Fachhochschule Dortmund
University of Applied Sciences and Arts

Fachbereich Elektrotechnik
Studiengang Energiewirtschaft

Bachelorarbeit

Steuerung eines Keithley 2400 Sourcemeters über eine RS-232 Schnittstelle mit Hilfe von SCPI Befehlen

Control of a Keithley 2400 Sourcemeters via an RS-232 Interface using SCPI Commands

Vorgelegt von
Abdallah Battai

Erstprüfer: Prof. Dr.-Ing. Michael Karagounis
Zweitprüfer: Herr Rolf Paulus

Dortmund, den 15.08.2022

Abstrakt

In diesem Projekt wird erläutert, wie eine Verbindung zwischen einem Keithley2401 Sourcemeter und einem Linux Rechner über eine RS232-Schnittstelle hergestellt und das Gerät unter Verwendung der Programmierumgebung Qt-Creator angesteuert werden kann.

In vorhergehenden Projekten wurde dies bereits mit einem Linux Rechner und anderen Geräten durchgeführt. Zum einen wurde eine Verbindung mit einem Keithley 2460 Sourcemeter über eine GPIB Schnittstelle etabliert und in einem weiteren Projekt wurde ein Keithley DM5600 über die USB-Schnittstelle verbunden. Im vorliegenden Projekt liegt der Fokus auf einem Keithley 2401 Sourcemeter und der Kopplung über eine RS232-Schnittstelle.

Abstract

This project explains how a connection between a Keithley2401 sourcemeter and a Linux computer can be established and how the device can be controlled using the Qt Creator programming environment.

In previous projects, this has already been done with a Linux computer and other devices. On the one hand, a connection was established with a Keithley 2460 source meter via a GPIB interface and in another project, a Keithley DM5600 was connected via an USB interface. In this project the focus is on a Keithley 2401 sourcemeter and the coupling through a RS232 interface.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

RS232	Recommended Standard 232
GPIB	General Purpose Interface Bus
USB	Universal Serial Bus
SPCI	Standard Commands for Programmable Instruments
SMU	Source Meter Unit

1. Einleitung

Da elektronische Geräte stets an Popularität gewinnen, werden sie in der Fertigung und auch in anderen Branchen gebraucht. Voraussetzung für die Nutzung ist eine einwandfreie Funktion der Geräte. Um die richtige und gewünschte Funktionsweise zu garantieren, werden die Geräte regelmäßig getestet und gemessen. Um diese Messungen automatisiert durchzuführen, werden hingegen komplexe Messgeräte benötigt, da eine manuelle Messung fehleranfällig ist und heutzutage nicht mehr praktiziert wird. Auf dem Gebiet der Messtechnik sind die meisten Arbeitsabläufe geprägt von der modernen Elektronik. Die Messgeräte sind dabei von Branche zu Branche unterschiedlich.[1] [2]

Ziel dieser Projektarbeit ist es, einen bestehenden Programmcode so abzuändern, dass ein Quelle(Source) / Messgerät über eine RS232-Schnittstelle angesteuert werden kann. Diese Arbeit wird zudem für zukünftige Labormessungen zur Verfügung gestellt, um Weiterentwicklungen zu ermöglichen.

Zur Programmierung des Vorhabens wird die Entwicklungsumgebung Qt Creator verwendet. Diese ermöglicht eine Kommunikation zwischen einem Linux-Computer und dem Sourcemeter Keithley 2401 und eine Ansteuerung über entsprechende Befehle. Es sollen sowohl die Spannungs- und Stromwerte eingestellt werden können, welche die Quelle liefert als auch Messungen der eingestellten Spannungs- und Stromwerte ermöglicht werden.

Zunächst wurde das Gerät mit allen notwendigen Einstellungen konfiguriert und die Programmierumgebung Qt Creator eingerichtet. Nach der Etablierung der Kommunikation zwischen dem Gerät und dem Rechner wurde die verwendete Software weiterentwickelt. Die unterschiedlichen Funktionen des Sourcemeters konnten dann über die Software angesteuert werden. Abschließend wurden neue Klassen erstellt und in den bestehenden Programmcode integriert.

2. Software und Hardware

In diesem Kapitel werden die genutzten und benötigten Software und Hardware Komponenten vorgestellt und erläutert. Der Arbeitsplatz setzt sich zusammen aus einem PC mit CentOS-Betriebssystem, sowie einer Qt-Applikation, einem Keithley Sourcemeter SMU (Modell "Keithley 2401") einem Kabel, das an den seriellen Ports des PCs und des Sourcemeters angeschlossen wird und einer Platine mit schaltbaren Widerständen, die Testweise über zwei Kabel vom Sourcemeter versorgt werden können.

Mit dem Qt Creator wurde die Software zur Bedienung des Geräts mit dem PC entwickelt. Im Folgenden werden alle verwendeten Hard- und Software detailliert erklärt.

2.1 Hardware

2.1.1 Keithley SMU 2401

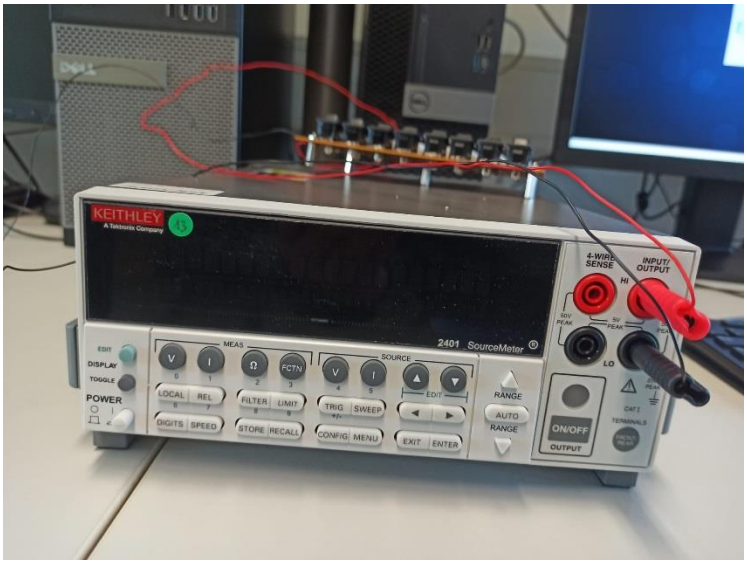

Abbildung 1: Keithley 2401

Das Modell 2401 von Keithley ist ein SourceMeter Multifunktionsgerät (SMU). Das Multifunktionsgerät ist hauptsächlich für Prüfanwendungen gedacht, die als Anforderungen eine enge Kopplung von Einspeisung und Messung haben. Mit dem Gerät kann demnach eine detaillierte Spannungs- und Stromeinspeisung, sowie Messung erfolgen. Das Modell 2410 besitzt

eine stabile und rauscharme DC-Stromquelle mit Readback und ein rauscharmes 6.5-stelliges Multimeter mit der Möglichkeit, den Messprozess zu wiederholen. Das DC-Parameter-Prüfgerät wird in der Herstellung von Bauelementen und Modulen, z. B. in der Kommunikations-, Halbleiter-, Computer-, Kfz- und Medizintechnik eingesetzt, da es viele verschiedene Charakterisierungs- und Testaufgaben während der Produktion übernehmen kann. [3]

Das Gerät hat eine Ausgangsleistung bis 20W und einen Spannungsmessbereich zwischen ± 200mV bis ± 20V. Der Strommessbereich liegt bei ± 1µA bis ± 1A. Die Schnittstellen GPIB und RS-232 sind ebenfalls bei Anschaffung bereits vorhanden. [4]

2.1.2 Serielle Schnittstelle RS-232

Allgemein sind serielle Schnittstellen notwendig, um einen physischen Austausch von Daten zwischen PC und Peripheriegeräten zu gewährleisten. Der Datenaustausch d.h. die Datenbits werden bei dieser Art von Schnittstellen nacheinander weitergeleitet, also seriell übermittelt. Die im Projekt verwendete Schnittstelle RS-232 wurde bereits in den 1960er entwickelt. Sie wird oft auch als COM-Schnittstelle oder V.24 bezeichnet. Einsatzbereiche dieser Schnittstelle waren vor allem die Telekommunikation beispielsweise bei Fernschreibern oder den späteren Modems und die EDV zur Anbindung von Terminals an Mainframes. [5]

Abbildung 2: RS232-Schnittstelle

Im Zeitalter der Digitalisierung und mit der rasanten Entwicklung von Elektrogeräten sind auch die Anforderungen gewachsen. Elektronische Geräte sollten und müssen

rechnergestützt ausgelesen und verwaltet werden können. Da die Schnittstelle RS-232 genau diese Anforderungen erfüllen konnte und sie zudem einfach zu implementieren ist, verbreitete sich diese Schnittstelle im Bereich der PC-Technik und der Unterhaltungselektronik. Die Schnittstelle eignet sich zum Beispiel zum Anschluss von Kassenterminals, Druckern und Messgeräten. Auch bei Plasmabildschirmen, Firmware, DVD-Playern und Satellitenreceiver wird die RS-232 genutzt. Erst als sich die USB-Schnittstelle (Universal Serial Bus) verbreitet hat, ist die Verbreitung der RS-232 rückläufig geworden. Vorher war nahezu jeder Computer mit mindestens einer RS-232-Schnittstelle ausgestattet. [5]

2.1.3 Platine mit Widerstand

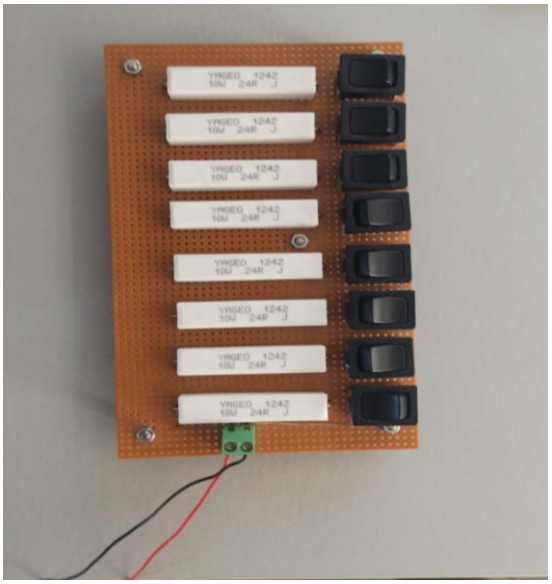

Abbildung 3: Platine mit Widerstand

Widerstände sind grundlegende, elektronische Bauelemente, welche weit verbreitet sind. Die Hauptaufgabe eines Widerstandes ist es den Strom zu limitieren bzw. zu regulieren oder einzustellen. [6]

Die genutzte Platine in diesem Projekt besteht aus 8 drahtgewickelten Widerständen und 8 Schaltern. Diese werden benötigt, um in einem geschlossenen Stromkreis Spannungen und Ströme durch das Keithley 2401 einzukoppeln und messen zu können.

2.2 Software

2.2.1 CentOS

CentOS 7 ist eine Linux-Distribution von Red Hat Enterprise Linux (RHEL). Diese Software ist quelloffen d.h. der Source-Code ist einsehbar. Außerdem ist die Software die frei verfügbar. Der Vorteil dieser CentOS-Software ist, dass sie binärkompatibel zu RHEL ist und nur leicht zeitversetzt zu den RHEL-Updates erscheint. Die gesamte Software ist zudem kostenfrei und kann daher auch frei genutzt werden. [7]

2.2.2 Qt-Creator

Die Software Qt-Creator ist eine Programmierumgebung, die Plattform übergreifend genutzt werden kann. Es ist eine Opensource-Software der Firma Nokia. Hauptsächlich wird Qt-Creator zur GUI-Programmierung eingesetzt. Der Vorteil dieser Entwicklungsumgebung ist, dass die geschriebenen Programme nach einer Neukompilierung kompatibel zu unterschiedlichen Betriebssystemen, wie zum Beispiel Linux und Windows sind. Falls andere Programmiersprachen genutzt werden sollen (z. B. Python, C#) können Qt-Erweiterungen verwendet werden. [2]

3. SPCI -Commands

Standard Commands for Programmable Instruments (SCPI) ist eine standardisierte Befehlssprache. Mit diesen Befehlen können elektronische Test- und Messgeräte programmiert, entwickelt und gesteuert werden. SCPI ist ein Teil des IEEE 488 Standards. Entwickelt wurde SCPI bereits in den 90er Jahren. SCPI beinhaltet nicht nur die Befehlsstruktur, sondern auch die notwendige Syntax und das Datenformat, welches zur Steuerung von Test- und Messgeräten notwendig sind. Zum Beispiel können ASCII Befehle wie CONFigure und MEASure genutzt werden. Diese werden in vielen programmierbaren Messgeräten benutzt. [8]

Befehle in SCPI können in Kurz- und Langform eingegeben werden. Der Benutzer kann demnach die Frequenz abfragen durch „FREQ“ oder „FREQuency“. Allgemein können viele unterschiedliche Befehle in SCPI genutzt werden. [8] Unter anderem sind dies folgende:

- Allgemeine Befehle (Common Commands),
- Subsystem-Befehle (Subsystem Commands),
- Auslese-Befehle (Query Commands) und
- Parameter-Befehle (Parameter Commands). [8]

Da in dem vorliegenden Projekt, sowohl im Testprogramm als auch in der Software, die ersten drei Befehlsarten vorrangig genutzt werden (Common Commands, Subsystem Commands, Query Commands), werden diese in den nachfolgenden Kapiteln detailliert erläutert.

3.1 SPCI Common Commands

Die Common Commands steuern nicht die Messgeräte im herkömmlichen Sinn, sondern die Schnittstellen. [8]

Die allgemeinen Befehle beginnen mit einem Stern (*). Da es sich um spezielle Systemkommandos handelt, werden sie ohne eine Angabe von Pfaden genutzt. Die Befehle werden mit einem Semikolon getrennt und können daher auch in Befehlsketten verwendet werden. [9]

Die in der Projektarbeit häufigsten verwendeten Common Commands sind:

*IDN? ➔ Gibt die ID (Identifikation) des Gerätes zurück

z. B. “KEITHLEY INSTRUMENTS INC., MODEL 2401“

*RST ➔ Startet das Gerät mit den Werkeinstellungen neu. [9]

3.2 SPCI Subsystem Commands

Die Subsystem-Befehle beinhalten alle gerätspezifischen Befehle. Hauptzweck dieser Kommandos ist die Steuerung der Geräte, um die Messungen durchzuführen. Eine Auflistung aller Befehle finden sich im Benutzerhandbuch des Herstellers. [2]

Die Befehle sehen mit der entsprechenden SCPI-Syntax zum Beispiel so aus:

:SENS:FUNC “VOLT: DC" ➔ Das Gerät wird auf DC Spannungsmessung gesetzt.

:SENS:VOLT:RANG 10 ➔ Die Ebene des Gerätes wird auf 10 gesetzt.

:SENS:VOLT:NPLC 10 ➔ Die Integrationsrate wird auf 10 gesetzt. [2]

3.3 SPCI Query Commands

Soll ein Parameter aus dem Gerät ausgelesen werden, wird ein Auslese-Befehl (Query Commands) verwendet. Zur Kennzeichnung eines Query Commands wird ein Fragezeichen am Ende gesetzt. [2] Zum Beispiel:

READ? ➔ Gibt einzelne Messwerte zurück.

FETCH? ➔ Liefert den letzten Messwert. [2]

3.4 SPCI Commands des Projektes

In dem vorliegenden Projekt wurden dieselben SCPI-Befehle genutzt, die bereits mit dem Messgeräte Keithley 2460 und DMM5600 funktioniert und programmiert wurden. Durch die Kompatibilität des Keithley 2401 Sourcemeter können die Messfunktionen, die für die beiden Geräte Keithley 2460 und DMM56000 programmiert wurden, wiederverwendet werden. Daher werden auch die Klassen, die in diesem Projekt programmiert wurde, aus den vorherigen Projekten übernommen. Da die SCPI-Befehle ebenfalls nahezu identisch sind, wird sowohl der Messaufbau als auch der Hauptprogrammcode nicht stark geändert. [2]

Die Geräte werden nicht für komplexe Messungen eingesetzt, sondern für Standardmessungen. Die Standardmessungen sind hierbei Gleich- oder Wechselspannungsmessungen und Gleich- oder Wechselstrommessungen. Daraus resultiert, dass keine neuen Messfunktionen für das Keithley 2401 Sourcemeter programmiert werden müssen. [2]

Die im Projekt am häufigsten genutzten Befehle und eine Erläuterung dazu, sind folgende:

- ":SOUR:FUNC:MODE VOLT": Mit diesem Befehl lässt sich auf die Spannungsquelle des Gerätes zugreifen.
- ":SOUR:VOLT:MODE FIXED": Der Modus „fixed“, also der Festspannungsquellenmodus wird gewählt.
- ":SOUR:VOLT:RANG:AUTO ON": Dieser Befehl erlaubt es einen automatischen Quellenbereich einzustellen.
- ":SOUR:VOLT:LEV:IMM 0": Hiermit kann der Wert der Ausgangsspannung eingestellt werden.
- ":SENS:FUNC:CONC ON": Der Befehl aktiviert die gleichzeitige Messung.
- ":SENS:FUNC:OFF:ALL": Alle Funktionen, die nicht zu den Hauptfunktionen zählen, werden durch diesen Befehl deaktiviert.
- ":SENS:FUNC:ON 'VOLT:DC','CURR:DC'": Durch diese Befehlskombination werden Spannungs- und Strom-Funktionen aktiviert.
- ":SENS:AVER:STAT OFF": Hiermit können die Messwertmittelungsfunktion deaktiviert bzw. ausgeschaltet werden. [10]

Die SPCI Befehle zum Messen bzw. zum Ausgeben der Spannung und des Stroms können hingegen durch folgende Befehle eingestellt werden:

- ":FORM:DATA ASCII": Durch diesen Befehl kann das Datenformat für die Übertragung von Messwerten über den Bus ausgewählt werden. Es ist zu beachten, dass über die RS 232-Schnittstelle nur das ASCII-Format erlaubt ist.
- ":FORM:ELEM VOLT,CURR": Wird dieser Befehl genutzt, liefert das Gerät die Messwerte der Spannung und des Stroms.
- ":SENSE:CURR:PROT": Mit diesem Befehl wird eine Strombegrenzung auf Englisch: „current compliance“ konfiguriert.
- ":OUTP ON": Der Ausgang wird eingeschaltet. [10]

4. Testprogramm für die Kommunikation

Das Testprogramm hat zum Ziel die Kommunikation zwischen dem Gerät Keithley 2401 und dem Rechner zu etablieren und auszuprobieren, wie die serielle Schnittstelle geöffnet und geschlossen werden kann und wie Daten von der seriellen Schnittstelle gelesen bzw. Daten auf die serielle Schnittstelle geschrieben werden können.

Zur Erstellung eines neuen Projektes wird zunächst der Qt-Creator gestartet. Dann wird ein neues Projekt angelegt und im nächsten Schritt eine passende Anwendung ausgewählt. Das Projekt zur Entwicklung des Testprogramms erhält den Namen „Seriell_Test“.

```
#include <QCoreApplication>
#include <QSerialPort>
#include <QFile>
#include <QTextStream>
#include <iostream>
#include <sstream>
#include <string>
#include <QDebug>
#include <cstdlib>
```

Quellcode 1: Testprogramm

Im Quellcode 1 werden zunächst alle notwendigen Includes für Qt Creator und C++ eingefügt und aufgerufen. Die wichtigsten Includes sind:

- <QCoreApplication>: Diese Klasse wird von Nicht-GUI-Anwendungen verwendet. Sie dient dazu Ereignisschleife bereitzustellen und genau ein QCoreApplication-Objekt auszugeben. [11]
- <QSerialPort>: Dies ist eine Bibliothek. Die den Zugriff auf physikalischer Ebene auf die serielle Schnittstelle ermöglicht. [12]
- <iostream> : Dies ist ebenfalls eine Bibliothek, die als Grundlage zur Verwaltung von Ein- und Ausgangsdatenströmen in C++ Anwendungen verwendet wird. [13]
- <QTextStream>: Entspricht der Include-Datei eines Streaming-Operators. Mit diesem Operator können Wörter und Zahlen einfacher ausgelesen und geschrieben werden. Der Operator lässt sich auf einem QIODevice ausführen, um QByteArrays oder QStrings zu übertragen. [14]

- <string> und "QString": Dies sind zwei weitere Bibliotheken, die den Umgang mit Strings in C++ ermöglichen. Die QString-Bibliothek ist speziell für die QT-Umgebung entwickelt und optimiert worden. [15]
- <QFile> : Dies ist eine Bibliothek für die Verwendung von E/A-Geräten. Dadurch können Text- und Binärdateien gelesen und geschrieben werden. Zur Verwendung kann ein QTextStream oder QDataStream genutzt werden. Alternativ kann QFile auch alleine stehen. [16]
- <QDebug> : Diese Bibliothek ermöglicht Daten anzuzeigen und vereinfacht die Fehlersuche während der Entwicklungsphase des Projekts. [17]

```
QSerialPort serial ("/dev/ttyS0");
serial.setPortName("/dev/ttyS0");
serial.setBaudRate(QSerialPort::Baud9600);
serial.setDataBits (QSerialPort::Data8);
serial.setFlowControl (QSerialPort::NoFlowControl);
serial.setParity(QSerialPort::EvenParity);
serial.open(QIODevice::ReadWrite);
serial.clear();
serial.write (QString("*RST\n").toUtf8());
serial.write (QString("*IDN?\n").toUtf8());
QByteArray responseData = serial.readAll();
 while (serial.waitForReadyRead(10))
  responseData += serial.readAll();
QString line = QString::fromUtf8(responseData);
qDebug() << "Ausgelesen:" <<line;
```

Quellcode 2: Testprogramm

Im Quellcode 2 wird ein Objekt der Klasse QSerialPort erzeugt. Hierfür wird der Gerätepfad auf die verwendete RS232-Schnittstelle benötigt.

Daher wird im Terminal des Betriebssystems CentOS der Inhalt des /dev-Ordners gelistet. Hierfür wurde zunächst der Befehl cd /dev eingegeben. Anschließend wurde mit Hilfe des Befehls ls der Inhalt des Ordners angezeigt. In diesem Ordner wurden einige mögliche Geräte angezeigt, die für den Zugriff auf die serielle Schnittstelle in Frage kommen. Durch Trail & Error wurde schließlich erkannt, dass das Gerät /dev/ttyS0 den Zugriff auf die serielle Schnittstelle

ermöglicht. Nachdem ein Name vergeben wurde, ist eine Initiierung der erforderlichen Parameter notwendig, welche ebenfalls im Gerät Keithley 2401 manuell eingestellt wurden. [2]

Nun werden die notwendigen Parameter der seriellen Schnittstelle definiert.

- serial.setBaudRate(QSerialPort::Baud9600)

Zunächst wird die Methode setBaudrate genutzt. Die Baudrate ist eine Einheitsform, mit welcher die Schrittgeschwindigkeit angegeben wird. Genutzt wird diese häufig in der Kommunikationstechnik, aber auch im Zusammenhang mit seriellen Schnittstellen. [18]

In diesem Projekt wird die Methode setBaudrate mit dem Wert Baud9600 und dem Aufzählungstyp „enum“ gesetzt. Es wird also festgelegt, dass 9600 Bit pro Sekunde übertragen werden sollen. Im Anschluss wird die Konfiguration dem Objekt „serial“ zugewiesen.

- serial.setDataBits (QSerialPort::Data8)

DataBits ist eine Einheit zur Messung der Länge von Daten, die über eine Schnittstelle übertragen werden können, z. B. Gerätebefehle, Sensormesswerte, Fehlermeldungen. Die Daten werden dabei als Binärdaten oder als Textdaten weitergeleitet. [19]
Im Rahmen dieses Projektes wird die Länge der Daten auf 8 Bits festgelegt und danach in „serial“ zugewiesen.

- serial.setFlowControl (QSerialPort::NoFlowControl)

Zur Begrenzung des Datenflusses haben serielle Geräte ein „FlowControl“ System. Durch dieses Verfahren kann die übertragene Datenmenge von dem Gerät selbst gesteuert werden. [20]
Dabei gibt es drei verschiedene Konstanten des FlowControls:

1. RTS/CTS- Hardware FlowControl: Dies ist ein Mechanismus bzw. ein Teil der seriellen Schnittstelle, welcher zwei weitere Pins RTS „Request to Send“ und CTS „Clear to Send“ am RS232-Anschluss nutzt. Durch diese Leitungen kann eine Kommunikation zwischen Empfänger und Sender stattfinden. [21]
2. XON/XOFF-Software FlowControl: Hierbei werden Datenübertragungssteuerzeichen entlang des Datenstroms (TxD und RxD) gesendet. [20]

3. NoFlowControl: Diese Konfiguration wird im vorliegenden Projekt genutzt, welche weder Hardware- noch Software elemente zur Steuerung des Datenflusses nutzt. [12]

- serial.setParity(QSerialPort::EvenParity)

Die Paritätsprüfung wird durch „parity“ durchgeführt. Dies ist ein Prozess der Sicherstellung der Kommunikation der Datenübertragung zwischen Knoten. An die ursprünglichen Datenbits wird jeweils ein Paritätsbit angehängt. So werden gerade („even“) oder ungerade („odd“) Bits erzeugt. [22]

Der Programmcode des Projekts verwendet setPartiy durch den Aufzählungstyp „enum“ mit EvenParity, also mit einer geraden Anzahl an gesetzten Bits.

In der nächsten Zeile des Codes wird mit dem open() Befehl und dem Attribut „QIODevice::ReadWrite“ das Objekt serial geöffnet, damit auf die serielle Schnittstelle sowohl geschrieben als auch davon ausgelesen werden kann. Danach werden SPCI-Befehle versendet, welche es ermöglichen zu prüfen, ob die Kommunikation zwischen dem Gerät Keithley 2401 und dem Rechner hergestellt wurde.

- Der Befehl "*RST\n" setzt die Schnittstelle des Messgeräts zurück.
- Der Befehl "*IDN?\n" gibt die ID des Messgerätes zurück.

Wie dem Code zu entnehmen ist, müssen alle Befehle mit dem Sonderzeichen für den Zeilenumbruch "\n" terminiert werden. Andernfalls reagiert das Messgeräte nicht, weil es annimmt, dass der Befehl unvollständig ist.

Um die Daten zu speichern, die das Keithley 2401 Sourcemeter an den Rechner sendet, wird das Objekt „responsData“ von „QByteArray“ erzeugt. Die serielle Schnittstelle bzw. das Obket serial wird mit der Funktion „readAll()“ausgelesen und alle ausgelesenen Daten in „responsData“ gespeichert. Die Funktion „waitForReadyread ()“ blockiert die Ausführung des Programms für die als Parameter angegebene Zeit in Millisekunden. Gibt die Funktion eine 1 zurück, sind neue Daten erhalten worden. Gibt die Funktion eine 0 zurück, sind während der Wartezeit keine neuen Daten empfangen worden. Diese Funktion wird mit 10 ms Wartezeit in einer while Schleife verwendet, um sicherzustellen, dass alle übertragenen Daten durch readAll () ausgelesen worden sind. Erst dann wird die Ausführung des Programms fortgesetzt.

In der darauffolgenden Zeile wird „responsData“ in ein „QString“ umgewandelt und im deklarierten Objekt „QString“ zugewiesen. Anschließend werden alle umgewandelten Daten, die in „line“ gespeichert worden sind, mit der Funktion „qDebug()“ ausgegeben.

```
  std::string TempV;   //lokale Variable für die Spannung
  std::string TempC;  //lokale Variable für den Strom
  std::stringstream ssV;
  ssV << 1.0;
  std::stringstream ssC;
  ssC << 1.0;
  serial.write (QString(":SOUR:FUNC:MODE VOLT\n").toUtf8()); // Auswahl die Spannungsquellenfunktion
  serial.waitForBytesWritten(30000);
  serial.write (QString(":SOUR:VOLT:MODE FIXED\n").toUtf8()); // Auswahl vom Festspannungsquellenmodus
  serial.waitForBytesWritten(30000);
  serial.write (QString(":SOUR:VOLT:RANG:AUTO ON\n").toUtf8()); // Auswahl der automatischen Quellenbereich
  serial.waitForBytesWritten(30000);
  serial.write (QString(":SOUR:VOLT:LEV:IMM 0\n").toUtf8()); // Auswahl des Niveaus der Spannungsquelle
  serial.waitForBytesWritten(30000);
  serial.write (QString(":SENS:FUNC:CONC ON\n").toUtf8()); // Aktivierung der gleichzeitigen Messungen
  serial.waitForBytesWritten(30000);
  serial.write (QString(":SENS:FUNC:OFF:ALL\n").toUtf8()); // Deaktivierung nebenläufige Funktionen.
  serial.waitForBytesWritten(30000);
  serial.write(QString(":SENS:FUNC:ON  'VOLT:DC','CURR:DC'\n").toUtf8()); // Aktivierung der Spannung- und Strom-Funktion
  serial.waitForBytesWritten(30000);
  serial.write (QString(":SENS:AVER:STAT OFF\n").toUtf8()); // Deaktivierung der Filter
  serial.waitForBytesWritten(30000);
 // Only Output voltage and current
  serial.write (QString(":FORM:DATA ASCII\n").toUtf8());
  serial.waitForBytesWritten(30000);
  serial.write (QString(":FORM:ELEM VOLT,CURR\n").toUtf8()); // Ausgabe des Messwertes der Spannung und des Stroms
  serial.waitForBytesWritten(30000);
  TempC = ":SENSE:CURR:PROT "; // Mit diesem Befehl wird das Gerät die Konformität vom Strom auf Englisch current compliance gesetzt.
  TempC = TempC + ssC.str()+"\n";
  QString qtempc = QString::fromStdString(TempC);
```

```
  TempV = ":SOUR:VOLT:LEVEL:IMM "; // Das Gerät wird Angegebenen Niveau auf Englisch Level sofort gesetzt
  TempV = TempV + ssV.str()+"\n";
  QString qtempv = QString::fromStdString(TempV);
  serial.write (QString(":SOUR:VOLT:RANG:AUTO ON\n").toUtf8());// Auswahl der automatischen Quellenbe-
reich
  serial.waitForBytesWritten(30000);
  serial.write(qtempc.toUtf8());
  serial.waitForBytesWritten(30000);
  serial.write (QString(":SOUR:VOLT:RANG:AUTO ON\n").toUtf8());// Auswahl der automatischen Quellenbe-
reich
  serial.waitForBytesWritten(30000);
  serial.write(qtempv.toUtf8());
  serial.waitForBytesWritten(30000);
  serial.write(QString(":OUTP ON\n").toUtf8()); // Einschaltung des Ausgangs
  serial.waitForBytesWritten(30000);
  serial.write (QString(":READ?\n").toUtf8()); // Aktivierung und Erfassung der Daten
  serial.waitForBytesWritten(30000);
  serial.waitForReadyRead(30000);
  responseData = serial.readAll();
   while (serial.waitForReadyRead(10))
    responseData += serial.readAll();
   line = QString::fromUtf8(responseData);
   qDebug() << "Read Ausgelesen:" <<line;
```

Quellcode 3: Testprogramm (Kommentare: Series 2400 SourceMeter User's Manual) [10]

Nach der ersten Antwort des Gerätes, welche die angefragte ID beinhaltet, soll als nächstes eine Ausgangsspannung und eine Strombegrenzung eingestellt werden. Dazu werden im Testprogramm „Seriell_Test“ die entsprechenden SPCI-Befehle über das Objekt „serial“ versendet.

Zu Beginn werden „TempV“ und „TempC“ als lokale String Objekte deklariert, mit deren Hilfe die Befehle für das Setzen einer Spannung und einer Strombegrenzung zusammengesetzt werden sollen. Danach werden die Objekte „ssV“ und „ssC“ als String Stream deklariert und die Spannung auf 1V und die Strombegrenzung auf 1A gesetzt.

Danach werden SPCI-Befehle zur Einstellung der Spannung und zur Messung des Laststroms versendet.

Die Befehle für die Einstellung der Spannung und der Strombegrenzung werden aus einer Kombination aus String Objekt und Zahlenwert zusammengesetzt, um flexibel Werte wählen zu können.

Zunächst wird dem String Objekt TempC der erste Teil des Befehls für die Festlegung der Strombegrenzung zugewiesen.

```
TempC = ":SENSE:CURR:PROT ";
TempC = TempC + ssC.str()+"\n";
QString qtempc = QString::fromStdString(TempC);
```

Dann wird mit Hilfe der Methode str() des Stringstreams der zu übermittelnde Zahlenwert in einen QString Objekt gewandelt und mit Hilfe des Konkatinierungsoperators "+" mit dem ersten Befehlsteil und dem Sonderzeichen des Zeilenumbruchs "\n" zusammengesetzt. Daraufhin wird TempC in einen QString konvertiert und alle zusammengesetzten Stromwerte dem „QString qtempc“ zugewiesen.

In gleicher Weise wird der Befehl für das Setzen der Spannung zusammengesetzt. Zunächst wird dem String Objekt TempV der erste Teil des Befehls für die Festlegung der Quellenspannung zugewiesen.

```
TempV = ":SOUR:VOLT:LEVEL:IMM ";
TempV = TempV + ssV.str()+"\n";
QString qtempv = QString::fromStdString(TempV);
```

Dann wird mit Hilfe der Methode str() des Stringstreams der zu übermittelnde Zahlenwert in ein QString Objekt gewandelt und mit Hilfe des Konkatinierungsoperators "+" mit dem ersten Befehlsteil und dem Sonderzeichen des Zeilenumbruchs "\n" zusammengesetzt. Schließlich wird TempV konvertiert darauf in einen QString und qtempv zugewiesen.

Der Befehl für das Setzen der Ausgangsspannung und der Strombegrenzung wird schließlich mit der Methode write() dem Objekt serial übergeben.

```
serial.write (QString(":SOUR:VOLT:RANG:AUTO ON\n").toUtf8());// Auswahl der automatischen Quellenbereich
serial.waitForBytesWritten(30000);
serial.write(qtempc.toUtf8());
serial.waitForBytesWritten(30000);
serial.write (QString(":SOUR:VOLT:RANG:AUTO ON\n").toUtf8());//Auswahl der automatischen Quellenbereich
serial.waitForBytesWritten(30000);
serial.write(qtempv.toUtf8());
```

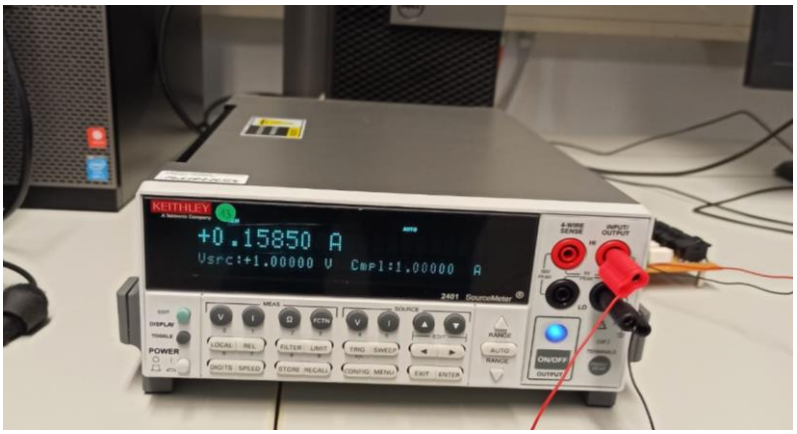

Abbildung 4: Einstellung der Ergebnisse (Testprogramm)

Sobald die Kommunikation und Ansteuerung zwischen dem Gerät und dem Rechner erfolgreich sind, wird eine Versorgungsspannung (Vsrc) von 1V bei einer Strombegrenzung (Cmpl) von 1A eingestellt, wie in Abbildung 4 zu sehen ist. Als Laststrom wurde über die SCPI Befehle ein Wert von 0.15850 A ausgelesen, was ebenfalls in Abbildung 4 zu sehen ist.

5. Projekt_RS232

Für die Umsetzung der Hauptaufgabe muss die Kommunikation zwischen dem Keithley 2401 über die RS232-Schnittstelle einwandfrei funktionieren. Die Ausführung des Testprogramms musste demnach zunächst vollständig und einwandfrei ausgeführt werden können.

Die SPCI Schnittstelle des Keithley 2401 Sourcemeters ähnelt sehr der Schnittstelle des Keithley 2460 Sourcemeters und des DMM6500 Multimeters. Aus diesem Grund werden drei neue Klassen eingeführt, die an die Klassenstruktur der Software für die Ansteuerung der zuvor genannten Geräte angelehnt ist. Um einen Zugriff vom PC auf das RS232-Gerät zu ermöglichen, werden zunächst die Klassen „RS232-Interface“ und „RS232-Device“ programmiert. Danach wird die Klasse „RS232_Keithley2401“ erstellt, die auf Methoden dieser beiden Klassen zurückgreift. Die Klasse „RS232_Keithley2401" beinhaltet die notwendigen Messfunktionen, auf die mithilfe von SCPI-Kommandos zugegriffen werden kann. Diese drei Klassen können dann in das bestehende Projekt eingebunden werden, um das Sourcemeter anzusprechen.

5.1 Klassenstruktur

Bevor der Quellcode erläutert wird, soll zunächst schematisch anhand einer Grafik die Klassenstruktur erläutert werden.

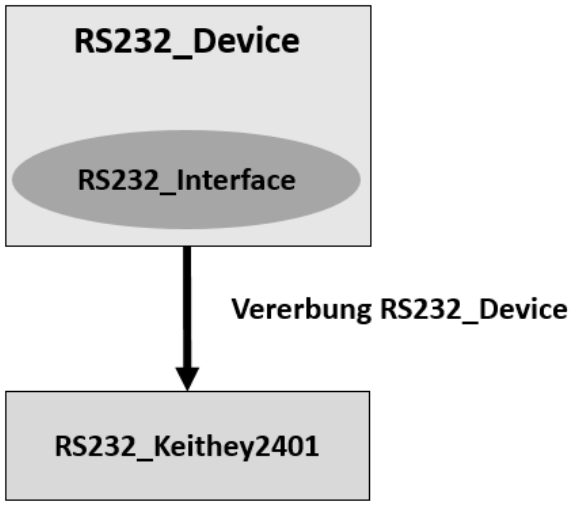

Abbildung 5: Verlauf des Projektes am Gerät Keithley2401

Die Abbildung 5 zeigt die drei Klassen, die im Projekt vorhanden sind und miteinander gekoppelt werden. Auf der obersten Abstraktionsebene befindet sich die Klasse

RS232_Keithley2401, die dem Nutzer High-Level Funktion für die Steuerung und Auslesung des Multimeters bietet. Die Klasse RS232_Keithley2401 erbt alle Objekte und Methoden der Klasse RS232_Device. Die Klasse RS232_Device erzeugt ein Objekt der Klasse RS232_Interface und greift über die Methoden Send und SendandReceive auf die serielle Schnittstelle zu. In der Klasse RS232_Interface wird die serielle Schnittstelle geöffnet und geschlossen und die Befehle für den lesenden und schreibenden Zugriff auf die serielle Schnittstelle durch die Methoden Send und SendandReceive gekapselt.

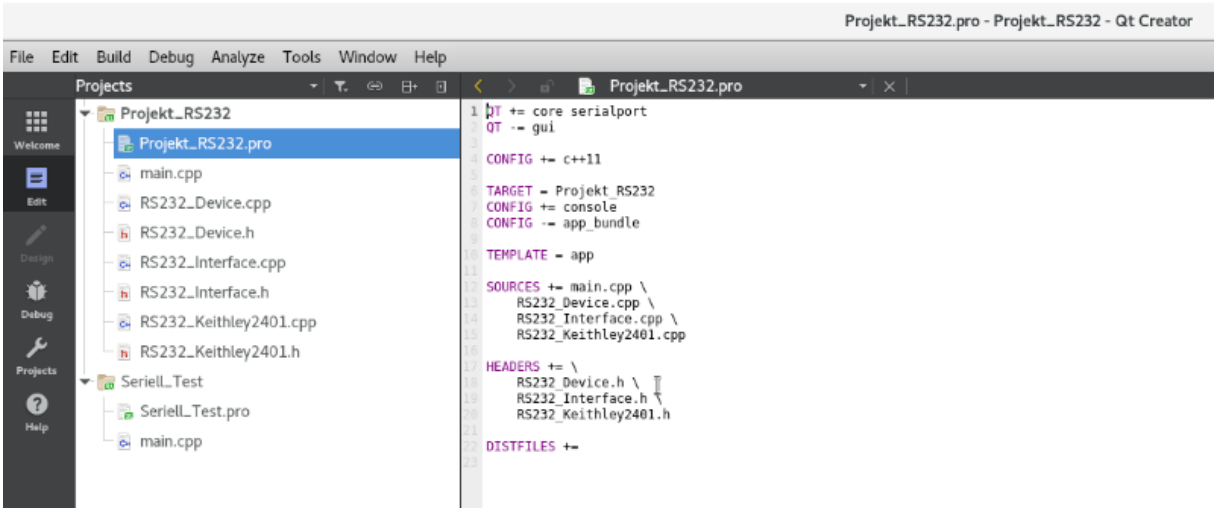

Abbildung 6: Verlauf des Projektes von Qt Creator

In Abbildung 6 sind die Dateien des Projektes „Projekt_RS232“ und des Testprogramms „Seriell_Test“ dargestellt.

5.2 RS232_Interface

RS232_Interface ist eine der wichtigste Klassen des Programmcodes, da alle Methoden zum Zugriff auf die serielle Schnittstelle in abstraktere leicht zu verwendende Methode gekapselt werden, die dann den Klassen RS232_Device bzw. RS232_Keithley2401 zur Verfügung gestellt werden.

```
class RS232_Interface
{
public:
    RS232_Interface(QString DevPath);
    ~RS232_Interface();
    void Send(QString Msg);
```

```
    QString SendAndReceive (QString Msg);
protected:
  QSerialPort *serial;
};
```

Quellcode 4: RS232_Inteface.h

Im Quellcode 4 werden zunächst der Konstruktor, der Destruktor, und alle Methoden Send und SendAndReceive als „Public“ deklariert. Zudem wird ein „Protected“ Objekt der Klasse QSerialPort angelegt. Der Konstruktor hat ein Paramater „DevPath“ und ist zunächst leer. „Send“ und „SendAndReceive“ werden mit dem Parameter „Msg“ deklariert, der den zu übertragenden Befehl beinhaltet.

```
RS232_Interface::RS232_Interface(QString DevPath){
    QFile Pfad1(DevPath);
        if(Pfad1.exists(DevPath))
        {
            serial = new QSerialPort(DevPath);
            serial->setPortName(DevPath);
            serial->setBaudRate(QSerialPort::Baud9600);
            serial->setDataBits(QSerialPort::Data8);
            serial->setFlowControl(QSerialPort::NoFlowControl);
            serial->setParity(QSerialPort::EvenParity);
            if (serial->open(QIODevice::ReadWrite))
            {
                serial->clear();
                serial->write (QString("*RST\n").toUtf8());
            }
            else
            {
                qDebug()<< "Gerät kann nicht geöffnet werden\r\n";
            }
         }
         else
         {
                qDebug()<< "Gerät existiert nicht\r\n";
         }
}
```

Quellcode 5: RS232_Interface

Zunächst wird ein QFile Objekt mit dem Namen „Pfad1“ angelegt und der Parameter „DevPath“ zugewiesen.

In der zweiten Zeile des Codes wird überprüft, ob der Pfad1 tatsächlich existiert. Dann wird ein neues Objekt „serial“ der Klasse „QSerialPort“ erzeugt und der Name des Serial Objekts auf den Gerätenamen samt Pfad DevPath gesetzt. Darauf aufbauend werden dann alle Parameter zur Einstellung der seriellen Schnittstelle gesetzt.

In der nächsten Zeile des Codes wird die serielle Schnittstelle bzw. das Objekt „serial“ mit lesendem und schreibenden Zugriff (QIODevice::ReadWrite Befehl) zur weiteren Bearbeitung geöffnet. Danach werden alle Eingangs- und Ausgangspuffer über die Methode clear() und das Sourcemeter über den SCPI Befehl *RST zurückgesetzt.

Die Befehle werden wie im Testprogramm über die Methode write() auf die serielle Schnittstelle geschrieben. An dieser Stelle sei erwähnt, dass nach der Übersendung des SCPI Befehls *RSTn ein Fehler auftritt, wenn direkt nach dem Aufruf der Methode write() die Methode waitForBytesWritten() verwendet wird. Aus diesem Grund wurde der wait Befehl an dieser Stelle weg gelassen.

Um den Messgeräten Befehle zu zusenden und Ergebnisse auszulesen, werden die Funktionen „Send()“ und „SendAndReceive()“ ausgeführt.

```
void RS232_Interface::Send(QString Msg)
//Die Funktion send sendet nur den Befehl an das angeschlossene Gerät
{
                        if (serial!=NULL)
                        {
                            qDebug() << "Will schreiben" << Msg;
                            serial->write(Msg.toUtf8());
                            serial->waitForBytesWritten(30000);
                        }
                        else
                        {
                    qDebug()<< "Gerät kann nicht geöffnet werden\r\n";
                        }
}
```

Quellcode 6: RS232_Interface_Send-Funktion

Durch eine if-Anweisung wird ermittelt, ob die serielle Schnittstelle noch geöffnet ist. Ist dies der Fall, wird eine vorgegebene Botschaft „Msg“ dem Objekt „serial“ durch die Methode write zugewiesen. Da die Methode write() einen UTF kodierten 8-Bit String erwartet, während die Nachricht Msg einem QString entspricht, wird die Methode toUTF8() der Klasse QString für die Parameterübergabe verwendet. Ist die serielle Schnittstelle nicht mehr geöffnet, wird mit der „qDebug ()“-Funktion eine Nachricht ausgegeben, dass kein angeschlossenes Gerät gefunden wurde.

```
QString RS232_Interface::SendAndReceive(QString Msg)
{
                  QString Result;
                  QByteArray responseData;

                       if (serial!=NULL)
                          {
                            qDebug() << "Will schreiben" << Msg;
                            serial->write(Msg.toUtf8());
                            serial->waitForBytesWritten(30000);
                            serial->waitForReadyRead(30000);
                            responseData = serial->readAll();
                            while (serial->waitForReadyRead(10))
                              responseData += serial->readAll();
                            Result = QString::fromUtf8(responseData);
                            qDebug() << "Result: " << Result << "\n";
                           }
                        else
                          {
                        qDebug()<< "Gerät kann nicht geöffnet werden\r\n";
                          }
   return Result;
}
```

Quellcode 7: RS232_Interface_SendAndReceive-Funktion

Die Methode „SendAndReceive()“ ist sehr ähnlich zur Methode „Send()“ aufgebaut. Der Unterschied besteht darin, dass nach der Versendung einer Nachricht zusätzlich noch die Antwort des Sourcemeters ausgelesen wird. Hierfür wird wie im Testprogramm die Methode readAll() in Kombination mit der Methode waitForReadyRead() verwendet.

Die als UTF-8 String eingelesenen Daten werden im QByteArray responseData gespeichert und schließlich über die Methode fromUtf8 der Klasse String in ein Objekt der Klasse QString gewandelt und dem Objekt Result zugewiesen, das als Rückgabe-Parameter an die aufrufende Methode übergeben wird.

5.3 RS232_Device

```
class RS232_Device
{
public:
               ~RS232_Device();
                RS232_Device   (QString DevPath);
      QString           GetName();
      QString              isOk;
protected:
      void          Send (QString Str); // Error checked in Interface
      QString       SendAndReceive (QString Str); // return false if error
      RS232_Interface   * MyIf;          // My interface
      QString           MyDevPath;       // My Serialnumber of device
      QString           MyName;          // My Name (like 'Keithley2401')
};
```

Quellcode 8: RS232_Device.h

Die im Quellcode 8 dargestellte „Header-Dateien“ der Klasse RS232_Device definiert den Namen des Konstruktors und Destruktors und die Methoden GetName() und den QString isOk als „Public“.

Die Methode GetName() kann verwendet werden, um die vollständigen Gerätebezeichnung des Sourcemeters auszulesen. isOk ist ein QString, der Auskunft darüber gibt, ob die Komunikation mit dem Sourcemeter korrekt funktioniert.

Als „Protected“ werden die Methoden Send und SendAndRecive deklariert. Außerdem wird ein Zeiger der Klasse RS232_Interface deklariert, wodurch nach Erzeugung eines Objekts dieser Klasse alle Methoden der Klasse RS232_Interface in der Klasse RS232_Device aufgerufen werden können.

Der Gerätename bzw DevicePath MyDevPath und der Name des Device MyName sind ebenfalls als „Protected“ deklariert worden.

```
RS232_Device::~RS232_Device(){}

RS232_Device::RS232_Device (QString DevPath)
 {
  MyIf = new RS232_Interface (DevPath); // von der RS233_Intrface Klasse
  MyDevPath = DevPath;
  qDebug()<< "Device with Path "<<MyDevPath;
  MyName = " Device with Path " + MyDevPath;
 }
void RS232_Device::Send (QString Str)    // return false if error
{
   MyIf->Send(Str);
}
QString RS232_Device::SendAndReceive (QString Msg)
{
  return MyIf->SendAndReceive(Msg);
}
// QString RS232_Device::GetName(){ ➔ wird hier kommentiert
   // return MyName;}

//void RS232_Device::Receive (QString Msg){  // return false if error
  //  return Msg;}
```

Quellcode 9: RS232_Device.cpp

Im Konstruktor der Klasse RS232_Device wird ein RS232_Interface-Objekt erzeugt. Der Konstruktor erhält den Gerätpfad DevPath als Paramater. Somit können wie bereits erwähnt alle Methoden verwendet werden, die sich in der RS232_Interface Klasse befinden und zum Beispiel send() oder SendAndReceive() aufgerufen werden.

5.4 RS232_Keithley2401

```
class RS232_Keithley2401: public RS232_Device      //Vererbung Device
{
public:
          RS232_Keithley2401 (QString DevPath);  //RS232_ID_K2401
          ~RS232_Keithley2401();
   void   SetVoltage       (double voltage, double maxcurrent=0);
   void   SetCurrent       (double current, double maxvoltage=0);
   void   Measure          (double &current, double &voltage, TVoltageUnit
VoltUnit = VOLT, TCurrentUnit CurrUnit = AMP);
   double   VSweep          (double &delay, double &steps, double &start,
double &end, double &limit, double &vrange, double &crange);
   double   CSweep          (double &delay, double &steps, double &start,
double &end, double &limit, double &vrange, double &crange);
   void   SwitchOff();
   void   SwitchOn();
   void   Beep();
   void   SetSpeed           (TDeviceSpeed Speed);
   void   DisplayText        (QString Msg);
   void   DisplayOff         (void);
private:
   bool   DeviceResponds     ();           // Test if device responds
   TDeviceStatus DeviceStatus;           // Remember setting of Multimeter
};
```

Quellcode 10: RS232_Keithley2401.h

Aus dem in Quellcode Nr10. dargestellten Header ist ersichtlich, dass die RS232_Keithley2401 die Klasse RS232_Device erbt und dementsprechend alle Methoden und Objekte dieser Klasse ebenfalls besitzt. Der Konstruktor erhält den Geräte-Pfad als Parameter.

Darüber hinaus sind weitere Methoden als Public definiert, die für die Verwendung des Sourcemeters nützlich sind.

In diesem Projekt wurden insbesondere die Methoden SetVoltage() und SetCurrent() zum Setzen von Versorgungsspannungen bzw. strömen und die Methode Measure zur Messung der Spannung am Messgerät und den Laststrom, den das Gerät liefert, untersucht. Darüber hinaus existieren noch weitere Methode z.B. SwitchOn und SwitchOff zum Ein- und Auschalten des Ausgangs des Messgeräts, VSweep() und CSweep() zur Aufnahme von Strom- oder Spannungskennlinien usw., die jedoch hier nicht weiter besprochen werden, weil Sie aus älteren Projekten stammen.

```
RS232_Keithley2401::RS232_Keithley2401 (QString DevPath): RS232_Device
(DevPath)
{
  MyName   = "Keithley Multimeter Type 2401 with ID = " + MyDevPath;
  if (!DeviceResponds()) {
        isOk = "false";
    MyIf = NULL;
    return;
  }
// Initilize Source and set to 0V, 0.1mA limit
    Send(":SOUR:FUNC:MODE VOLT\n"); // Auswahl die Spannungsquellenfunktion
    DeviceStatus = VOLTAGEMODE; // Devicestatus : Voltagemode
    Send(":SOUR:VOLT:MODE FIXED\n");// Auswahl vom Festspannungsquellenmodus
    Send(":SOUR:VOLT:RANG:AUTO ON \n"); // Auswahl der automatischen Quellenbereich
    Send(":SOUR:VOLT:LEV:IMM 0\n"); // Auswahl des Niveaus der Spannungsquelle

// Measure only current and voltage
  Send(":SENS:FUNC:CONC ON\n");
  Send(":SENS:FUNC:OFF:ALL\n");
  Send(":SENS:FUNC:ON  'VOLT:DC','CURR:DC'\n");
  Send(":SENS:AVER:STAT OFF\n");

// Only Output voltage and current
  Send(":FORM:DATA ASCII\n");
  Send(":FORM:ELEM VOLT,CURR\n");

// Output on
 Send(":OUTP ON\n");
}
```

Quellcode 11: RS232_Keithley2401

Im Konstruktor der Klasse RS232_Keithley2401 wird über die Methode DeviceResponds() überprüft, ob die Kommunikation zum Sourcemeter fehlerfrei funktioniert. Wenn die Antwort des Geräts nicht dem erwarteten Ergebnis entspricht, wird die Bearbeitung des Konstruktors abgebrochen. Das Objekt der Klasse Interface MyIf wird auf Null zurückgesetzt. Andernfalls wird die Ausgangsspannung des Sourcemeters initial auf null gesetzt, der Voltagemodus eingestellt und der Ausgang wird eingeschaltet.

```
void RS232_Keithley2401::SetVoltage (double voltage, double maxcurrent)
{
 QString TempV; //lokale Variable für die Spannung
 QString TempC; //lokale Variable für den Strom
 QString ssV = QString::number(voltage);
 QString ssC = QString::number(maxcurrent);

 if (DeviceStatus != VOLTAGEMODE) {
        Send(":SOUR:FUNC:MODE VOLT\n");
        DeviceStatus = VOLTAGEMODE;
        Send(":SOUR:VOLT:MODE FIXED\n");
 }
 Send(":SOUR:VOLT:RANG:AUTO ON\n");
        TempV = ":SOUR:VOLT:LEVEL:IMM ";//Befehlskette des Meßgerät
        TempV = TempV + ssV + "\n";    //Umwandeln des Datenstroms nach
String und anhängen an Variable

        TempC =":SENSE:CURR:PROT ";
        TempC = TempC + ssC + "\n";
     // Temp = Temp + ssI.str();
    if (maxcurrent!=0) Send(TempC);
    Send(":SOUR:VOLT:RANG:AUTO ON\n");
    Send(TempV);
}
```

Quellcode 12: RS232_Keithley2401-SetVoltage-Funktion

Für die Einstellung der Ausgangsspannung des Sourcemeters bietet die Klasse die Methode SetVoltage(). Dabei wird wie folgt vorgegangen.

1. Mit einer if-Anweisung wird überprüft, ob das Gerät bereits auf Voltagemodus eingestellt ist. Wenn dem nicht der Fall ist, wird der Voltagemode durch die SPCI- Befehle (Spannungsquellenfunktion, Festspannungsquellenmodus, automatischer Quellenbereich) eingestellt. Der Befehl zur Konfiguration der Ausgangsspannung wird aus dem QString TempV und den in den QString ssV gewandelten Spannungswert voltage zusammengesetzt und versendet.
2. Es wird durch eine if-Anweisung überprüft, ob die zu konfigurierende Strombegrenzung den Wert 0 enthält. Wenn dies nicht der Fall ist, wird der Befehl, der zuvor aus dem QString TempC und den in den QString ssC gewandelten Stromwert current zusammengesetzt worden ist, versendet.

```
void RS232_Keithley2401::SetCurrent (double current, double maxvoltage)
{
 QString TempV; //lokale Variable für die Spannung
 QString TempC; //lokale Variable für den Strom

 QString ssV = QString::number(maxvoltage);
 QString ssC = QString::number(current);

    if (DeviceStatus != CURRENTMODE) {
        Send(":SOUR:FUNC:MODE CURR\n");
        DeviceStatus = CURRENTMODE;
        Send(":SOUR:CURR:MODE FIXED\n");
        Send(":SOUR:VOLT:LEVEL:IMM 0\n");
        Send(":SOUR:CURR:RANG:AUTO ON\n");
//Stiller: Auto Range funktioniert nicht wie erwartet, ohne Anpassungen des
Sense Range wird die Spannung limitiert
        Send(":SENS:VOLT:RANG 2\n"); // Set Voltage range
}
        TempV = ":SENSE:VOLT:PROT ";   //Befehlskette des Meßgerät
        TempV = TempV + ssV + "\n";    //Umwandeln des Datenstroms nach
                                        String und anhängen an Variable
        TempC = ":SOUR:CURR:LEVEL:IMM ";
        TempC = TempC + ssC + "\n";
    if (maxvoltage!=0)
    Send(TempV);
    Send(":SOUR:CURR:RANG:AUTO ON\n");
    Send(TempC);
}
```

Quellcode 13: RS232_Keithley2401-SetCurrent-Funktion

Die Methode „setCurrent“ wird verwendet, wenn das Sourcemeter als Stromquelle statt als Spannungsquelle verwendet werden soll.

Als Parameter benötigt die Methode den zu konfigurierenden Versorgungsstrom current und die Spannungsbegrenzung maxvoltage. Der Verlauf des Methoden-Codes ist analog zur Methode setVoltage().

```
void RS232_Keithley2401::Measure(double &current, double &voltage, TVolt-
ageUnit VoltUnit, TCurrentUnit CurrUnit)
{
  QString Res = SendAndReceive(":READ?\n");
  QString Tmp;
  if (Res != "Error" && Res != "") {
      Tmp = Res.mid(8,13);
      voltage = Tmp.toFloat();
      Tmp = Res.mid(22,13);
      current = Tmp.toFloat();
  }

  switch (CurrUnit) {
    case AMP:                       break;
    case MILLIAMP: current *= 1e3; break;
    case MICROAMP: current *= 1e6; break;
    case NANOAMP:  current *= 1e9; break;
    default:                        break;
  }

  switch (VoltUnit) {
    case VOLT:                       break;
    case MILLIVOLT: voltage *= 1e3; break;
    case MICROVOLT: voltage *= 1e6; break;
    default:                         break;
  }
}
```

Quellcode 14: RS232_Keithley2401-Measure-Funktion

Im Quellcode 14 ist die Methode zum Messen der Spannung am Sourcemeter und des Stroms, der aus dem Sourcemeter fließt, dargestellt. Als Parameter werden zwei Zeiger auf die Double Variablen current und voltage übergeben. Mit Hilfe der Zeiger können die ausgelesenen Messwerte an die aufrufende Methode zurückgegeben werden. Zunächst werden die beiden Objekte als QString „Res“ und „Tmp“ deklariert. Dann werden mit Hilfe der Methode „SendAndReceive ()“ und des SPCI Befehls „:READ?“ Spannung und Strom zeitgleich ausgelesen.

Als nächstes wird die Antwort des Sourcemeters ausgewertet. Wenn die die Methode SendAndReceive einen Error zurückgegeben hat oder das in Res zurückgegebene Resultat leer ist, wird die Bearbeitung abgebrochen. Andernfalls, werden die Spannungs- und Stromwerte mit der Methode mid() aus dem QString ausgeschnitten und mit der Methode toFloat() in Floats umgewandelt. Die Parameter von mid() entsprechen dabei der Zeichenstelle, an der auszuschneidende Bereich beginnt und der Länge des auszuschneidenden Bereichs. Dieser Vorgang wird sowohl für die Messung bei der Spannung als auch beim Strom durchgeführt.

Nun werden die beiden Methoden „VoltUnit“ und „CurrUnit“ genutzt, um die Einheit der Spannung mit Volt und die Einheit des Stroms mit Amp zu definieren. Die „Switch Case“ Struktur ermöglicht die Auswahl der Einheit beim Messprozess. Es kann zwischen den Einheiten Amp, Milli oder Micro gewählt werden.

```
#include <QCoreApplication>
#include "RS232_Device.h"
#include "RS232_Interface.h"
#include "RS232_Keithley2401.h"

int main(int argc, char *argv[])
{
    double voltage, current;
    QCoreApplication a(argc, argv);

    RS232_Keithley2401 Device("/dev/ttyS0");
    Device.SetVoltage(1.1,0.9); //Erster Paramater von Voltage und
zweiter von Current
    Device.SwitchOn(); // Ausgang einschalten
    Device.Measure(current,voltage);
    qDebug() << "voltage Ausgelesen:" << QString::number(voltage , 'f',
4) << "\n";
    qDebug() << "current Ausgelesen:" << QString::number(current , 'f',
4) << "\n";
```

Quellcode 15: main.cpp

Die Datei „main.cpp“ entspricht letztlich dem Hauptprogramm mit dem die programmierten Klassen getestet werden sollen. Die Variablen „voltage“ und „current“ werden als „double“ deklariert, um empfangene Spannungs- und Stromwerte abzuspeichern. Danach wird das Objekt Device der Klasse RS232_Keithley2401 mit dem Gerätepfad "/dev/ttys0" als Übergabeparameter erstellt.

Es wird die Methode „SetVoltage“ des Objektes Device mit den beiden Parametern 1.1 und 0.9 aufgerufen, wobei dies der einzustellenden Spannung von 1.1V und einer Strombegrenzung von 0,9 A entspricht. Um nun die Spannung am Sourcemeter und den Strom, der durch das Sourcemeter fließt, zu messen, wird die Methode Measure des Objekts Device verwendet. Die ausgelesenen Werte werden in den Variablen curent und voltage gespeichert und anschließend über qDebug() ausgegeben. Bei Tests des Hauptprogramms konnte festgestellt werden, dass die Spannung und die Strombegrenzung korrekt am Sourcemter eingestellt wurden und die eingelesenen Werte für die Spannung am Sourcemeter und dem Strom, der durch das Sourcemeter fließt, mit der Anzeige des Sourcemters übereinstimmt.

6. Fazit

Das Ziel des Projektes war zunächst die Kommunikation zwischen einem Keithley2401 Sourcemeter mit einem Rechner über die RS232-Schnittstelle zu etablieren. Ein vollständiger Programmcode wurde bereits für andere Geräte und die Kommunikation über andere Schnittstellen entwickelt. Dieser Code wurde in der Programmiersprache C++ mit dem Opensource-Programm Qt Creator verfasst.

Um die Kommunikation aufzubauen, musste der Programmcode überprüft und auf die Verwendung der seriellen Schnittstelle angepasst werden. Nachdem die Kommunikation in Betrieb genommen wurde, wurden die Klassen RS232_Interface, RS232_Device und RS232_Keithley2401 für das Keithley2401 Sourcemeter programmiert. Um den korrekten Lösungsweg zu finden, wurden mehrere Tests durchgeführt, teilweise improvisiert und viel ausprobiert. Zum Schluss wurden die neu erstellten Klassen in ein Hautprogramm integriert und getestet. Die Veränderungen und Anpassung der Klassen waren essenziell, damit die RS232-Schnittstelle einwandfrei funktionieren kann.

Alles in allem wurde das Ziel des Projektes erreicht. Anfängliche Schwierigkeiten wurden beseitigt und die Klassen und Methoden sorgfältig erarbeitet und implementiert. Das Gerät kann jetzt für weitere Messungen im Labor eingesetzt werden.

Als weiterer Forschungsausblick wären eine Optimierung und Erweiterung des Programms denkbar. Es könnten weitere Funktionen und Methoden für komplexe Messungen entwickelt und hinzugefügt werden.

7. Danksagung

Ein besonderes Dankeschön möchte ich an Herrn Prof. Dr.-Ing Michael Karagounis richten. Vielen Dank für die Möglichkeit an diesem Projekt zu arbeiten und für Ihren Support und Bereitschaft meine Fragen zu beantworten und mir bei dem Projekt zu helfen.

Außerdem bedanke ich mich bei allen Mitgliedern im Labor für integrierten Schaltungsentwurf für ihren Support und auch bei meiner Familie und allen Freunden, die mich während der Zeit unterstützt haben.

8. Literaturverzeichnis

[1] „Elektronische Messgeräte“, *Tooler*. https://www.tooler.de/messtechnik/elektronische-messgeraete/ (zugegriffen 9. August 2022).

[2] „Bachelor_Thesis_Ömer_Icyer.pdf“.

[3] „2401 - Multifunktionsgerät (SMU), SourceMeter, 1 Kanal, 20V, 1A, 20W“, *Farnell*. https://de.farnell.com/keithley/2401/multifunktionsger-t-smu-20v-1a/dp/2772526 (zugegriffen 30. Mai 2022).

[4] „Keithley 2401 Sourcemeter“, *RS-online*. https://de.rs-online.com/web/p/sourcemeter/7588857 (zugegriffen 5. August 2022).

[5] „RS232_Schnittstelle“, *kompendium.infotip*. https://kompendium.infotip.de/rs-232-die-serielle-schnittstelle.html (zugegriffen 18. Juli 2022).

[6] „Widerstand-Definition und Typen“, *Basic Electronik Tutorials*, 9. Juli 2018. https://www.electronics-tutorials.ws/de/widerstande/widerstands-typen.html (zugegriffen 5. August 2022).

[7] „Definition CentOS“, *golem*. https://www.golem.de/specials/centos/ (zugegriffen 7. August 2022).

[8] K. Lipinski, „SCPI-Befehle (Commands)“, *ITWissen.info*. https://www.itwissen.info/standard-commands-for-programmable-instruments-SCPI.html (zugegriffen 7. August 2022).

[9] „DBL_HM_HM2005_DATENBLATT1_DEUTSCH.pdf“. Zugegriffen: 2. August 2022. [Online]. Verfügbar unter: http://www.pewa.de/DATENBLATT/DBL_HM_HM2005_DATENBLATT1_DEUTSCH.pdf

[10] K. Instruments, „Series 2400 SourceMeter User’s Manual“, S. 496.

[11] „QCoreApplication Class | Qt Core 6.3.2“, *qt.io*. https://doc.qt.io/qt-6/qcoreapplication.html#details (zugegriffen 9. August 2022).

[12] „QSerialPort Class | Qt Serial Port 6.3.1“, *qt.io*. https://doc.qt.io/qt-6/qserialport.html#details (zugegriffen 9. August 2022).

[13] TylerMSFT, „<iostream>“, *docs.microsoft*. https://docs.microsoft.com/de-de/cpp/standard-library/iostream (zugegriffen 9. August 2022).

[14] „QTextStream Class | Qt Core 6.3.2“, *qt.io*. https://doc.qt.io/qt-6/qtextstream.html#details (zugegriffen 9. August 2022).

[15] „QString Class | Qt Core 6.3.2“, *qt.io*. https://doc.qt.io/qt-6/qstring.html#details (zugegriffen 9. August 2022).

[16] „QFile-Klasse | Qt-Core 6.3.2“, *qt.io*. https://doc.qt.io/qt-6/qfile.html#details (zugegriffen 9. August 2022).

[17] „QDebug-Klasse | Qt-Core 6.3.2“, *qt.io*. https://doc.qt.io/qt-6/qdebug.html#details (zugegriffen 9. August 2022).

[18] „Baudraten Grundlagen“. https://www.kunbus.de/baudraten-grundlagen (zugegriffen 19. August 2022).
[19] „Serial Port Overview - MATLAB & Simulink - MathWorks Deutschland“. https://de.mathworks.com/help/instrument/serial-port-overview.html (zugegriffen 19. August 2022).
[20] „IBM Documentation“, 5. April 2022. https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/aix/7.1?topic=communication-flow-control (zugegriffen 19. August 2022).
[21] „What is RTS / CTS Hardware Flow Control? | Brainboxes“. https://www.brainboxes.com/faq/what-is-rts-cts-hardware-flow-control (zugegriffen 19. August 2022).
[22] „Parity_Definition“, *Techopedia.com*. http://www.techopedia.com/definition/1803/parity-check (zugegriffen 19. August 2022).

9. Quellcodeverzeichnis

9. Codeverzeichnis

9.1 main.cpp „Seriell_Test“

```
#include <QCoreApplication>
#include <QSerialPort>
#include <QFile>
#include <QTextStream>
#include <iostream>
#include <sstream>
#include <string>
#include <QDebug>
#include <cstdlib>

using namespace std;
int main(int argc, char *argv[]){
   QCoreApplication a(argc, argv);
   QSerialPort serial ("/dev/ttyS0");
   serial.setPortName("/dev/ttyS0");
   serial.setBaudRate(QSerialPort::Baud9600);
   serial.setDataBits (QSerialPort::Data8);
   serial.setFlowControl (QSerialPort::NoFlowControl);
   serial.setParity(QSerialPort::EvenParity);
   serial.open(QIODevice::ReadWrite);
   serial.clear();

   serial.write (QString("*RST\n").toUtf8());
   serial.write (QString("*IDN?\n").toUtf8());
   serial.waitForBytesWritten(30000);
   serial.waitForReadyRead(30000);
   QByteArray responseData = serial.readAll();
    while (serial.waitForReadyRead(10))
     responseData += serial.readAll();
   QString line = QString::fromUtf8(responseData);
   qDebug() << "Ausgelesen:" <<line;

   std::string TempV;   //lokale Variable für die Spannung
   std::string TempC;
   std::stringstream ssV;
   ssV << 1.0;
   std::stringstream ssC;
   ssC << 1.0;

   serial.write (QString(":SOUR:FUNC:MODE VOLT\n").toUtf8());
   serial.waitForBytesWritten(30000);
   serial.write (QString(":SOUR:VOLT:MODE FIXED\n").toUtf8());
   serial.waitForBytesWritten(30000);
   serial.write (QString(":SOUR:VOLT:RANG:AUTO ON\n").toUtf8());
   serial.waitForBytesWritten(30000);
   serial.write (QString(":SOUR:VOLT:LEV:IMM 0\n").toUtf8());
   serial.waitForBytesWritten(30000);
   serial.write (QString(":SENS:FUNC:CONC ON\n").toUtf8());
   serial.waitForBytesWritten(30000);
   serial.write (QString(":SENS:FUNC:OFF:ALL\n").toUtf8());
   serial.waitForBytesWritten(30000);
   serial.write(QString(":SENS:FUNC:ON 'VOLT:DC','CURR:DC'\n").toUtf8());
   serial.waitForBytesWritten(30000);
```

```
    serial.write (QString(":SENS:AVER:STAT OFF\n").toUtf8());
    serial.waitForBytesWritten(30000);
  // Only Output voltage and current
    serial.write (QString(":FORM:DATA ASCII\n").toUtf8());
    serial.waitForBytesWritten(30000);
    serial.write (QString(":FORM:ELEM VOLT,CURR\n").toUtf8());
    serial.waitForBytesWritten(30000);

    TempC = ":SENSE:CURR:PROT ";
    TempC = TempC + ssC.str()+"\n";
    QString qtempc = QString::fromStdString(TempC);
    TempV = ":SOUR:VOLT:LEVEL:IMM ";//Befehlskette des Meßgerät
    TempV = TempV + ssV.str()+"\n";   //Umwandeln des Datenstroms
    QString qtempv = QString::fromStdString(TempV);

    serial.write (QString(":SOUR:VOLT:RANG:AUTO ON\n").toUtf8());
    serial.waitForBytesWritten(30000);
    serial.write(qtempc.toUtf8());
    serial.waitForBytesWritten(30000);
    serial.write (QString(":SOUR:VOLT:RANG:AUTO ON\n").toUtf8());
    serial.waitForBytesWritten(30000);
    serial.write(qtempv.toUtf8());
    serial.waitForBytesWritten(30000);
    serial.write(QString(":OUTP ON\n").toUtf8());
    serial.waitForBytesWritten(30000);

    serial.write (QString(":READ?\n").toUtf8());
    serial.waitForBytesWritten(30000);
    serial.waitForReadyRead(30000);
    responseData = serial.readAll();
     while (serial.waitForReadyRead(10))
      responseData += serial.readAll();
     line = QString::fromUtf8(responseData);
     qDebug() << "Read Ausgelesen:" <<line;
    serial.close();
    return a.exec();
}
}
```

9.2 RS232_Interface.h

```
#ifndef FHDO_RS232_INTERFACE_H
#define FHDO_RS232_INTERFACE_H
#include <QCoreApplication>
#include <iostream>
#include <string>
#include "QString"
#include <QFile>
#include "qfile.h"
#include <QDebug>
#include <QTextStream>
#include <QSerialPort>

using namespace std;
class RS232_Interface
{
public:
    RS232_Interface(QString DevPath);
    ~RS232_Interface();

    void Send(QString Msg);
    QString SendAndReceive (QString Msg);

protected:
  QSerialPort *serial;
};
#endif // FHDO_RS232_INTERFACE_H
```

9.3 RS232_Interface.cpp

```
#include "RS232_Interface.h"
#include <QCoreApplication>
#include <iostream>
#include <string>
#include "QString"
#include <QFile>
#include "qfile.h"
#include <QDebug>
#include <QTextStream>
#include <vector>
#include <stdio.h>
#include <QSerialPort>

using namespace std;
RS232_Interface::RS232_Interface(QString DevPath){
    QFile Pfad1(DevPath);
        if(Pfad1.exists(DevPath))
        {
            serial = new QSerialPort("/dev/ttyS0");
            serial->setPortName("/dev/ttyS0");
            serial->setBaudRate(QSerialPort::Baud9600);
            serial->setDataBits(QSerialPort::Data8);
            serial->setFlowControl(QSerialPort::NoFlowControl);
```

```
            serial->setParity(QSerialPort::EvenParity);
            if (serial->open(QIODevice::ReadWrite))
            {
                serial->clear();
                serial->write (QString("*RST\n").toUtf8());
            }
            else
            {
                qDebug()<< "Gerät kann nicht geöffnet werden\r\n";
            }
        }
        else
        {
                qDebug()<< "Gerät existiert nicht\r\n";
        }
}
RS232_Interface::~RS232_Interface(){
    if (serial!=NULL)
     serial->close();
}
void RS232_Interface::Send(QString Msg)
//Die Funktion send sendet nur den Befehl an das angeschlossene Gerät
{
                      if (serial!=NULL)
                      {
                          qDebug() << "Will schreiben" << Msg;
                          serial->write(Msg.toUtf8());
                          serial->waitForBytesWritten(30000);
                      }
                      else
                      {
                      qDebug()<< "Gerät kann nicht geöffnet werden\r\n";
                      }
}
QString RS232_Interface::SendAndReceive(QString Msg)
{
                  QString Result;
                  QByteArray responseData;

                     if (serial!=NULL)
                        {
                          qDebug() << "Will schreiben" << Msg;
                          serial->write(Msg.toUtf8());
                          serial->waitForBytesWritten(30000);
                          serial->waitForReadyRead(30000);
                          responseData = serial->readAll();
                          while (serial->waitForReadyRead(10))
                            responseData += serial->readAll();
                          Result = QString::fromUtf8(responseData);
                          qDebug() << "Result: " << Result << "\n";
                         }
                      else
                        {
                      qDebug()<< "Gerät kann nicht geöffnet werden\r\n";
                        }
   return Result;
}
```

9.4 RS232_Device.h

```
#ifndef FHDO_RS232_DEVICE_H
#define FHDO_RS232_DEVICE_H

#include "RS232_Interface.h"
#include <QFile>
#include <QString>

#ifndef VOLT_CUR_UNIT
#define VOLT_CUR_UNIT

enum TVoltageUnit {VOLT, MILLIVOLT, MICROVOLT};
enum TCurrentUnit {AMP,  MILLIAMP,  MICROAMP, NANOAMP};
#endif

using namespace std;

class RS232_Device
{
public:
              ~RS232_Device();

               RS232_Device (QString DevPath); //

      QString           GetName();
      QString              isOk;

protected:
      void         Send (QString Str); // Error checked in Interface
      QString      SendAndReceive (QString Str); // return false if error
      QString           Receive ();

      RS232_Interface   * MyIf;            // My interface
      QString           MyDevPath;         // My Serialnumber of device
      QString           MyName;           // My Name (like'Keithley2401')
};
#endif // FHDO_RS232_DEVICE_H
```

9.5 RS232_Device.cpp

```
#include "RS232_Device.h"
#include <QCoreApplication>
#include <iostream>
#include <string>
#include "QString"
#include <QFile>
#include "qfile.h"
#include <QDebug>
#include <QTextStream>
using namespace std;
RS232_Device::~RS232_Device(){}

RS232_Device::RS232_Device (QString DevPath)
 {
```

```
  MyIf = new RS232_Interface (DevPath);
  MyDevPath = DevPath;
  qDebug()<< "Device with Serialnumber "<<MyDevPath;
  MyName = " Device with Serialnumber " + MyDevPath;
 }
void RS232_Device::Send (QString Str)    // return false if error
{
   MyIf->Send(Str);
}
QString RS232_Device::SendAndReceive (QString Msg)
{
  return MyIf->SendAndReceive(Msg);
}
```

9.6 RS232_Keithley2401.h

```
#ifndef FHDO_RS232_KEITHLEY2401_H
#define FHDO_RS232_KEITHLEY2401_H

#include "RS232_Interface.h"
#include "RS232_Device.h"
#include <stdio.h>
#include <string>
#include <QString>
#include <cstdlib>

using namespace std;
#ifndef KEITHLEY
#define KEITHLEY
enum TDeviceStatus {UNKNOWN, VOLTAGEMODE, CURRENTMODE};
enum TDeviceSpeed  {FAST, MEDIUM, SLOW};
enum TDeviceRange  {AUTO};
#endif // KEITHLEY

class RS232_Keithley2401: public RS232_Device   //Vererbung Device
{
public:
          RS232_Keithley2401 (QString DevPath);  //RS232_ID_K2401
          ~RS232_Keithley2401();
   void   SetCurrentRange1mA (void);
   void   SetVoltage         (double voltage, double maxcurrent=0);
   void   SetCurrent         (double current, double maxvoltage=0);
   void   Measure            (double &current, double &voltage, TVolta-
geUnit VoltUnit = VOLT, TCurrentUnit CurrUnit = AMP);
   double   VSweep           (double &delay, double &steps, double
&start, double &end, double &limit, double &vrange, double &crange);
   double   CSweep           (double &delay, double &steps, double
&start, double &end, double &limit, double &vrange, double &crange);

   void   SwitchOff();
   void   SwitchOn();
   void   Beep();

   void   SetSpeed           (TDeviceSpeed Speed);
   void   DisplayText        (QString Msg);
```

```
    void    DisplayOff          (void);

private:
    bool    DeviceResponds      ();       // Test if device responds
    TDeviceStatus DeviceStatus;           // Remember setting of Multimeter
};
#endif // FHDO_RS232_KEITHLEY2401_H
```

9.7 RS232_Keithley2401.cpp

```
#ifdef USE_GLOBALS
    #include "Globals.h"
#else
    #ifdef USE_INVISIBLE_GLOBALS
        #include <Globals.h>
    #endif
#endif

#include "RS232_Keithley2401.h"

using namespace std;

RS232_Keithley2401::RS232_Keithley2401 (QString DevPath): RS232_Device
(DevPath)
{
  MyName    = "Keithley Multimeter Type 2401 with ID = " + MyDevPath;
  if (!DeviceResponds()) {
        isOk = "false";
    MyIf = NULL;
    return;
  }

// Initilize Source and set to 0V, 0.1mA limit
    Send(":SOUR:FUNC:MODE VOLT\n");
    DeviceStatus = VOLTAGEMODE;
    Send(":SOUR:VOLT:MODE FIXED\n");
    Send(":SOUR:VOLT:RANG:AUTO ON\n");
    Send(":SOUR:VOLT:LEV:IMM 0\n");

// Measure only current and voltage
  Send(":SENS:FUNC:CONC ON\n");
  Send(":SENS:FUNC:OFF:ALL\n");
  Send(":SENS:FUNC:ON  'VOLT:DC','CURR:DC'\n");
  Send(":SENS:AVER:STAT OFF\n");

// Only Output voltage and current
  Send(":FORM:DATA ASCII\n");
  Send(":FORM:ELEM VOLT,CURR\n");

// Output on
 Send(":OUTP ON\n");

}
//---------------------------------------------------------------------------
```

```
RS232_Keithley2401::~RS232_Keithley2401()
{
  SwitchOff();
}
//---------------------------------------------------------------------------
bool RS232_Keithley2401::DeviceResponds()
{
    QString GoodResult("KEITHLEY INSTRUMENTS INC.,MODEL 2401");
    QString Result = SendAndReceive("*IDN?\n");
    Result.resize(GoodResult.length());
    return(Result == GoodResult);
}
//---------------------------------------------------------------------------
void RS232_Keithley2401::SetVoltage (double voltage, double maxcurrent)
{
 QString TempV;//lokale Variable für die Spannung
 QString TempC;//lokale Variable für den Strom
 QString ssV = QString::number(voltage);
 QString ssC = QString::number(maxcurrent);

 if (DeviceStatus != VOLTAGEMODE) {
        Send(":SOUR:FUNC:MODE VOLT\n");
        DeviceStatus = VOLTAGEMODE;
        Send(":SOUR:VOLT:MODE FIXED\n");
 }
 Send(":SOUR:VOLT:RANG:AUTO ON\n");
        TempV = ":SOUR:VOLT:LEVEL:IMM ";//Befehlskette des Meßgerät
        TempV = TempV + ssV + "\n";    //Umwandeln des Datenstroms nach
                                        String und anhängen an Variable

        TempC =":SENSE:CURR:PROT ";
        TempC = TempC + ssC + "\n";
     // Temp = Temp + ssI.str();
    if (maxcurrent!=0) Send(TempC);
    Send(":SOUR:VOLT:RANG:AUTO ON\n");
    Send(TempV);
}
//---------------------------------------------------------------------------
void RS232_Keithley2401::SetCurrent (double current, double maxvoltage)
{
 QString TempV;//lokale Variable für die Spannung
 QString TempC;//lokale Variable für den Strom

 QString ssV = QString::number(maxvoltage);
 QString ssC = QString::number(current);

    if (DeviceStatus != CURRENTMODE) {
        Send(":SOUR:FUNC:MODE CURR\n");
        DeviceStatus = CURRENTMODE;
        Send(":SOUR:CURR:MODE FIXED\n");
        Send(":SOUR:VOLT:LEVEL:IMM 0\n");
        Send(":SOUR:CURR:RANG:AUTO ON\n");
//Stiller: Auto Range funktioniert nicht wie erwartet, ohne Anpassungen
des Sense Range wird die Spannung limitiert
        Send(":SENS:VOLT:RANG 2\n"); // Set Voltage range
}
        TempV = ":SENSE:VOLT:PROT ";  //Befehlskette des Meßgerät
```

```
        TempV = TempV + ssV + "\n"; //Umwandeln des Datenstroms nach
                                    String und anhängen an Variable
        TempC = ":SOUR:CURR:LEVEL:IMM ";
        TempC = TempC + ssC + "\n";

    if (maxvoltage!=0) Send(TempV);
    Send(":SOUR:CURR:RANG:AUTO ON\n");
    Send(TempC);
}
//---------------------------------------------------------------------------
void RS232_Keithley2401::Measure(double &current, double &voltage, TVolt-
ageUnit VoltUnit, TCurrentUnit CurrUnit)
{
  QString Res = SendAndReceive(":READ?\n");
  QString Tmp;
  if (Res != "Error" && Res != "") {
      Tmp = Res.mid(8,13);
      voltage = Tmp.toFloat();
      Tmp = Res.mid(22,13);
      current = Tmp.toFloat();
  }

  switch (CurrUnit) {
    case AMP:                       break;
    case MILLIAMP: current *= 1e3; break;
    case MICROAMP: current *= 1e6; break;
    case NANOAMP:  current *= 1e9; break;
    default:                        break;
  }

  switch (VoltUnit) {
    case VOLT:                       break;
    case MILLIVOLT: voltage *= 1e3; break;
    case MICROVOLT: voltage *= 1e6; break;
    default:                         break;
  }
}
//---------------------------------------------------------------------------
void RS232_Keithley2401::SetSpeed (TDeviceSpeed Speed)
{
  switch (Speed) {
    case FAST:
      Send (":SENSE:VOLTAGE:NPLC 0.1\n");
      Send (":SENSE:CURRENT:NPLC 0.1\n");
    break;
    case MEDIUM:
      Send (":SENSE:VOLTAGE:NPLC 1\n");
      Send (":SENSE:CURRENT:NPLC 1\n");
    break;
    case SLOW:
      Send (":SENSE:VOLTAGE:NPLC 10\n");
      Send (":SENSE:CURRENT:NPLC 10\n");
    break;
    default: break;
  }
}
//---------------------------------------------------------------------------
```

```
void RS232_Keithley2401::DisplayText (QString Msg)
{
  const int MAXLEN = 12;
  if (Msg.length() > MAXLEN) Msg = Msg.mid(0,MAXLEN);

  Send(":DISPLAY:TEXT:DATA \"\n" + Msg + "\"");
  Send(":DISPLAY:TEXT:STATE ON\n");

}
//---------------------------------------------------------------------------
void RS232_Keithley2401::DisplayOff (void)
{
  Send(":DISPLAY:TEXT:STATE OFF\n");
}
//---------------------------------------------------------------------------
void RS232_Keithley2401::SwitchOff (void)
{
  Send(":OUTP OFF\n");
}
//---------------------------------------------------------------------------
void RS232_Keithley2401::SwitchOn (void)
{
  Send(":OUTP ON\n");
}
//---------------------------------------------------------------------------
void RS232_Keithley2401::Beep (void)
{
  Send(":System:Beeper 500, 2\n");
  Send(":System:Beeper 20000, 2\n");
  Send(":System:Beeper 500, 2\n");
}
//---------------------------------------------------------------------------
```

9.8 main.cpp

```
#include <QCoreApplication>
#include "RS232_Device.h"
#include "RS232_Interface.h"
#include "RS232_Keithley2401.h"
int main(int argc, char *argv[])
{
    double voltage, current;
    QCoreApplication a(argc, argv);

    RS232_Keithley2401 Device("/dev/ttyS0");
    Device.SetVoltage(1.1,0.9);
    Device.SwitchOn();
    Device.Measure(current,voltage);
    qDebug() << "voltage Ausgelesen:" << QString::number(voltage , 'f',
4) << "\n";
    qDebug() << "current Ausgelesen:" << QString::number(current , 'f',
4) << "\n";
    return a.exec();
}
 }
```

Eidesstattliche Versicherung

Ich, Abdallah Battai, hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
Alle Quellenangaben und Zitate sind richtig und vollständig und in den jeweiligen Kapiteln und im Literaturverzeichnis wiedergegeben.
Mir ist bekannt, dass falsche Angaben im Zusammenhang mit dieser Erklärung strafrechtlich verfolgt werden können.

Ort, Datum: Dortmund, den 15.08.2022 **Unterschrift:**