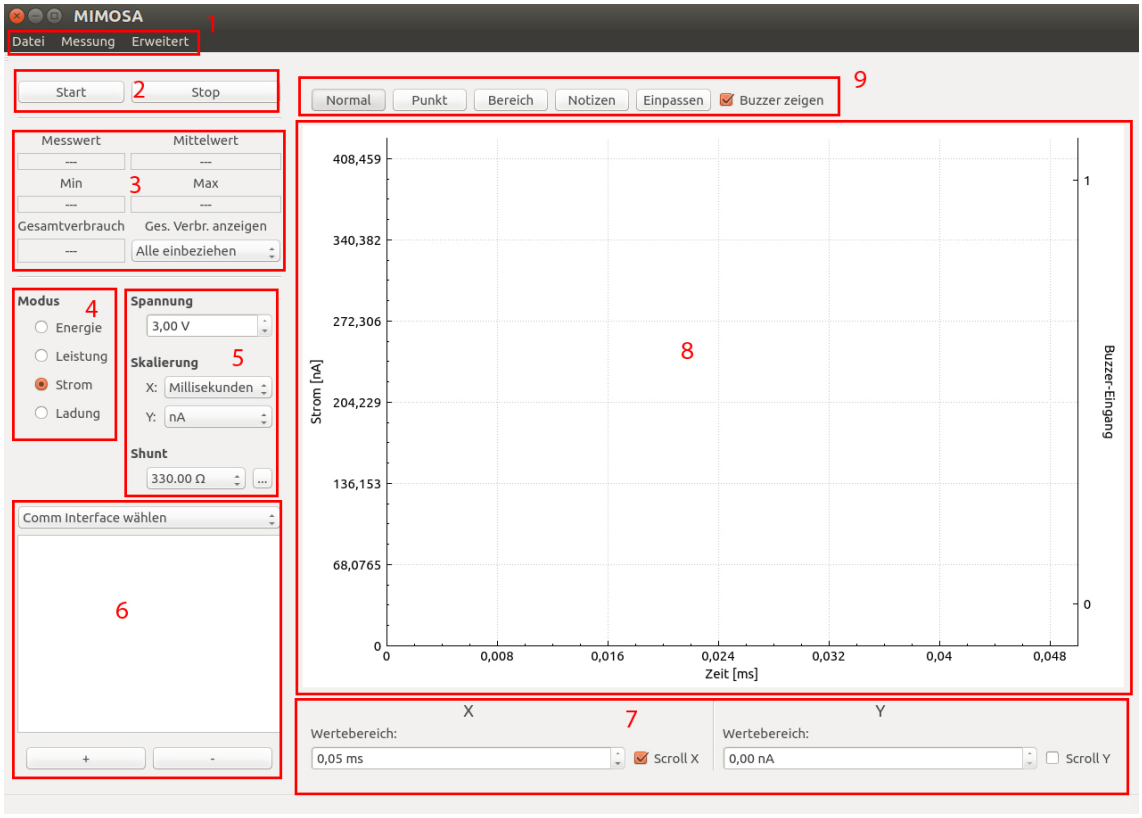


Abbildung 3.7.: Screenshot der GUI mit markierten Bereichen

3.3.1.1. Auswerten von Experimenten

Das Auswerten von Experimenten erfolgt mit Hilfe der Maus und den Elementen in den Bereichen 7 und 9. Durch ein Scrollen mit dem Mausrad kann dabei der Wertebereich der Achsen vergrößert oder verkleinert werden. Welche Achsen durch das Scrollen beeinflusst werden lässt sich über die Schaltflächen *Scroll X* und *Scroll Y* festlegen. Die Buttons in Bereich 9 legen fest, was bei einem Mausklick innerhalb des Graphen passiert. Im Modus *Normal* kann durch gedrückt halten der linken Maustaste der angezeigte Bereich verschoben werden. Der Modus *Punkt* wird anhand der X-Koordinate des angeklickten Punktes der Messwert ermittelt und in Bereich 3 angezeigt. Wird *Bereich* ausgewählt, so lassen sich durch klicken und ziehen Bereiche auf der X-Achse markieren. Dazu wird beim Drücken und Loslassen der linken Maustaste die X-Koordinate des jeweiligen Punktes gespeichert. Anschließend werden die statistischen Größen aus Bereich 3 für den gewählten Bereich berechnet und der Bereich wird grau hinterlegt.
Durch einen Klick auf *Notizen* kann der Benutzer Anmerkungen im Graphen erstellen. Dazu erscheint nach Klick in den Graphen ein Dialog zur Eingabe eines Textes. Die Notizen werden anschließend über *MimosaFileManager* gespeichert.

3.3.1.2. Skalierung der Achsen

Bei den zum Graph hinzugefügten Y-Werten handelt es sich um die A/D-Wandler Werte, die aus den Datenworten extrahiert wurden. Diese können mit Hilfe der Formeln aus 1.1.3 in physikalische Größen umgerechnet werden, welche im Bereich 4 (*Modus*) gewählt werden können. Würden die Messwerte direkt in physikalische Größen umgerechnet, so müssten alle Werte des Graphen beim Ändern des Modus neu berechnet werden, was je nach Anzahl der Messwerte sehr lange dauern kann. Um dies zu vermeiden werden statt dessen die A/D Werte gespeichert und die Skalierung der Y-Achse wird angepasst. Dies ist möglich, da die Umrechnung von A/D Werten in reale Größen nur der Multiplikation mit einem Faktor entspricht. Zu diesem Zweck wurde die Plot-Bibliothek (QCustomPlot) wie in Anhang A.3 beschrieben erweitert.
Zur Verdeutlichung ein kurzes Beispiel: Wird von MIMOSA ein Wort mit dem Datenwert 46962 empfangen, so wird dieser Wert in den Graphen eingetragen. Im Graph befindet sich nun der Punkt $(x, y) = (0, 46962)$. Auf Grund der in Bereich 4 und 5 eingestellten Parameter ergibt sich beispielsweise ein Faktor von $8,518e^{-5}$ für die Y-Achse. Der Benutzer sieht letztendlich auf der Y-Achse also den Wert 4 (mA) und nicht 46962. Wird eine andere Größe gewählt, so muss lediglich der Faktor für die Y-Achse neu berechnet werden, aber nicht alle Datenwerte.

3.3.1.3. Nachladen von Werten

Bei Experimenten, die länger als einige Sekunden dauern, würde es auf Grund der großen Anzahl von Messdaten lange dauern alle Daten zu laden und anzuzeigen. Aus diesem Grund wird immer nur ein Teil der Daten gleichzeitig geladen. Abbildung 3.8 zeigt das dabei angewandte Prinzip. Ausgehend vom Mittelpunkt des Sichtfensters (grüner Bereich) wird ein Bereich mit einer festen Größe geladen (grauer Bereich). Innerhalb des geladenen Bereichs kann die Ansicht nun frei verschoben, vergrößert und verkleinert werden. Erst wenn die Grenze des geladenen Bereichs erreicht wird (rote Linie), wird wieder ein neuer Bereich um den aktuellen Mittelpunkt geladen.

3.3.1.4. Kalibrierung

Über den Menüpunkt *Erweitert->Kalibrieren* lässt sich ein Fenster zum Kalibrieren von MIMOSA öffnen (siehe Abbildung 4.1). Zur Kalibrierung werden hier, ähnlich wie im Hauptfenster der GUI, durchgehend Werte von MIMOSA gelesen. Diese werden anschließend anhand der im Datenwort enthaltenen Integrierernummer sortiert in einem von drei Bereichen im Plot angezeigt. Dabei wird die Skalierung der Y-Achse (Strom) automatisch angepasst, so dass immer alle drei Integratorwerte zu sehen sind. Diese Funktion kann allerdings auch über das Feld *Y-Achse automatisch skalieren* deaktiviert werden.
Weitere für die Kalibrierung wichtige Funktionen sind das Bestimmen des Leerlaufoffsets,

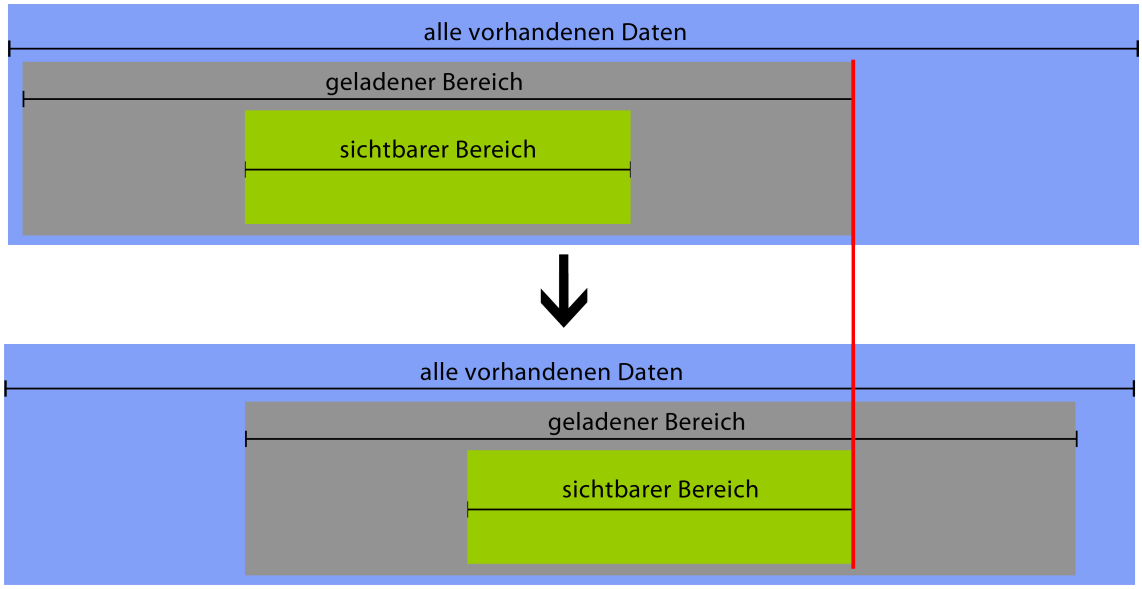


Abbildung 3.8.: Nachladen von Bereichen

sowie das Darstellen eines Sollwertes. Zur Ermittlung des Offsets wird über den Button *Offset bestimmen* ein Timer gestartet, der zwei Sekunden lang Werte von MIMOSA liest und anschließend den Mittelwert dieser Werte bildet.
Wird über das Feld *Soll* ein Sollwert eingestellt, so wird an der entsprechenden Stelle im Graph eine graue, horizontale Linie eingezeichnet.
In Abschnitt 4.2.1 wird der gesamte Kalibrierungsvorgang nochmals genauer beschrieben.

3.3.1.5. Comm-Interface konfigurieren

Über den Menüpunkt *Erweitert->Comm-Konfiguration*, oder durch die Auswahl eines Comm-Moduls im Bereich 6, lässt sich ein Dialog zur Konfiguration der Kommunikationsinterfaces öffnen (Abbildung 3.9)
Im oberen Bereich des Dialogs kann das zu konfigurierende Comm-Interface gewählt werden. Die verfügbaren Module werden hierzu mit Hilfe der Klasse *MimosaCommModuleManager* abgefragt. Der untere Bereich des Dialogs wird dynamisch an das gewählte Modul angepasst. Hierzu wird die XML Konfigurationsdatei des Moduls geladen.

```
<string default="/dev/ttyACM1" name="serialPort" text="Port"/>
<int default="8" name="databits" text="Data Bits" min="5" max="8"/>
```

Listing 3.3: Ausschnitt der Datei rs232.xml

Listing 3.3 zeigt einen Ausschnitt der Konfigurationsdatei für ein RS-232 Modul. Die komplette Datei ist in Anhang A.5 zu finden Für jeden Parameter der Konfigurationsdatei wird ein entsprechendes GUI-Element zum Dialog hinzugefügt. So wird beispielsweise für

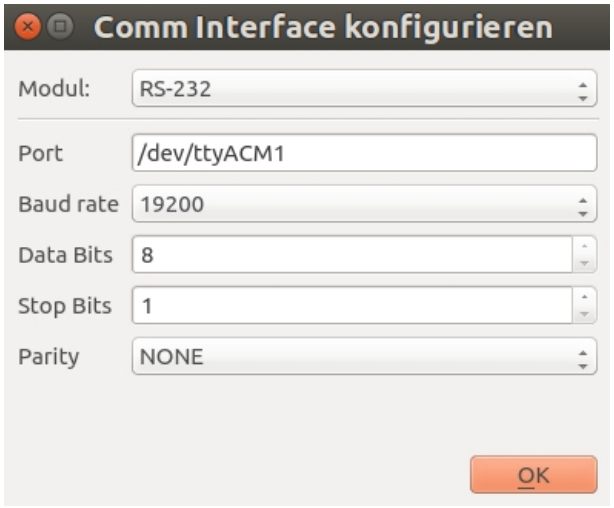


Abbildung 3.9.: Comm-Konfigurationsdialog

String-Parameter ein Textelement eingefügt. Enthält die Konfigurationsdatei einen Standardwert für einen Parameter (Schlüsselwort *default*), so wird dieser automatisch im Dialog eingetragen bzw. ausgewählt.

3.3.1.6. Export

Über den Menüpunkt *Datei->Export* kann der Benutzer die Messdaten exportieren. Dabei stehen die Möglichkeiten der Export als CSV-Datei oder als Grafik zur Verfügung. Der Export als CSV-Datei erfolgt über die Funktion *exportCSV()* der Klasse *MimosaFileManager* des Backends.
Für den Export der als Grafik wird die in die Plot-Bibliothek QCustomPlot integrierte Exportfunktionalität verwendet. Möglich ist der Export als Rastergrafik in den Formaten JPG, PNG, TIFF und BMP, sowie als Vektorgrafik im Format PDF[1].

3.3.2. CLI

Das Kommandozeilenfrontend wurde so entworfen, dass der Prozess nach dem Starten im Hintergrund läuft und auf eingehende Befehle wartet. Wird das Programm dazu mit dem entsprechenden Parameter aufgerufen (-s oder –start), so wird der Prozess mit Hilfe der Funktion daemon()[2] zu einem System-Daemon gemacht, der fortan im Hintergrund aktiv bleibt, bis der Befehl zum Beenden empfangen wird. Anschließend kann das Programm mit weiteren Parametern (siehe Anhang A.6) aufgerufen werden. Der Vorgang ist vereinfacht

[1] Der PDF Export funktioniert unter QCustomPlot 1.2.1 und Qt 5.3 nicht richtig. Die Grafik wird immer auf A4 Format skaliert, was je nach gewählter Auflösung und Größe zu starken Verzerrungen der Grafik führt. In der nächsten Version von QCustomPlot (1.3) wird dieser Fehler behoben sein [3].
[2] siehe *man 3 daemon*

in Abbildung 3.10 dargestellt. Der Daemon wartet durchgehend auf eingehende Befehle, die über einen lokalen Unix Domain Socket empfangen werden. Der Befehl wird daraufhin ausgeführt und der Prozess geht wieder in den Wartezustand über. Auf Seiten des Userprozesses wird zunächst überprüft, ob ein gültiger Parameter übergeben wurde. Wurde der Parameter für einen Befehl übergeben, für dessen Ausführung weitere Daten erforderlich sind, so werden diese zunächst gespeichert, so dass der Daemon sie anschließend laden kann. Ein Beispiel hierfür ist der Befehl *String senden* (-t, –send <Text>). In diesem Fall handelt es sich bei den für die Ausführung erforderlichen Daten um den Text, der gesendet werden soll. Über den lokalen Socket wird anschließend der Befehl an den Daemon gesendet und das Programm wird beendet.

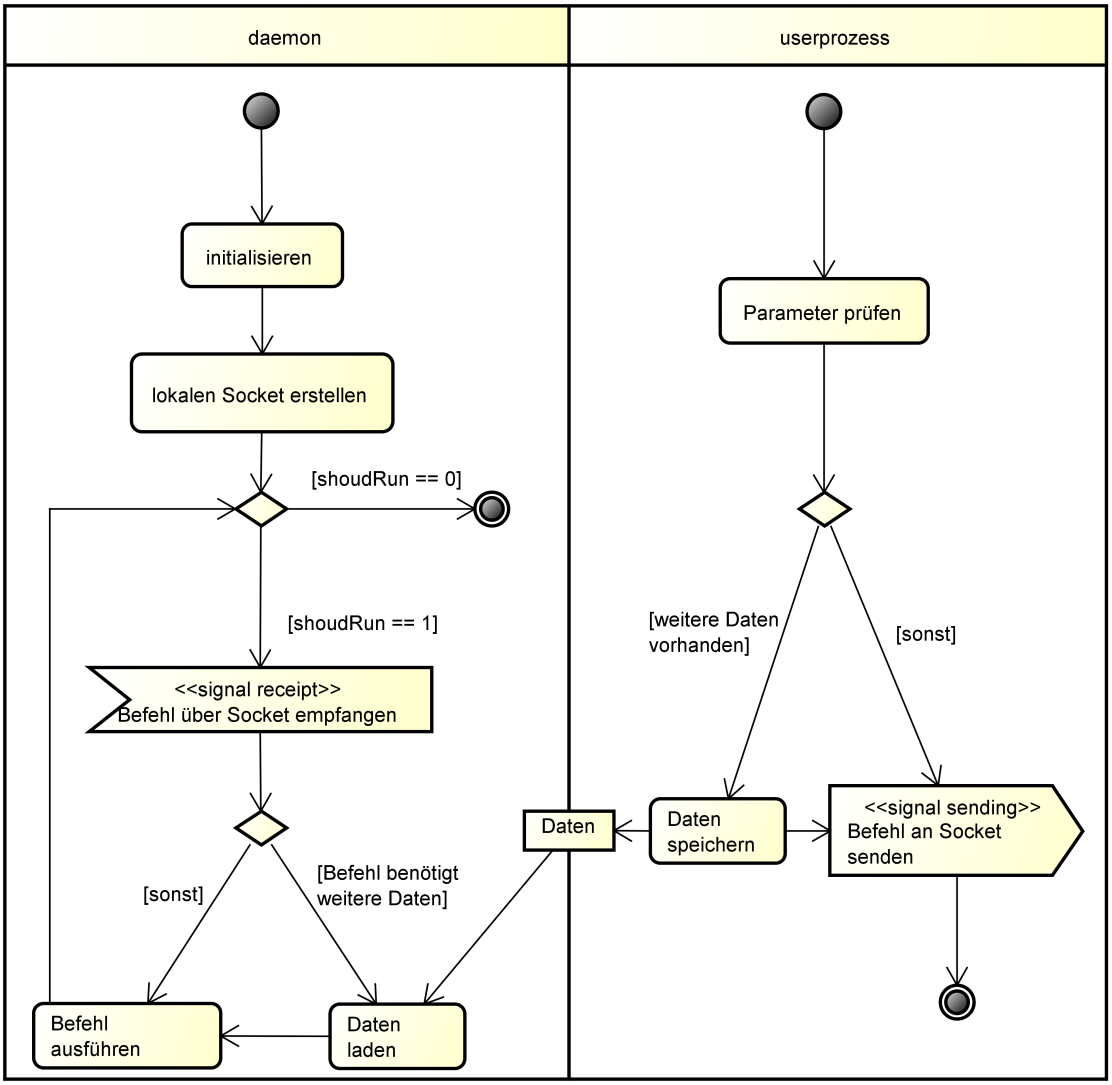


Abbildung 3.10.: Vereinfachtes Aktivitätsdiagramm des CLI

3.4. COMM-Module

Comm-Module dienen der Kommunikation mit dem DUT aus der Experimentierumgebung heraus. Sie stellen eine Schnittstelle zwischen der Experimentierumgebung und dem jeweiligen Hardwareinterface dar. Das in 3.2.6 beschriebene Design erlaubt dabei eine Anbindung beliebiger Schnittstellen an die Experimentierumgebung. Für diese Arbeit wurde ein RS-232 Modul für die serielle Schnittstelle implementiert, möglich sind aber auch andere Module, beispielsweise für USB-, Funk- oder Netzwerkgeräte.

3.4.1. Definition von Modulen

Kommunikationsinterfaces können verschiedene einstellbare Parameter besitzen. Ein Parameter einer seriellen RS-232 Schnittstelle ist z.B. die Baudrate, welche die Übertragungsgeschwindigkeit angibt.
Die für ein Comm-Modul relevanten Parameter werden in einer XML-Datei definiert. Der Dateiname dieser Konfigurationsdatei wird in der Variablen *configFileName* des Comm-Moduls gespeichert. Unterstützt werden Parameter vom Typ Integer, Double, String, sowie Aufzählungen (Enums) dieser Typen. Eine Beschreibung der verschiedenen Typen und des Formats von Konfigurationsdateien ist in Anhang A.4 zu finden. Jeder Parameter hat dabei einen eindeutigen Namen, anhand dessen der Programmierer bei der Implementierung des Moduls entscheiden kann, wie genau der Parameter zu setzen ist.

3.4.2. Implementierung von Modulen

Um ein Comm-Modul zu implementieren muss die Klasse *CommModule* (siehe Abbildung 3.6) abgeleitet werden. Dabei ist es zwingend erforderlich, dass die rein virtuellen Funktionen (kursiv im Klassendiagramm) überschrieben werden. Dies sind die Funktionen, in denen die Funktionalität für das zum Comm-Modul gehörende Hardwareinterface individuell implementiert wird. So muss beispielsweise in der Funktion *setParameter()* je nach übergebenem Parameternamen eine andere Funktion ausgeführt werden. Der Codeausschnitt in Listing 3.4 zeigt dies exemplarisch für den Parameter *baudrate*.

```
1 if(parameterName == "baudrate"){
2         int val = *((int*)value);
3         setBaudRate(val);
4         return true;
5 }
```

Listing 3.4: Ausschnitt der Funktion setParameter()

Nachdem ein Modul fertig implementiert wurde muss dieses noch in die Comm-API eingebunden werden, damit es dem Frontend zur Verfügung steht. Dies geschieht in der Klasse *MimosaCommModuleManager*, die einen Vektor mit allen verfügbaren Modulen verwaltet.

Im Konstruktor der Klasse werden alle verfügbaren Module instanziiert und dem Vektor hinzugefügt. Der folgende Codeausschnitt zeigt dies für das RS-232 Modul.

```
1  modules->push_back(new RS232CommModule(idCounter++));
```

Listing 3.5: Hinzufügen eines Comm-Moduls

3.5. Dateiformat

Bei der Durchführung von Experimenten werden von der Experimentierumgebung verschiedene Dateien angelegt, die die Reproduktion des Experiments ermöglichen. Zum einen sind dies die in 3.2.2 beschriebenen Skalierungsdateien, die die eigentlichen Messdaten enthalten. Zusätzlich gibt es noch je eine Datei für die vom Benutzer hinzugefügten Notizen, den Ablaufplan des Experiments, sowie die Konfiguration, also die gewählte Betriebsspannung des DUT, der Wert des verwendeten Shunt Widerstands und die Kalibrierungsparameter. Um die Übersichtlichkeit zu erhöhen werden all diese Dateien beim Speichern mit *MimosaFileManager* in einzelnes **.mim*-Archiv verpackt. Hierzu wird die freie Blibliothek *libarchive*[3] verwendet, welche außerdem die verlustfreie Kompression der archivierten Daten unterstützt. Dadurch lässt sich die Dateigröße auf etwa ein Fünftel[4] der unkomprimierten Daten reduzieren.

[3] Version 3.1.2, http://www.libarchive.org/, Stand: 13.11.2014
[4] Ermittelt anhand der Ergebnisse der Experimente aus Kapitel 4

4. Validierungsexperiment

Im letzten Kapitel wurde die Implementierung der Experimentierumgebung beschrieben. Dieses Kapitel beschäftigt sich nun mit der Validierung der Software. Ziel der Validierung ist es zum einen zu zeigen, dass das in Abschnitt 1.3 aufgestellte Ziel der Arbeit erreicht wurde und die neue Software tatsächlich die Durchführung von Experimenten erlaubt und dies eine Erleichterung im Vergleich zur Ausgangssituation darstellt. Des Weiteren soll gezeigt werden, dass die gelieferten Werte korrekt sind. Um dies zu erreichen wurden verschiedene Experimente mit einem TI LaunchPad durchgeführt.
Im ersten Abschnitt dieses Kapitels wird dabei zunächst das DUT beschrieben. Anschließend widmet sich Abschnitt 4.2 der Beschreibung der eigentlichen Experimente und der erwarteten Ergebnisse. Die Präsentation der erzielten Ergebnisse erfolgt in Abschnitt 4.3. Den Abschluss dieses Kapitels stellt das Fazit in Abschnitt 4.4 dar.

4.1. Das TI LaunchPad

Die Experimente zur Validerung der Software wurden mit einem MSP-EXP430FR5969 LauchPad (Rev. 2.0) von Texas Instruments [1] durchgeführt. Dabei handelt es sich um eine Entwicklerplatine auf Basis des MSP430FR5969 Mikrocontrollers, die auf einen besonders geringen Energieverbrauch („Ultra-Low Power“) ausgelegt wurde und sich daher besonders gut für Experimente mit MIMOSA eignet. Des Weiteren verfügt das LaunchPad über eine integrierte serielle Schnittstelle und lässt sich daher über das RS-232 Modul der Experimentierumgebung ansteuern. Die Energieversorgung der Platine erfolgt wahlweise über einen Micro-USB Anschluss, oder über eine externe Stromquelle. Diese Funktionalität ist wichtig, da es zur Messung mit MIMOSA zwingend erforderlich ist, dass das DUT über MIMOSA mit Strom versorgt wird.
Entwickler haben außerdem die Möglichkeit, die Funktionalität des LaunchPads mit sogenannten „BoosterPacks“ zu erweitern. BoosterPacks sind Hardwaremodule, die über eine 20 bzw. 40 Pin Verbindung auf die Platine gesteckt werden. Für die Experimente dieser Arbeit wurde ein „Sharp ® Memory LCD BoosterPack“ [6] verwendet, welches über ein Display mit einer Auflösung von 96x96 Pixeln verfügt und sich zeilenweise ansteuern lässt. Für größtmögliche Flexibilität besitzt das LaunchPad weiterhin eine Vielzahl von Messpunkten, Jumpern und anderen Komponenten. Für die durchzuführenden Experimente relevant sind dabei die Jumper J6, J9 und J13, sowie die LEDs LED1 und LED2.

Der Jumper J6 kann dazu verwendet werden, die Leuchtdiode LED1 von der Stromversorgung zu trennen, wodurch es möglich wird, das Signal der LED abzugreifen und als Buzzersignal für MIMOSA zu verwenden.
J9 dient der Strommessung der Platine. Wird dieser Jumper entfernt und durch ein Amperemeter ersetzt, so kann der gesamte Stromverbrauch des LaunchPads gemessen werden. Dies ist hilfreich, um die von MIMOSA gelieferten Werte zu verifizieren.
Um Daten über die serielle Schnittstelle an den Mikrocontroller zu übertragen ist es erforderlich, dass die Platine über USB mit dem PC verbunden ist. Dies führt allerdings dazu, dass die Platine auch über den USB Port mit Strom versorgt wird. Um dies zu verhindern kann über die Jumper in Block J13 festgelegt werden, welche Signalleitungen mit dem MSP430 Mikrocontroller verbunden sind. Um Daten vom PC zu empfangen sind hier nur die Signale RxD und GND erforderlich. Alle anderen Jumper dieses Blocks sollten entfernt werden um die Strommessung nicht zu verfälschen.
Die Leuchtdioden LED1 und LED2 können getrennt voneinander aktiviert und deaktiviert werden. Wie genau sie verwendet werden wird im nächsten Abschnitt beschrieben.

4.2. Die Experimente

Zur Validierung der Experimentierumgebung wurden drei Experimente durchgeführt. Das erste Experiment dient dabei der Verifikation der Messwerte, während die beiden letzten Experimente reale Anwendungsfälle darstellen und sowohl die Plausibilität der Messwerte, als auch das Erreichen des Ziels dieser Arbeit zeigen. Um jedoch genaue Messungen durchführen zu können, wurde MIMOSA zunächst mit der entwickelten Kalibrierungs-GUI kalibriert.

4.2.1. Kalibrierung

MIMOSA verfügt über drei Integratoren, die nacheinander die Messwerte liefern. Auf Grund von Bauteiltoleranzen kann es hierbei zur Abweichungen der Messwerte untereinander kommen. Um dies zu korrigieren verfügt jeder der Integratoren über einen variablen Widerstand, über den sich der Strom, und damit der Messwert, verändern lässt.
Die Kalibrierung wird in zwei Schritten ausgeführt: Schritt 1 ist die Bestimmung des Offsets. Wird MIMOSA ohne Last betrieben, so sollte der A/D-Wandler im Idealfall einen Wert von 0 liefern. In der Realität liegt dieser Wert allerdings deutlich höher. Um das Offset zu bestimmen muss der Button „Offset bestimmen“ (1) in der Kalibrierungs-GUI (Abbildung 4.1) gedrückt werden.
Anschließend erfolgt die Kalibrierung der Integratoren. Dazu wird ein Verbraucher angeschlossen. Um die Kalibrierung zu vereinfachen sollte hier ein Verbraucher mit einem konstanten Stromverbrauch verwendet werden. Für dieses Experiment wurde ein ohmscher Widerstand mit 600 Ω verwendet. Bei einer Betriebsspannung von 3V ergibt sich so ein

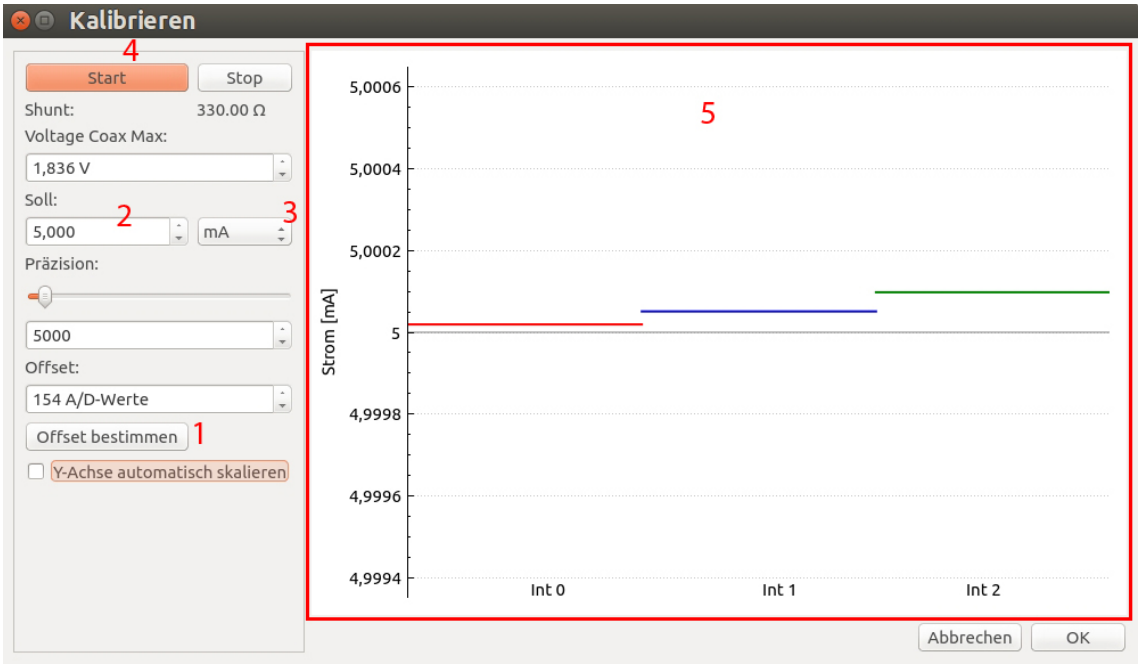


Abbildung 4.1.: Werte nach der Kalibrierung

Sollwert von 5 mA. Dieser Wert wird über die Schaltflächen (2) und (3) eingestellt. Danach wird der Button „Start“ (4) gedrückt. Im Bereich (5) werden nun die Werte der einzelnen Integratoren als horizontale Linie angezeigt. Auf der MIMOSA Platine werden nun die Widerstände der Integratoren nacheinander so verändert, bis die tatsächlichen Werte dem Sollwert entsprechen. Abbildung 4.1 zeigt das Ergebnis der Kalibrierung. Ein perfektes Ergebnis, bei dem alle drei Werte exakt dem Sollwert entsprechen, ist nicht erreichbar, da es durch das Rauschen immer zu Abweichungen kommt. Die hier erreichte Genauigkeit liegt im Bereich von unter 0.1 μA und ist damit ausreichend.

4.2.2. Versuchsaufbau

Abbildung 4.2 zeigt den verwendeten Versuchsaufbau, der typisch für Experimente mit MIMOSA ist. Das DUT wird über MIMOSA mit Strom versorgt. MIMOSA ist dazu selbst mit einer Gleichspannungsquelle verbunden. Über eine USB Verbindung werden die Messdaten von MIMOSA zum PC, auf dem die Experimentierumgebung läuft, übertragen. Der Rückkanal von der Experimentierumgebung zum DUT wird hier über eine serielle RS-232 Verbindung realisiert. Weiterhin ist ein Ausgang des DUT mit dem Buzzereingang von MIMOSA verbunden, wodurch für die Energiemessung relevante Aktionen signalisiert werden können. Für folgenden Experimente wurde als Ausgang das Signal von LED1 gewählt, da sich dieses leicht über den Jumper J6 am LaunchPad abgreifen lässt.

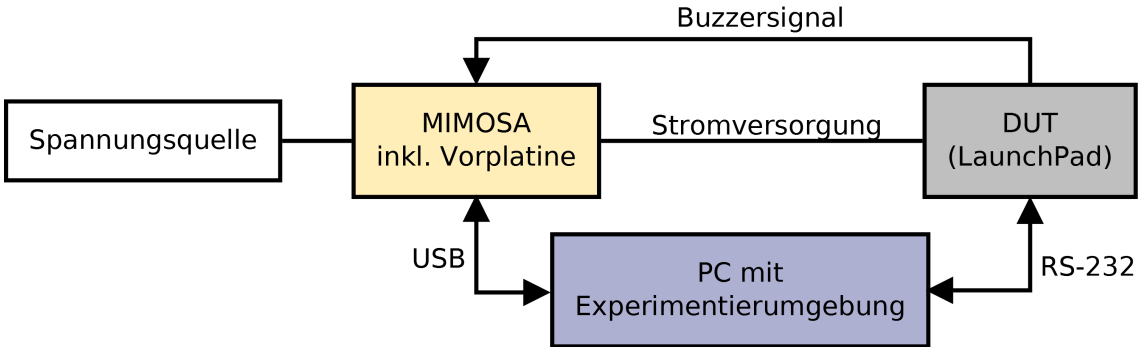


Abbildung 4.2.: Typischer Versuchsaufbau eines Experiments

4.2.3. Verifikation der Messwerte

Der erste Versuch dient der Verifikation der dargestellten Messwerte. Dazu wurde über den Jumper J9 ein digitales Multimeter zur Strommessung an das LaunchPad angeschlossen. Die gemessenen Werte wurden anschließend mit den von der Experimentierumgebung gelieferten Messwerten von MIMOSA verglichen. Da das Strommessgerät nicht die gleiche hohe zeitliche Auflösung besitzt wie MIMOSA, und daher nur Mittelwerte liefert, wurde das LaunchPad so programmiert, dass es einen möglichst konstanten Strombedarf hat. Hierzu wurde eine Firmware aufgespielt, die nach dem Einschalten des LaunchPads LED2 der Platine aktiviert und danach keine Aktionen mehr durchführt. Arbeiten sowohl MIMOSA als auch die Experimentierumgebung korrekt, so sollten die Werte der Software und des Messgeräts übereinstimmen.

4.2.4. Verifikation der Software

Zur Validierung der Software wurden zwei Experimente durchgeführt, die einfache, reale Anwendungsfälle darstellen. Hierzu wurde das LaunchPad mit einem Programm bespielt, das über die serielle Schnittstelle Zeichenketten (Strings) von der Experimentierumgebung empfängt. Diese werden in einem 64 Byte großen Puffer gespeichert, so dass ein String maximal 64 Zeichen lang sein kann. Ob nach dem Empfangen des Strings eine Aktion ausgeführt wird, und welche dies ist, hängt davon ab, ob der String einem von zwei Mustern entspricht.

Das erste Experiment simuliert die Übertragung von Zeichen, wie sie beispielsweise von einer Infrarot Fernbedienung durchgeführt wird. Das Programm wartet dabei auf Strings mit dem Muster *L[!-zA-Z]{1,62}@*. Das bedeutet, dass auf ein „L“ maximal 62 weitere Zeichen folgen können, gefolgt von einem „@“, welches das Ende des Strings anzeigt. Wurde eine gültige Zeichenkette empfangen, so wird LED1 aktiviert. Das Signal von LED1, das über J6 abgegriffen werden kann, wird hierbei als Eingangssignal für den Buzzereingang von MIMOSA verwendet. So kann später der relevante Bereich aus den Messdaten gefiltert werden. Anschließend wird die binäre Repräsentation der empfangenen Zeichen

Anzahl Einsen	String	Binär
4	AA	0100000101000001
5	AC	0100000101000011
6	CC	0100001101000011
7	CG	0100001101000111
8	GG	0100011101000111
9	GO	0100011101001111
10	OO	0100111101001111
11	O_	0100111101011111
12	__	0101111101011111

Tabelle 4.1.: Verwendete Zeichen und deren binäre Codierung

(siehe Tabelle 4.1) bitweise über die Leuchtdiode LED2 ausgegeben, wobei das LSB zuerst ausgegeben wird. Dazu wird jedes Bit des Zeichen-Bytes nacheinander einzeln betrachtet. Handelt es sich um eine 1, so wird LED2 aktiviert, handelt es sich um eine 0, so wird die Leuchtdiode deaktiviert. Anschließend wartet das Programm eine Millisekunde. Dies ist erforderlich, da die „Übertragungszeit“ eines Bits sonst zu kurz wäre um noch zuverlässig von MIMOSA erfasst werden zu können. Nach Abschluss der Ausgabe des gesamten Strings wird LED1 wieder deaktiviert und somit das Ende des relevanten Bereichs markiert.
Ziel ist es, ein Experiment durchzuführen, mit dessen Hilfe der Zusammenhang zwischen dem Energiebedarf des LaunchPads und der Anzahl der codierten Einsen des übertragenen Strings hergestellt werden kann. Je mehr Einsen ein Zeichen enthält, desto öfter wird die LED aktiviert und desto größer ist der zu erwartende Strombedarf. Zu erwarten ist hier ein linear steigender Zusammenhang. Tabelle 4.1 zeigt die bei dem Experiment verwendeten Zeichenketten und deren binäre Codierung. Es wurden in allen Fällen Strings der Länge zwei gewählt, damit die Ausgabedauer immer gleich lang ist somit bei der Erstellung des Energiemodells nicht zu berücksichtigt werden braucht.

Das zweite Experiment ist dem ersten sehr ähnlich, wobei diesmal der Energiebedarf bei der Ansteuerung des Displays untersucht wird. Dazu werden Strings der Form *D[0-96]@* erwartet. Wurde eine solche Zeichenkette empfangen, so wird das Display zunächst initialisiert und das alte Bild wird gelöscht. Anschließend wird das Buzzersignal über LED1 aktiviert, das neue Bild wird dargestellt und das Buzzersignal wird wieder deaktiviert. Die Zahl zwischen 0 und 96 gibt dabei an, wie viele Zeilen des Bildes dargestellt werden sollen. Hierbei ist zu erwarten, dass sowohl die für die Darstellung nötige Zeit, als auch der Energiebedarf der Darstellung, linear mit der Anzahl der Zeilen steigen.

4.3. Ergebnisse

4.3.1. Verifikation der Messwerte

Die Korrektheit der Messwerte wurde wie in 4.2.3 beschrieben dadurch überprüft, dass der konstante Stromfluss beim Betreiben einer einzelnen LED auf dem LaunchPad gemessen wurde. Abbildung 4.3 zeigt das Ergebnis dieser Messung. Im Mittel liegt der mit MIMOSA gemessene Strom bei 1,99 mA. Der mit dem digitalen Multimeter ermittelte Wert beträgt für das gleiche Szenario 1,96 mA. Bei dem verwendeten Multimeter handelt es sich um ein „V&A Intrument VA19“, das bei Strommessungen im Bereich bis 40 mA eine Genauigkeit von ±1,5% hat (siehe Bedienungsanleitung auf der beigelegten CD). Die ermittelten Werte liegen somit innerhalb des Toleranzbereichs und können als korrekt angesehen werden.
Eine weitere Bestätigung der Korrektheit der Messwerte liefern die Ergebnisse der Experimente mit dem Display und der LED. Abbildung 4.5 zeigt einen Ausschnitt des Displayexperiments. Gut zu erkennen sind dabei die Stromspitzen (Peaks), die dadurch entstehen, dass der Mikrocontroller vom Scheduler alle 10ms in den sogenannten „active mode“ versetzt wird. Laut Datenblatt liegt der Strombedarf in diesem Modus bei etwa 100μA/MHz bei 3V Betriebsspannung [2]. Da der Mikrocontroller bei diesem Experiment mit 16 MHz betrieben wurde, sollte daher ein Strom von ca. 1600μA fließen. Auch dies konnte mit der Experimentierumgebung bestätigt werden. Die Peaks in Abbildung 4.5 liegen bei etwa 1640μA. Da es sich bei der Angabe von 100μA/MHz im Datenblatt auch nur um einen ungefähren Wert handelt, ist der Wert der Experimentierumgebung plausibel.

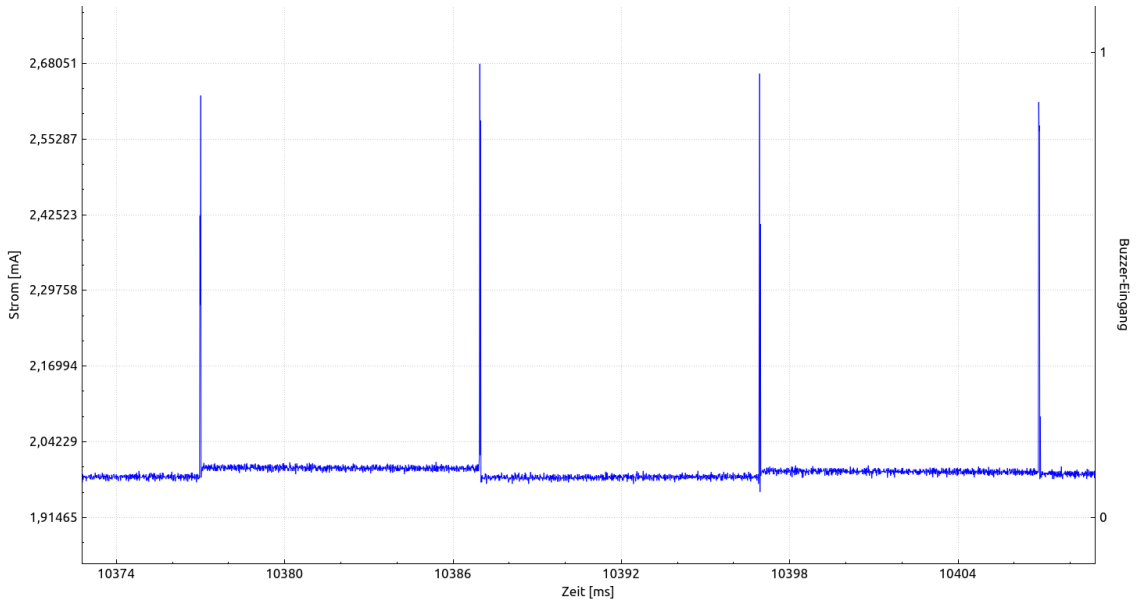


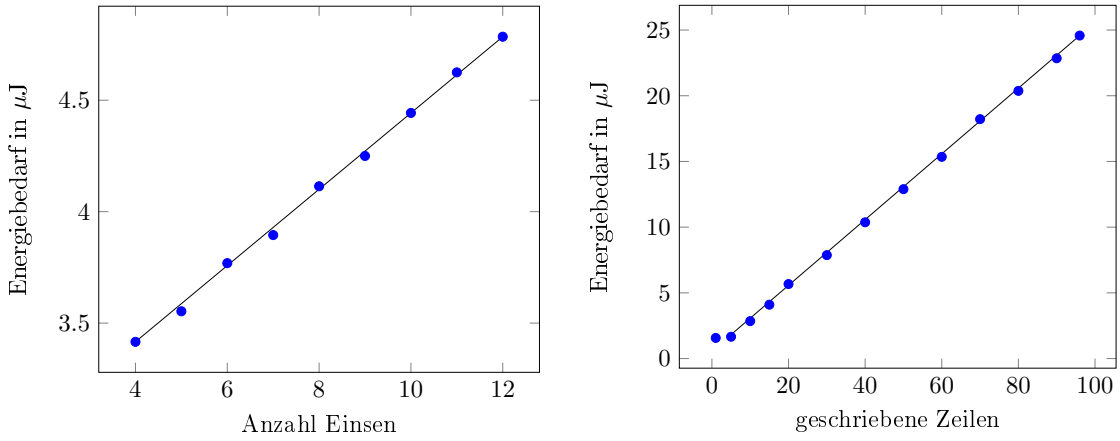
Abbildung 4.3.: Ergebnis der Messung des Strombedarfs von LED2 mit MIMOSA

4.3.2. Verifikation der Software

Vor der Durchführung der Experimente wurde für das Display- und das LED-Experiment jeweils ein Zeitplan innerhalb der GUI erstellt und abgespeichert. Diese befinden sich, ebenso wie die Ergebnisse der Experimente, auf der beiliegenden CD (Anhang A.8). Die Möglichkeit, die Zeitpläne separat zu erstellen und auch ohne durchgeführte Messung abzuspeichern ist eine besonders nützliche Funktion der Experimentierumgebung, denn dadurch ist die Reproduktion der Experimente sehr einfach.
Die eigentliche Durchführung eines Experiments beschränkt sich danach für den Benutzer auf einen einzelnen Klick auf den Start-Button der GUI um das Experiment zu starten, sowie einen weiteren Klick auf den Stop-Button nachdem der Zeitplan abgearbeitet wurde. Dies stellt bereits eine erhebliche Verbesserung gegenüber der alten Software dar. Der alte Messvorgang hätte es erfordert, zunächst ein separates Programm oder Skript zu schreiben, dass die Ansteuerung der seriellen Schnittstelle und das Abarbeiten eines Zeitplans erlaubt. Anschließend hätten dieses Programm und die MIMOSA-Software getrennt voneinander gestartet werden müssen.
Die Auswertung des Experiments stellt eine weitere Verbesserung dar. Beim alten Messablauf hätte hierzu wiederum ein separates Programm verwendet werden müssen. Mit der neuen Experimentierumgebung konnte die Auswertung direkt in der GUI erfolgen, da diese über alle nötigen Funktionen verfügt. Als besonders hilfreich erwies sich auch die Möglichkeit, die Messwerte anhand des Buzzersignals zu filtern, sowie die automatische Berechnung der Energieaufnahme in den relevanten Bereichen. Diese Funktion ermöglichte es die Gesamtenergie der Bereiche an den gewünschten Stellen schnell abzulesen. Daraus konnten anschließend sehr einfach Energiemodelle für das Display und die Übertragung von Zeichen mittels der LED aufgestellt werden, welche in Abbildung 4.4 zu sehen sind. In beiden Fällen ist ein annähernd linearer Zusammenhang zwischen der variablen Größe und dem Energiebedarf zu erkennen, was den Erwartungen entspricht.

4.4. Fazit

Durch die durchgeführten Experimente konnten beide Ziele der Validierung erreicht werden. Zum einen sind die von MIMOSA gelieferten und von der Experimentierumgebung verarbeiteten Messwerte korrekt. Zum anderen können mit der entwickelten Software Experimente auf eine Weise durchgeführt und ausgewertet werden, die eine erhebliche Erleichterung zur Ausgangssituation darstellt. Mit der alten Software wäre der Ablauf deutlich komplizierter gewesen, da neben dem MIMOSA Programm mindestens zwei weitere Programme für die Ansteuerung des DUT und die Auswertung der Daten nötig gewesen wären. Zur Auswertung der Daten hätte weiterhin zuerst ein Programm oder Skript geschrieben werden müssen, das die relevanten Daten aus der CSV Datei filtert und in brauchbare Ener-



(a) Energiemodell LED

(b) Energiemodell Display

Abbildung 4.4.: Energiemodelle für LED und Display

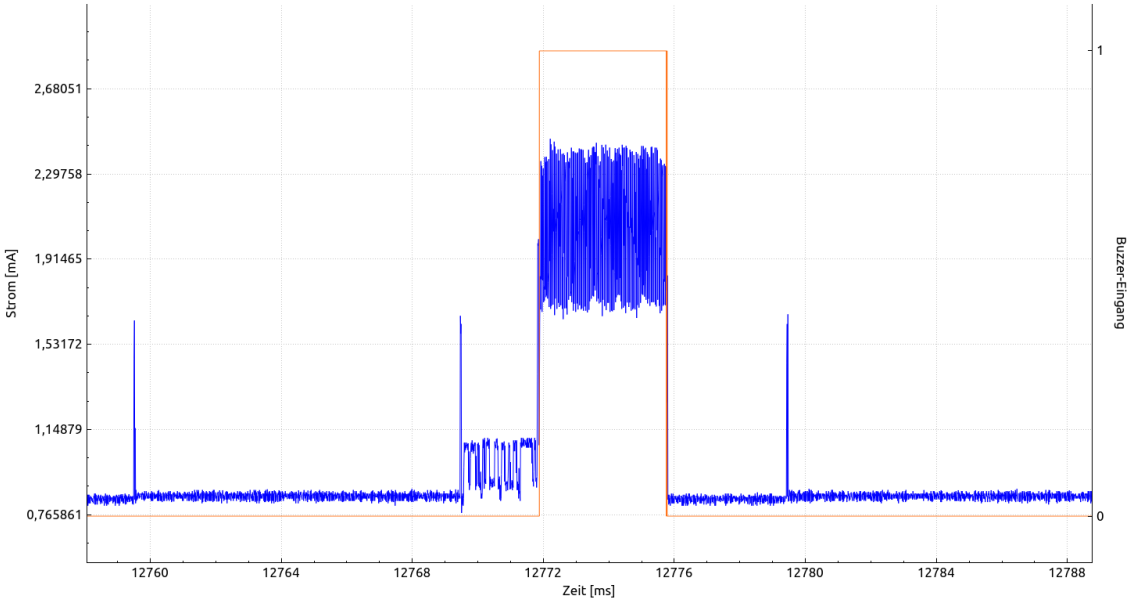


Abbildung 4.5.: Ausschnitt des Displayexperiments

giewerte umrechnet. Bei der Planung von Experimenten ist die Möglichkeit des Erstellens und Abspeicherns von Zeitplänen eine große Erleichterung. Selbst wenn die Reproduzierbarkeit der Experiment nicht erforderlich ist, so ist diese Funktion dennoch hilfreich. Ist es beispielsweise auf Grund begrenzter Ressourcen erforderlich, dass mehrere Wissenschaftler mit einem Messgerät arbeiten, so können diese ihre Experimente bereits vorher am eigenen PC planen und den Zeitplan abspeichern. Diese Datei muss dann zur Messung nur noch geladen und ausgeführt werden, was die Zeit, in der MIMOSA für ein Experiment belegt ist, deutlich reduziert. Die Auswertung kann anschließend wieder unabhängig von der Messhardware geschehen. Hierbei sind besonders das einfache Umschalten zwischen

Größen wie Strom oder Energie, das Betrachten einzelner Messpunkte und -bereiche sowie das Verschieben des angezeigten Ausschnitts hilfreiche Funktionen der GUI, die das Auswerten von Experimenten bedeutend erleichtern.
Zusammenfassend lässt sich daher sagen, dass die Experimente mit der alten Software, unter Zuhilfenahme weiterer Programme, zwar möglich gewesen wären, der Aufwand jedoch deutlich höher und der Komfort deutlich niedriger gewesen wäre.

5. Zusammenfassung

Im Rahmen dieser Arbeit sollte eine Software zur Durchführung von Messungen und Experimenten mit MIMOSA entwickelt und implementiert werden.
Dazu wurde zunächst eine Betrachtung von MIMOSA durchgeführt. Die wichtigsten Eigenschaften des Messgerätes sind die Verwendung einer Schaltung zur Kompensation des Spannungsabfalls, sowie die Verwendung von analogen Integratoren. Dadurch ist es möglich, auch sehr kleine Ströme und sehr kurze Energiespitzen zu messen.
Anschließend wurde der Ablauf eines Messvorgangs mit MIMOSA analysiert und als langsam und unpraktisch eingestuft. Gründe hierfür waren das Fehlen einer Live-Darstellung, das verwendete CSV-Dateiformat, die Notwendigkeit von verschiedener Software für die Durchführung und die Auswertung der Messung, sowie die nicht vorhandene Reproduzierbarkeit der Messungen. Besonders der letzte Punkt stellte einen zentralen Aspekt bei der Aufstellung des Ziels der Arbeit dar. Die neue Experimentierumgebung sollte einen Rückkanal besitzen, mit dem es möglich ist, direkten Einfluss auf das DUT zu nehmen.
In einer Anforderungsanalyse wurde daraufhin genauer untersucht, welche Funktionalität die Experimentierumgebung benötigt. Es wurde entschieden, dass die Software aus mehreren miteinander interagierenden Komponenten bestehen soll, um die Wartbarkeit und Übersichtlichkeit zu verbessern. Dazu wurde anschließend ein Konzept erstellt, das im wesentlichen aus fünf Komponenten bestand: Ein Hintergrundprozess ist für die Kommunikation mit MIMOSA zuständig und gewährleistet, dass alle Messdaten gelesen und in einer FIFO zwischengespeichert werden. Der Zugriff darauf erfolgt über das Backend, das aus verschiedenen APIs besteht. Der Benutzer interagiert mit der Experimentierumgebung über das Frontend. Hier wurde eine grafische, sowie eine kommandozeilenbasierte Schnittstelle entworfen. Über sogenannte Comm-Module ist außerdem die Anbindung beliebiger Kommunikationsinterfaces an die Experimentierumgebung möglich.
Die Komponenten wurden anschließend implementiert. Abschließend wurde die Experimentierumgebung validiert. Dazu wurden verschiedene Experimente durchgeführt, deren Ziel es war, die Korrektheit der gelieferten Werte und das Erreichen des Ziels der Arbeit zu zeigen. Es wurde ein LaunchPad von Texas Instruments mit Hilfe von MIMOSA und der Experimentierumgebung vermessen. Die Durchführung der Experimente mit der neuen Software stellte eine erhebliche Verbesserung im Vergleich zum alten Messablauf dar und die gelieferten Ergebnisse entsprachen den Erwartungen. Beide Ziele der Validierung wurden somit erreicht.

A. Anhang

A.1. CD

Der Arbeit liegt eine CD bei, auf der sich die entwickelte Software als Quellcode und in Form von kompilierten Dateien befindet.
Außerdem befinden sich auf der CD verschiedene Datenblätter, die Ergebnisse der Validierung, sowie die Datei mimosa-berechnungen.ods (siehe A.2).

A.2. mimosa-berechnungen.ods

Auf der beiliegenden CD befindet sich die Datei *mimosa-berechnungen.ods*. Dieses Dokument wurde von Markus Buschhoff erstellt und zeigt die Zusammenhänge der Umrechnung des MIMOSA-Datenwerts in physikalische Größen.

A.3. Anpassung von QCustomPlot

Um die einfache Umrechnung von physikalischen Größen innerhalb der GUI zu ermöglichen wurde die Klasse *QCPAxis* der Bilbiothek *QCusomPlot* so angepasst, dass die dargestellten Werte sich aus dem gespeicherten Datenwert multipliziert mit einem konstanten Faktor ergeben.
Dazu wurde die Variable *axisLabelScaleFactor* als konstanter Faktor, sowie Funktionen zum Auslesen und setzen dieser Variablen, implementiert.

```
1 double mAxisLabelScaleFactor;
2 Q_PROPERTY(double axisLabelScaleFactor READ axisLabelScaleFactor)
3 double axisLabelScaleFactor() const { return mAxisLabelScaleFactor;
4 void setAxisLabelScaleFactor(double factor){mAxisLabelScaleFactor = factor;} }
```

Listing A.1: Ausschnitt der Datei qcustomplot.h

In der Datei *qcustomplot.cpp* erfolgt die Verwendung des neuen Faktors. Dazu wurde der Faktor in der Funktion *void QCPAxis::setupTickVectors()* eingefügt.

```
1 mTickVectorLabels[i] = mParentPlot->locale().toString(
2         mTickVector.at(i)* mAxisLabelScaleFactor,
3         mNumberFormatChar,
```

```
4          mNumberPrecision);
```

Listing A.2: Ausschnitt der Funktion QCPAxis::setupTickVectors()

A.4. Aufbau einer Konfigurationsdatei für Comm-Module

Eine XML-Konfigurationsdatei für Comm-Module beginnt mit der XML Deklaration *<?xml version='1.0'?>*. Darauf folgt ein einzelnes Element mit dem Namen *Device* und den Attributen *name* und *description*.
Innerhalb des Device-Elements gibt für jeden Parameter des Moduls ein weiteres Element vom Typ *string*, *int*, *double* oder *enum*. Jedes dieser Elemente muss die Attribute *name* und *text* haben. Das Attribut *name* dient zur Identifizierung des Parameters und muss daher eindeutig sein. *text* enthällt den Text, der später dem Benutzer bei der Eingabe des Parameters angezeigt wird.
Optional kann es das Attribut *default* geben, das den Standardwert für den Parameter angibt. Elemente vom Typ int oder double können außerdem ein *min* und *max* Attribut haben. Diese geben jeweils den minimal, bzw. maximal möglichen Wert des Parameters an.
enum-Elemente sind Aufzählungen der anderen Elementtypen und müssen daher ein Attribut namens *type* haben, das den Typ angibt.
Ein Beispiel für eine Konfigurationsdatei ist Anhang A.5.

A.5. Konfigurationsdatei des RS-232 Moduls

```
<?xml version='1.0'?>
<Device name="RS232" description="Serielle Schnittstelle">
 <string default="/dev/ttyACM1" name="serialPort" text="Port"/>
 <enum default="19200" name="baudrate" text="Baud rate" type="int">
  <int>50</int>
  <int>75</int>
  <int>110</int>
  <int>134</int>
  <int>150</int>
  <int>200</int>
  <int>300</int>
  <int>600</int>
  <int>1200</int>
  <int>1800</int>
  <int>2400</int>
```

```
  <int>4800</int>
  <int>9600</int>
  <int>19200</int>
  <int>38400</int>
  <int>57600</int>
  <int>115200</int>
  <int>230400</int>
 </enum>
 <int default="8" name="databits" text="Data Bits" min="5" max="8"/>
 <int default="1" name="stopbits" text="Stop Bits" min="1" max="2"/>
 <enum default="100" name="parity" text="Parity" type="int">
  <int description="NONE">100</int>
  <int description="EVEN">200</int>
  <int description="ODD">300</int>
  <int description="MARK">400</int>
  <int description="SPACE">500</int>
 </enum>
</Device>
```

Listing A.3: Die Datei rs232.xml

A.6. Parameter des CLI

-s, –start
Startet den Hintergrundprozess der die eigentliche Arbeit übernimmt. Diese Befehl muss zuerst ausgeführt werden

-k, –stop
Stoppt den Hintergrundprozess.

-m, –mimosa-start
Startet eine Messung mit MIMOSA.

-n, –mimosa-stop
Hällt eine laufende eine Messung mit MIMOSA an.

-t, –send <Text>
Sendet den String <Text> über das gewählte Comm-Interface

-f, –sendfile <Dateiname>

Sendet eine Datei über das gewählte Comm-Interface. Dabei muss der komplette absolute Pfad zur Datei angegeben werden

–p, –parameter <Parameter> <Wert>
Setzt den Parameter <Parameter> auf den Wert <Wert>

Liste aller Parameter:

Parameter	Typ	Beschreibung
shunt	Double	Der bei der Vorplatine verwendete Shunt Widerstand in Ohm (default: 330 Ohm)
voltage	Double	Die Betriebsspannung des DUT in Volt (default: 3V)
directory	String	Das Verzeichnis in das die Daten nach der Messung gespeichert werden (default: aktuelles Arbeitsverzeichnis)
vcoaxmax	Double	Maximaler Messbereich (default: 1.836V)
offset	Integer	Offset von MIMOSA ohne angeschlossenen Verbraucher (default: 198)

-i, –interface
Auswahl eines Comm-Interfaces aus einer Liste aller verfügbaren Interfaces.

-c, –interface-config
Ruft einen Konfigurationsdialog für das aktuell gewählte Comm-Interface auf.

-e, –export-csv
Exportiert alle vorhandenen Daten als CSV Datei.
Die Datei wird im aktuellen Verzeichnis gespeichert, sofern mit -p directory kein anderes Verzeichnis festgelegt wurde.

-g, –export-graphic <Format> <Y-Achse> <X-Achse> <Skalierung> <Breite> <Höhe>
Exportiert alle vorhandenen Daten als CSV Datei.
Die Datei wird im aktuellen Verzeichnis gespeichert, sofern mit -p directory kein anderes Verzeichnis festgelegt wurde.
<Format>: jpg, png, pdf
<Y-Achse>: Energie, Strom, Ladung, Leistung
<X-Achse>: Skala der X-Achse: ms, sek, min, h (Millisekunden, Sekunden, Minuten, Stunden)
<Y-Skalierung>: piko, nano, micro, milli
<Breite>: Die Breite der Grafik in Pixel
<Höhe>: Die Höhe der Grafik in Pixel

-h, –help
Aufruf der Hilfe

A.7. Systemvoraussetzungen / Installationsanweisungen

Die Experimentierumgebung setzt die Bibliotheken libftdi1, libftdi-dev, sowie libarchive voraus.
Die FTDI-Bibliotheken können unter Ubuntu beispielsweise mit dem Befehl **sudo apt-get install libftdi1 libftdi-dev** installiert werden.
Die Bibliothek libarchive befindet sich auf der beiliegenden CD oder kann unter `http://www.libarchive.org` heruntergeladen werden.
Die Installation erfolgt anschließend mit den Befehlen

tar xzf libarchive-3.1.2.tar.gz
cd libarchive-3.1.2
./configure
make
make check
make install

Weitere Informationen sind unter `https://github.com/libarchive/libarchive/wiki/BuildInstructions` zu finden.

Die Experimentierumgebung erfordert zur Ausführung normalerweise Rootrechte. Um die Ausführung als normaler Benutzer zu ermöglichen sind zwei Schritte zu unternehmen.

Zuerst muss die Datei
/etc/udev/rules.d/92-ftdi.rules
mit dem Inhalt
ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6001", MODE=="0666", GROUP="uucp"
erstellt werden. Der aktuelle Benutzer muss in der Gruppe *uucp* sein.
Um die eben erstellte Regel zu laden muss der Befehl **udevadm control –reload-rules** ausgeführt werden.
Des Weiteren müssen in der Datei
/etc/security/limits.conf
die Zeilen

<username> soft rtprio 100
<username> hard rtprio 100
ergänzt werden, wobei <username> durch den eigenen Benutzernamen ersetzt werden muss. Anschließend ist ein neues Login erforderlich.

Um die Programme des Frontends (MimosaCMD, MimosaGUI) zu verwenden müssen die LibMimosa Dateien (siehe CD) in den Ordner /**usr**/**lib**/ kopiert werden.

A.8. Ergebnisse der Validierung

Die Dateien mit den für die Validierung verwendeten Zeitplänen, sowie die Ergebnisse der Experimente, befinden sich im Ordner *Validierung* auf der beiliegenden CD.

Abkürzungsverzeichnis

CLI Command Line Interface

CSV Comma-seperated Values

DUT Device Under Test

FIFO First In - First Out

GUI Graphical User Interface

JPG Joint Photographic Experts Group

LSB Least Significant Bit

MIMOSA Messgerät zur integrativen Messung ohne Spannungsabfall

PNG Portable Network Graphics

Abbildungsverzeichnis

Literaturverzeichnis

[1] *MSP-EXP430FR5969 LaunchPad Development Kit User's Guide.* http://www.ti.com/lit/pdf/slau535. Version 23.06.2014.

[2] *MSP430FR59xx Mixed-Signal Microcontrollers Datenblatt.* http://www.ti.com/lit/gpn/msp430fr5969.

[3] *PDF Bug in QCustomPlot.* http://www.qcustomplot.com/index.php/support/forum/492.

[4] *QCustomPlot.* http://www.qcustomplot.com.

[5] *Qt Project.* http://www.qt-project.org/.

[6] *Sharp LCD BoosterPack Datenblatt.* http://www.ti.com/lit/pdf/slau553.

[7] *FT245BL USB FIFO (USB - Parallel) I.C. datasheet.* http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT245BL.pdf, 2005.

[8] Buschhoff, Markus, Christian Günter und Olaf Spinczyk: *MIMOSA, a Highly Sensitive and Accurate Power Measurement Technique for Low-Power Systems.* In: *Real-World Wireless Sensor Networks*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013.

[9] Corbet, Jonathan, Alessandro Rubini und Greg Kroah-Hartman: *Linux Device Drivers, 3rd Edition.* O'Reilly Media, Inc., 2005.

[10] Love, Robert: *Linux System Programming, 2nd Edition.* O'Reilly Media, 2 Auflage, 2013.

[11] Wolf, Juergen: *C++ von A bis Z.* Band 2. Galileo Press, 2009.