Fachhochschule
Dortmund
University of Applied Sciences and Arts

Auslesung und Steuerung eines Oszilloskops mit SCPI-Befehlen über Ethernet unter Verwendung der LXI-Bibliothek

Bachelorarbeit

im Studiengang „Elektrotechnik“

vorgelegt von

Matekeu Fokwa Oriane

am 20.10.2023
an der Fachhochschule Dortmund

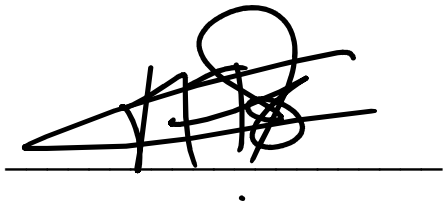
Erstprüfer: Prof Dr. Ing Michael Karagounis

Zweitprüfer: Ing Alexander Walsemann

Selbständigkeitserklärung:

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Literatur und Hilfsmittel angefertigt habe. Wörtlich übernommene Sätze und Satzteile sind als Zitate belegt, andere Anlehnungen hinsichtlich Aussage und Umfang unter Quellenangabe kenntlich gemacht.

Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und ist auch noch nicht veröffentlicht.



Dortmund, 20.10.2023

Ort, Datum

Unterschrift

Abstract

Deutsch

Die Fernsteuerung und Datenerfassung von Oszilloskopen über Ethernet ist ein relevantes Thema für die Optimierung von Messaufbauten. Das Ziel dieses Projekts besteht darin, eine benutzerfreundliche Anwendung zu entwickeln, die es ermöglicht, ein Oszilloskop über Ethernet anzusteuern und Messdaten abzurufen. Um dieses Ziel zu erreichen, sind mehrere Schritte erforderlich. Zunächst wird eine virtuelle Maschine mit der Linux-Distribution Ubuntu eingerichtet. Anschließend wird die Entwicklungsumgebung Qt Creator installiert. Weiterhin wird die Bibliothek LXI (LAN eXtensions for Instrumentation) [4] installiert. Schließlich werden die Programmiersprache C und die SPCI (Standard Commands for Programmable Instruments [10]) -Befehlsdefinitionen in Qt Creator verwendet, um die gewünschte Aufgabe auszuführen. Der entwickelte Code wird getestet, indem das Oszilloskop über Ethernet mit der virtuellen Maschine verbunden wird. Am Ende dieses Projekts wird es möglich sein, verschiedene Daten eines Referenzsignals automatisch und aus der Ferne zu messen. Dies ermöglicht die flexible Fernsteuerung und Datenabfrage von Oszilloskopen über ein Netzwerk, wodurch eine effiziente Erfassung und Analyse von Messdaten ermöglicht wird.

Englisch

Remote control and data acquisition of oscilloscopes via Ethernet is a relevant topic for the optimization of measurement setups. The goal of this project is to develop a user-friendly application that allows to control an oscilloscope over Ethernet and to retrieve measurement data. To achieve this goal, several steps are required. First, a virtual machine is set up with the Linux distribution Ubuntu. Then the development environment Qt Creator is installed. Furthermore, the library LXI (LAN eXtensions for Instrumentation) [4] is installed. Finally, the C programming language and the SPCI (Standard Commands for Programmable Instruments [10]) command definitions in Qt Creator are used to perform the desired task. The developed code will be tested by connecting the oscilloscope to the virtual machine via Ethernet. At the end of this project, it will be possible to measure various data of a reference signal automatically and remotely. This will allow flexible remote control and data retrieval of oscilloscopes over a network, enabling efficient acquisition and analysis of measurement data.

Inhaltsverzeichnis

1 Einleitung

In der Ära der fortschreitenden Digitalisierung gewinnt die präzise Messung und Analyse elektronischer Signale immer mehr an Bedeutung. Die Fähigkeit, elektronische Signale exakt zu erfassen und zu interpretieren, ist für verschiedene Fachbereiche von großer Relevanz, darunter Ingenieurwesen, Wissenschaft und viele Industriezweige. Dabei nehmen Oszilloskope eine bedeutende Rolle ein, da sie eine effektive Methode bieten, um elektrische Signale grafisch darzustellen und detaillierte Analysen durchzuführen. Insbesondere die Fernsteuerung von Oszilloskopen über Ethernet wird aufgrund der zunehmenden Verbreitung von Ethernet-Netzwerken und der Standardisierung von LXI für die Steuerung von Messgeräten über Netzwerke immer beliebter. In dieser Arbeit liegt der Fokus darauf, ein Oszilloskop mithilfe der Programmierumgebung Qt Creator und der LXI-Bibliothek über Ethernet auszulesen. Es werden die Grundlagen der Oszilloskope und der Ethernet-Kommunikation eingeführt, die Implementierung der SCPI-Befehle in Qt Creator unter Verwendung der Programmiersprache C behandelt und abschließend die Frequenz, den positiven und negativen Duty Cycle, den Effektivwert, den Mittelwert eines Rechtecksignals zu messen. Sowie die Fast-Fourier-Transformation Analyse durchzuführen. Durch diese Arbeit wird es ermöglicht, Oszilloskope flexibel und effizient über ein Netzwerk fernzusteuern und Daten abzufragen, was die Datenerfassung und -analyse erheblich verbessert.

1.1 Motivation und Zielsetzung

Die Motivation für die Verwendung eines Computers zur Steuerung und Automatisierung von Oszilloskopen über Ethernet liegt in der Verbesserung der Benutzerfreundlichkeit und der Möglichkeit zur umfassenden Datenerfassung und -verarbeitung. Durch die Verbindung des Oszilloskops mit einem Computer über Ethernet kann der Benutzer das Oszilloskop von seinem Arbeitsplatz aus fernsteuern, Messparameter einstellen und Messungen automatisieren. Der Computer fungiert als Schnittstelle zwischen Benutzer und Messgerät, um die Verbindung herzustellen, die Kommunikation zu ermöglichen und die Steuerbefehle an das Oszilloskop zu senden. Die Automatisierungsfunktionen, die über die Ethernet-Verbindung realisiert werden, ermöglichen eine effiziente Datenerfassung und -verarbeitung. Der Computer kann Messungen automatisch in festgelegten Intervallen durchführen, mehrere Messungen kombinieren oder komplexe Messabläufe automatisieren. Die erfassten Messdaten können direkt auf dem Computer gespeichert, analysiert und weiterverarbeitet werden. Das Hauptziel dieser Arbeit besteht darin, eine Lösung zu entwickeln, die es ermöglicht, ein Oszilloskop über Ethernet mit einem Computer zu verbinden, um die Steuerung und Automatisierung von Messungen zu ermöglichen. Durch die Entwicklung einer benutzerfreundlichen Software wird der Prozess der Steuerung und Datenerfassung erleichtert und die Effizienz der Messungen verbessert. Dadurch wird die Flexibilität und Leistungsfähigkeit des Oszilloskops erweitert, indem die Möglichkeiten der Fernsteuerung und Automatisierung über Ethernet genutzt werden. Dies kann Ingenieuren, Wissenschaftlern und anderen Fachleuten eine effizientere Durchführung von Messungen, eine einfachere Erfassung von Daten und eine detaillierte Analyse für ihre Forschung und Entwicklungsprojekte ermöglichen.

1.2 Überblick auf die Arbeit

In dieser Arbeit wird ein umfassender Überblick auf die Entwicklung einer Lösung zur Fernsteuerung von Oszilloskopen über Ethernet gegeben. Der Fokus liegt dabei auf der Verwendung der Programmierumgebung Qt Creator und der LXI-Bibliothek zur Umsetzung einer benutzerfreundlichen Steuerungssoftware. Zunächst wird eine Einführung in die Grundlagen der Oszilloskope und deren Bedienung gegeben. Dabei werden wichtige Konzepte und Funktionen erläutert, um ein grundlegendes Verständnis für die Arbeitsweise von Oszilloskopen zu schaffen. Anschließend wird die Ethernet-Kommunikation eingeführt und erläutert, wie Oszilloskope über eine Netzwerkverbindung gesteuert werden können. Die LXI-Standardisierung spielt hierbei eine wichtige Rolle und wird näher erläutert. Im nächsten Schritt wird die Programmierumgebung Qt Creator vorgestellt und ihre Vorteile für die Entwicklung von Benutzeroberflächen erläutert. Es werden grundlegende Kenntnisse über die Nutzung von Qt Creator vermittelt, um eine solide Grundlage für die Umsetzung der Steuerungssoftware zu schaffen. Darauf aufbauend wird die Integration der LXI-Bibliothek in Qt Creator behandelt. Es wird erklärt, wie die Bibliothek genutzt werden kann, um die Kommunikation mit dem Oszilloskop über Ethernet herzustellen und Steuerbefehle zu senden. Im weiteren Verlauf der Arbeit wird die Implementierung der SCPI-Befehle (Standard Commands for Programmable Instruments [10]) in Qt Creator mit Hilfe der Programmiersprache C behandelt. Diese Befehle ermöglichen die Steuerung und Abfrage von Funktionen des Oszilloskops über die Ethernet-Verbindung. Abschließend wird die entwickelte Steuerungssoftware getestet und validiert. Es werden verschiedene Testszenarien durchgeführt, um die Funktionalität und Zuverlässigkeit der Lösung zu überprüfen. Die Messdaten werden erfasst, visualisiert und analysiert, um die Effektivität der Fernsteuerung und Datenerfassung über Ethernet zu bewerten. Insgesamt liefert diese Arbeit einen umfassenden Überblick auf die Entwicklung einer Lösung zur Fernsteuerung von Oszilloskopen über Ethernet. Es werden wichtige Aspekte der Implementierung behandelt und die erzielten Ergebnisse präsentiert. Die Arbeit bietet somit einen wertvollen Beitrag zur Verbesserung der Steuerung und Automatisierung von Oszilloskopen in verschiedenen Anwendungsbereichen.

2 Grundlagen

Im Rahmen dieser Arbeit wird ein Oszilloskop R&S®RTB2004 über Ethernet ausgelesen. Hierfür wird die Programmierumgebung Qt Creator verwendet, welche die Integration der LXI-Bibliothek ermöglicht. Um das Ziel zu erreichen, wird zunächst die Funktionsweise des Oszilloskops R&S®RTB2004 vorgestellt. Anschließend werden die Grundlagen der Ethernet-Kommunikation und der Bibliothek LXI erläutert, einschließlich der Verwendung von SCPI-Befehlen. Daraufhin erfolgt die Vorstellung der Programmierumgebung Qt Creator und der C-Programmierung.

2.1 Oszilloskope und ihre Funktionsweise

Das Oszilloskop ist neben dem Multimeter das verbreitetste und wichtigste Mess- und Diagnosegerät in der Elektronik und Elektrotechnik. [27] Sie ermöglichen die Darstellung elektrischer Signale wie Spannungen oder Ströme und werden daher häufig für Fehlerbehebung und Signalanalyse eingesetzt. Die Funktionsweise eines Oszilloskops basiert auf der Abtastung des elektrischen Signals und der Darstellung der Messwerte auf einem Bildschirm [1]. Das Signal wird von einer Sonde erfasst und in ein elektrisches Signal umgewandelt, das dann vom Oszilloskop verarbeitet werden kann. Moderne Oszilloskope bieten eine Vielzahl von Funktionen wie automatische Messungen, FFT-Analyse und digitale Signalverarbeitung. Sie können auch hohe Abtastraten und Bandbreiten erreichen, um schnelle Signale und hochfrequente Systeme zu messen [1]. In diesem Projekt wird das Oszilloskop R&S®RTB2004 über Ethernet ausgelesen. Das R&S RTB2004 Oszilloskop ist für die Messung von Stromkreisen ausgelegt, die indirekt oder gar nicht mit dem Stromnetz verbunden sind. Es wird in Industrie-, Verwaltungs- und Laborumgebungen für die Entwicklung, Produktion und Prüfung elektronischer Bauteile und Geräte verwendet. Das R&S RTB2004 digitale Oszilloskop arbeitet durch Abtastung und Darstellung elektrischer Signale, um genaue Analyse und Darstellung der Signalverläufe zu ermöglichen [2]. Das R&S RTB2004 erfasst das Signal und wandelt es in digitale Abtastwerte um. Die digitalen Abtastwerte werden gemäß den Erfassungseinstellungen verarbeitet.

Das Ergebnis ist eine Messkurvenaufzeichnung, die auf dem Bildschirm angezeigt und im Speicher abgelegt wird [2]. Das Oszilloskop verfügt über ein hochauflösendes Farbdisplay, auf dem die erfassten Signale in Echtzeit dargestellt werden. Es bietet eine klare und detaillierte Anzeige von Signalformen, Amplituden, Zeitachsen und anderen Messparametern [2]. Das Oszilloskop bietet verschiedene Triggeroptionen, um bestimmte Ereignisse im Signalverlauf zu erfassen. Es können Triggerbedingungen wie Flanken, Pulsbreiten, Impulse und Muster festgelegt werden, um relevante Signalereignisse hervorzuheben [2]. Das R&S RTB2004 ermöglicht eine Vielzahl von Messungen, einschließlich Effektivwertmessungen, Mittelwertmessungen, Frequenzmessungen, Duty-Cycle bzw. Abtastgrad und mehr. Es bietet automatische Messfunktionen, welche die Auswertung von Signalparametern erleichtern. Damit können bis zu 6 verschiedene Messungen gleichzeitig durchgeführt werden [2]. Das Oszilloskop unterstützt erweiterte Analysefunktionen wie Fast Fourier Transformation (FFT), um Frequenzspektren von Signalen zu analysieren [2]. Das R&S RTB2004 verfügt über verschiedene Anschlussmöglichkeiten wie Ethernet, USB und GPIB, um das Oszilloskop mit externen Geräten und Netzwerken zu verbinden. Dies ermöglicht die Datenübertragung, Fernsteuerung und Integration in bestehende Messsysteme [2]. Die Abbildungen 1 und 2 stellen die Frontplatte und die Rückseitenansicht des R&S RTB2004 Oszilloskops dar.

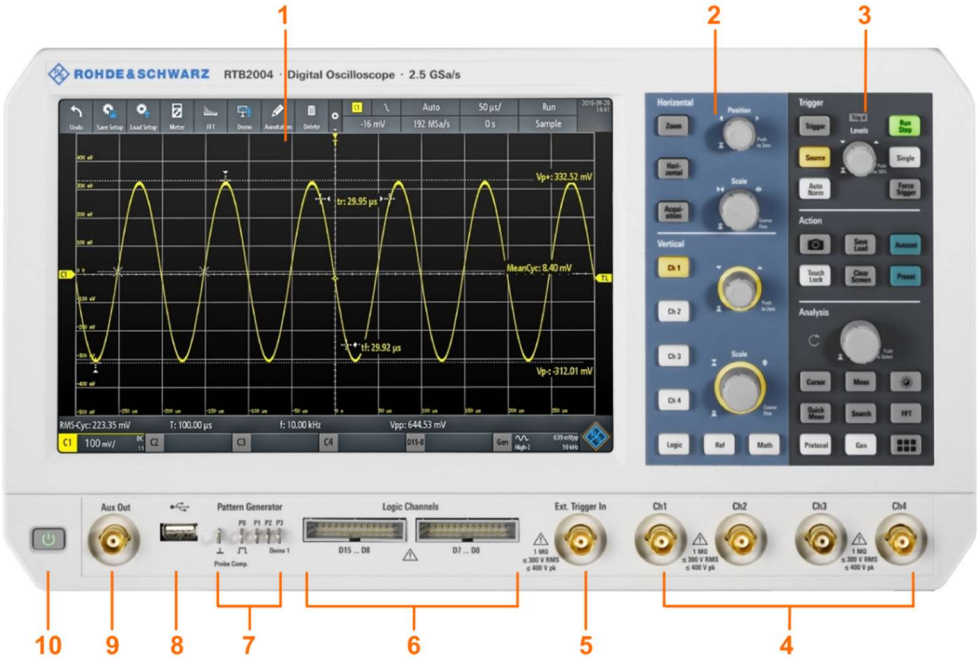


Abbildung 1: Frontplatte des R&S RTB2004 mit 4 Eingangskanälen [2].

1 = Display

2 = Bedienelemente für horizontale und vertikale Einstellungen

3 = Bedienelemente für Triggereinstellungen, Aktion und Analyse

4 = Analoge Eingangskanäle (2 Kanäle bei R&S RTB2002, 4 Kanäle bei R&S RTB2004)

5 = Externer Triggereingang

6 = Logiktastkopfanschlüsse (Option R&S RTB-B1)

7 = Anschlüsse für Tastkopfkompensation und optionalen Mustergenerator (R&S RTB-B6)

8 = USB-Anschluss

9 = Anschluss Aux Out

10 = [Standby] Taste

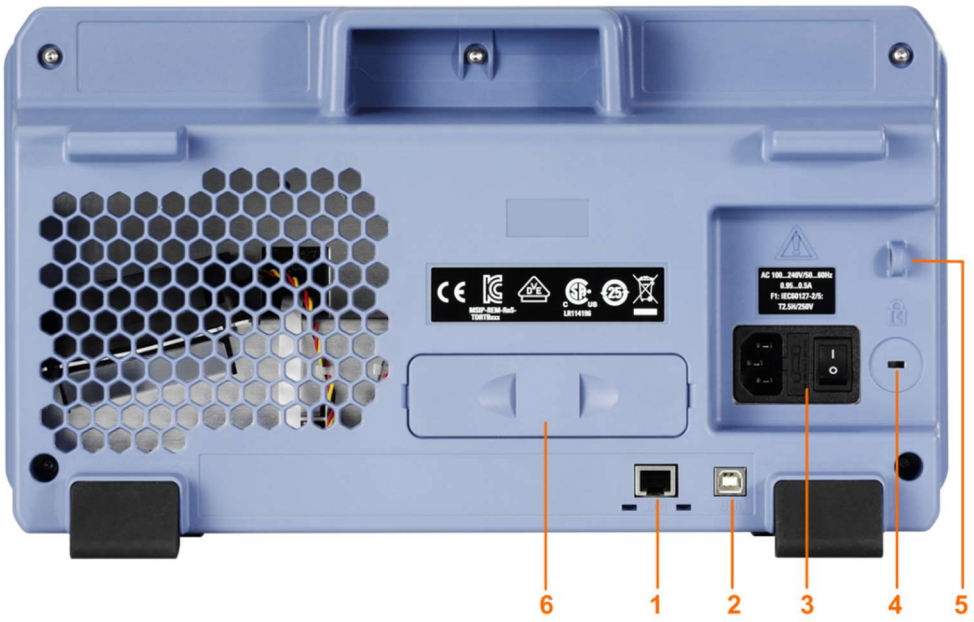


Abbildung 2: Rückseitenansicht des R&S RTB2004 [2].

1 = LAN-Anschluss

2 = USB-Anschluss, Typ B

3 = Anschluss für Wechselstromversorgung und Hauptnetzschalter

4 = Kensington-Schloss zum Sichern des Geräts gegen Diebstahl

5 = Ring für Schloss zum Sichern des Geräts gegen Diebstahl

6 = Nicht belegt

2.2 Ethernet und LXI-Bibliothek

Ethernet ist eine Netzwerktechnologie, die Daten zwischen verschiedenen Geräten in einem geschlossenen Netzwerk über Kabel überträgt [3]. Die Übertragung geschieht via Ethernet-Kabel (vgl. Abb. 3). Es handelt sich um ein Standardprotokoll, das die Kommunikation zwischen Geräten über ein Netzwerk ermöglicht. Die LXI-Bibliothek ist eine Sammlung von Software-Tools und -Bibliotheken, die zur Steuerung von LXI-Geräten über Ethernet-Verbindungen verwendet werden können. LXI (Lan eXtensions for Instrumentation) ist ein Standardprotokoll, das speziell für die Fernsteuerung von Messgeräten wie Oszilloskopen entwickelt wurde. Der LXI-Standard definiert die Kommunikationsprotokolle für moderne Instrumentierungs- und Datenerfassungssysteme, die Ethernet verwenden [4]. Durch die Integration von Ethernet und LXI in die Programmierumgebung Qt Creator können Entwickler eine einfachere und effizientere Möglichkeit zur Steuerung von Oszilloskopen implementieren. Dies ist besonders nützlich für Anwendungen, bei denen Messdaten in Echtzeit gesammelt und analysiert werden müssen. Durch die Integration von Ethernet und LXI in die Qt Creator-Entwicklungsumgebung können Entwickler auch auf eine Vielzahl von Bibliotheken, APIs (Application Programming Interfaces oder Programmierschnittstellen [11]) und Funktionen zugreifen, die speziell für die Entwicklung von Anwendungen für die Fernsteuerung von Oszilloskopen entwickelt wurden. Insgesamt bieten Ethernet und LXI eine leistungsstarke Kombination für die Fernsteuerung von Oszilloskopen und

ermöglichen es Entwicklern, eine schnelle und effektive Möglichkeit zur Steuerung dieser Geräte zu implementieren.

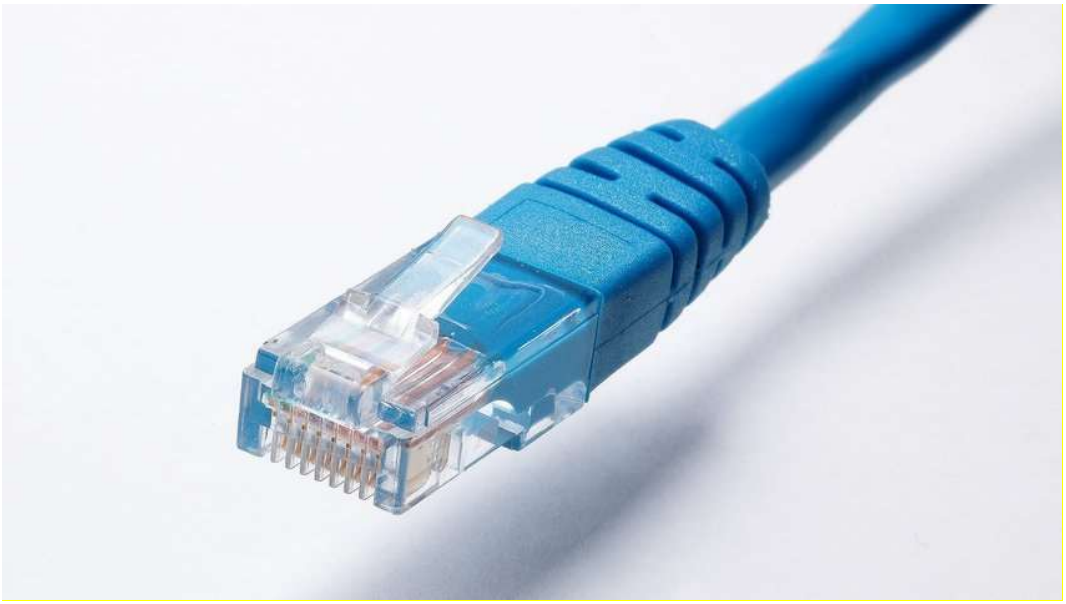


Abbildung 3: Ein typisches LAN-Kabel für Ethernet Kommunikation [5].

2.3 SCPI-Befehle und deren Bedeutung

Die Steuerung des Oszilloskops erfolgt über Befehle im Format des SCPI-Standards. Das Ethernet-Protokoll macht die Übertragung von SCPI-Befehlen vom Hostcomputer zum Oszilloskop möglich. SCPI-Befehle bestehen [2] aus einem Header und in den meisten Fällen einem oder mehreren Parametern. Die Header können aus mehreren mnemonischen Codes (Schlüsselwörter) zusammengesetzt sein. Abfragebefehle werden gebildet, indem ein Fragezeichen direkt an den Header angehängt wird. Die Befehle können entweder gerätespezifisch oder geräteunabhängig (Common Commands) sein. [2] Common Commands und gerätespezifische Befehle unterscheiden sich durch ihre Syntax [2]. Common Commands (geräteunabhängige Befehle) bestehen aus einem Header, dem ein Stern (*) vorangestellt ist und gegebenenfalls einem oder mehreren Parametern. In der Tabelle 1 sind einige Beispiele für Common Commands dargestellt. Die Syntax für gerätespezifische Befehle basiert oft auf einer Kombination von Buchstaben, Zahlen, Sonderzeichen und Parametern. Eine Beispiel Befehl ist

DISPlay [: WINDow<1...4>]: MAXimize <Boolescher Ausdruck> [2]. Mit diesem Befehl kann die Fensteranzeige des R&S RTB2000 Oszilloskops gesteuert und angepasst, um die gewünschte Darstellung oder Ansicht zu erreichen.

*RST	RESET	Setzt das Gerät zurück.
*ESE	EVENT STATUS ENABLE	Setzt die Bits des Event-Status-Enable-Registers.
*ESR?	EVENT STATUS QUERY	Fragt den Inhalt des Event-Status-Registers ab.
*IDN?	EVENT STATUS QUERY	Fragt den Identifikationsstring des Geräts ab

Tabelle 1: Beispiele für Common Commands [2]

2.4 Entwicklungsumgebung Qt Creator und Programmierung in C

Die Entwicklungsumgebung Qt Creator bietet eine intuitive grafische Benutzeroberfläche und ist eine der am häufigsten verwendeten Entwicklungsplattformen für die Entwicklung von betriebssystemunabhängingen Desktopanwendungen. Qt erfreut sich besonderer Beliebtheit unter Linux- und Unix-Systemen. C ist eine grundlegende Programmiersprache, die in vielen Anwendungen und Systemen eingesetzt wird. Qt Creator ermöglicht Entwicklern, Anwendungen zu erstellen und zu bearbeiten. Mit C als Programmiersprache ist es möglich, Code zu schreiben, um spezifische Aufgaben auszuführen. Um das Oszilloskop über Ethernet in diesem Projekt auszulesen, ist es erforderlich, C-Programmierkenntnisse zu besitzen und diese in der Qt Creator-Entwicklungsumgebung anzuwenden. Dies kann durch die Integration der LXI-Bibliothek erreicht werden, die eine benutzerfreundliche API (Application Programming Interface) [28] für die Steuerung von Netzwerkgeräten bietet. Eine benutzerfreundliche API ist eine Bibliothek mit Funktionen, die darauf ausgelegt

ist, die Programmierung so einfach und intuitiv wie möglich zu gestalten. Sie soll es den Programmierern ermöglichen, die Funktionen und Leistungen der Software effizient und ohne unnötige Komplexität zu nutzen. Die Verwendung der LXI-Bibliothek ermöglicht es dem Entwickler, einfache Befehle wie das Abrufen von Geräteinformationen und das Senden von SCPI-Befehlen an das Oszilloskop über Ethernet auszuführen. Die Integration der LXI-Bibliothek in die Qt Creator-Entwicklungsumgebung ist relativ einfach und erfordert das Hinzufügen der Bibliotheksdateien zur Projektdatei und die Anpassung der Kompilierungseinstellung. Sobald die Bibliothek integriert ist, können Entwickler die umfangreiche Dokumentation und die Beispiele nutzen, um schnell und effektiv mit dem Oszilloskop zu interagieren. Qt Creator und C-Programmierung bieten eine gute Grundlage für die Verwendung der LXI-Bibliothek und die Steuerung von Oszilloskopen über Ethernet. In einer typischen C-Programmdateistruktur finden sich in der Regel die folgenden Dateiengruppen: Header-Dateien haben die Erweiterung .h und enthalten die Deklarationen von Funktionen, Strukturen, Konstanten und Typdefinitionen, die in den Quelldateien verwendet werden. Header-Dateien dienen als Schnittstelle zwischen verschiedenen Quelldateien und ermöglichen es, den Code zu modularisieren und die Wiederverwendbarkeit zu verbessern. Quelldateien haben die Erweiterung .c und enthalten den eigentlichen Code, der die Funktionalität des Programms implementiert. Hier werden Funktionen definiert und die Logik des Programms umgesetzt. Normalerweise gibt es eine Haupt-Quelldatei (main.c), die den Einstiegspunkt des Programms enthält und die Ausführung startet. Weitere Quelldateien können verwendet werden, um verschiedene Funktionen oder Module zu implementieren, die im Hauptprogramm verwendet werden.

3 Vorbereitung

3.1 Installation einer Virtual Maschine mit Ubuntu

Eine virtuelle Maschine (VM) ist ein Computer, der vollständig softwarebasiert und nicht hardwarebasiert arbeitet. [9] Virtuelle Maschinen verwenden Software auf einem physischen (Host-) Computer, um die Funktionen eines anderen Computers oder Betriebssystems nachzubilden oder zu emulieren. Zusammengefasst ist eine VM ein simulierter Computer innerhalb eines realen Computers [9]. Virtuelle

Maschinen verwenden eine spezielle Software, die Virtual Maschine Monitor oder Manager (VMM) genannt wird, um die Hauptkomponenten und Hardwareressourcen eines Host-Computers zu emulieren. Die VMM arbeitet als Vermittler zwischen dem physischen Host-Computer und der virtuellen Gastmaschine und verteilt die Ressourcen auf der Grundlage der individuellen Anforderungen und der Host-Kapazität an die VM. Wie jede andere App läuft auch eine VM in einem Fenster und es können mehrere VMs gleichzeitig ausgeführt werden. So kann zum Beispiel neben dem Betriebssystem des Host-Computers gleichzeitig eine virtuelle Maschine mit Android und eine virtuelle Maschine mit Linux ausgeführt werden [9]. Um die virtuelle Umgebung für die Auslesung des Oszilloskops über Ethernet einzurichten, wird die Virtualisierungssoftware Oracle VirtualBox benötigt. Mit dieser Software kann die virtuelle Maschine erstellt und ausführt werden. Der erste Schritt der Installation einer Virtual Maschine mit Ubuntu besteht darin, das Ubuntu ISO-Image von der offiziellen Ubuntu-Website herunterzuladen. Es wird empfohlen, die LTS (Long-Term Support) -Version auszuwählen, da sie eine stabile und gut unterstützte Umgebung bietet. Nachdem die Virtualisierungssoftware gestartet wird, wird eine neue virtuelle Maschine erstellt. Die VM wird benannt und es wird "Linux" als Betriebssystem und "Ubuntu" als Version ausgewählt [12]. Die Speichereinstellungen für die VM wird konfiguriert, indem 2 GB RAM zugewiesen wird, um eine reibungslose Ausführung von Ubuntu sicherzustellen. Die virtuelle Festplatte wird für die VM erstellt und die Größe und der Speichertyp festgelegt. Es wird empfohlen, eine dynamisch allokierte Festplatte zu verwenden, um Speicherplatz zu sparen. Jetzt wird das zuvor heruntergeladene Ubuntu ISO-Image als Installationsmedium für die VM ausgewählt und die VM wird gestartet [12]. Es wird den Anweisungen zur Installation von Ubuntu gefolgt, indem die gewünschten Einstellungen wie Sprache, Tastaturlayout und Benutzerdaten ausgewählt werden. Nachdem die Virtual Maschine mit Ubuntu erfolgreich eingerichtet ist, kann die Programmierumgebung Qt Creator installiert und die LXI-Bibliothek eingebunden werden.

3.2 Einrichtung von Qt Creator

Bevor mit dem Programmieren beginnen können, ist eine Entwicklungsumgebung (Qt Creator) zu installieren. zunächst wird Qt-creator je nach Ubuntu-Version von der Qt-Downloadseite heruntergeladen. Nachdem die Datei heruntergeladen wurde, wird die Installationsdatei ausgeführt. Über gemeinsame Ordner wird die heruntergeladene Installationsdatei von dem Host-Computer auf die virtuelle Maschine übertragt.[13] In der virtuellen Maschine wird die Installationsdatei vom entsprechenden Speicherort aus gestartet. Dabei werden gegebenenfalls die Berechtigungen angepasst und das Installationsprogramm ausgeführt. [25] Das Installationsprogramm wird als erstes darum bitten, einen "Qt Account" anzulegen, ohne den man nicht weiterkommt.[25] Danach muss man die Vertragsbedingungen akzeptieren. Schließlich kommst du zur Auswahl der Qt Komponenten.[25] Die gewünschten Komponenten werden ausgewählt, die installiert werden sollen. Dabei wird darauf geachtet, die Option "Qt Creator" auszuwählen. Gewartet wird, bis der Installationsvorgang abgeschlossen ist. Dies kann je nach Geschwindigkeit der Virtual Maschine und der Internetverbindung einige Zeit in Anspruch nehme. Nach der Installation kann Qt Creator aus dem Startmenü geöffnet werden. Den Anweisungen des Setup-Assistenten wird gefolgt und die gewünschten Einstellungen werden ausgewählt: Sprache, Lizenzvereinbarung akzeptieren, Komponentenauswahl: Dies könnte zusätzliche Compiler oder Entwicklungsumgebungen sein, die mit Qt zusammenarbeiten, Pfade festlegen, Compiler auswählen, Debugger auswählen. Alle ausgewählten Optionen werden von dem Assistenten zusammengefasst und eine Übersicht wird angezeigt. Wenn alles korrekt ist, wird du die Installation abgeschlossen. die bereits installierten Qt-Kits werden in den Einstellungen des Qt Creator hinzugefügt. Schließlich ist es möglich, ein Qt-Testprojekt zu erstellen, um zu überprüfen, ob alles richtig funktioniert. Die Abbildung 4 stellt der Begrüßungsbildschirm von Qt Creator dar.

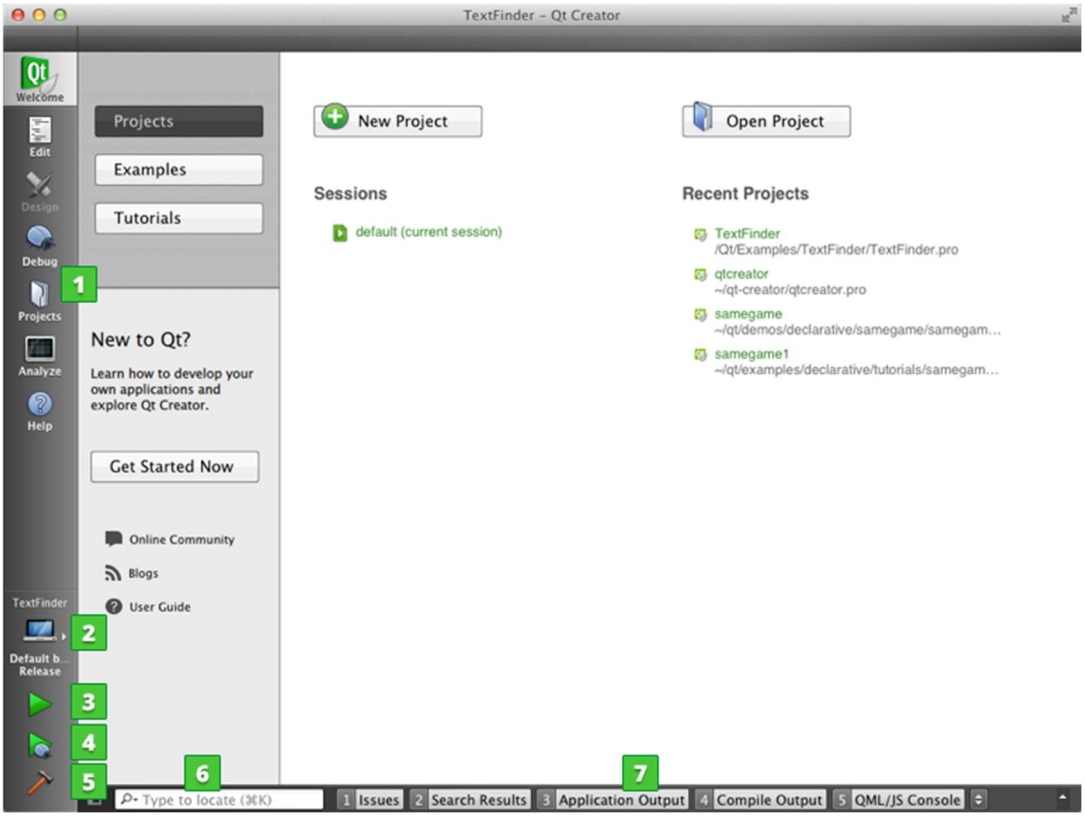


Abbildung 4: Der Begrüßungsbildschirm von Qt Creator [26]

Eine andere Möglichkeit wäre Qt Creator durch das Terminal über die Befehlszeile zu installieren. Um Qt Creator einzurichten, wird das Terminal geöffnet und der Befehl „sudo apt-get install qtcreator“ eingegeben und die Installation wird bestätigt, indem ‘J' eingeben wird. [13] Dadurch wird das Paket qtcreator aus den offiziellen Ubuntu-Repositories heruntergeladen und installiert. Sobald die Installation abgeschlossen ist, kann Qt Creator entweder über das Anwendungsmenü oder das Terminal gestartet werden [13]. Das System wird durch den Befehl „$ sudo apt-get update“ aktualisiert, um sicherzustellen, dass die neuesten Paketinformationen vorhanden sind. Es folgt die Installation durch den Befehl „$ sudo apt-get install Build-essential“ der Compilerwerkzeuge und relevante Hilfsprogramme, die für die Kompilierung von Qt-Anwendungen erforderlich sind [13]. Der Befehl „sudo apt-get install libfontconfig1“ wird verwendet, um das Paket libfontconfig1 auf dem Ubuntu-System zu installieren. Dieses Paket enthält die Bibliothek "Fontconfig", die zur Verwaltung und Konfiguration von Schriftarten auf Linux-Systemen verwendet wird. Nachdem die

Installation von Qt-creator abgeschlossen ist, wird Qt Creator über den Anwendungs-Dash gesucht und gestartet. Alternativ kann den Befehl „qtcreator“ ausgeführt werden. Dies öffnet die Qt Creator Entwicklungsumgebung. Um sicherzustellen, dass alles ordnungsgemäß installiert wurde, wird ein neues Projekt in Qt Creator erstellt. Eine Projektvorlage wird ausgewählt und die Anweisungen im Assistenten gefolgt.

3.3 Einrichten der LXI-Bibliothek

Um das Oszilloskop über Ethernet auslesen zu können, ist es notwendig, die LXI-Bibliothek in die Programmierumgebung einzubinden. Dazu müssen zunächst die notwendigen Dateien und Bibliotheken heruntergeladen und installiert werden. Um die LXI-Bibliothek in Qt Creator einzurichten, muss sie zunächst von der offiziellen Webseite heruntergeladen werden [14] und dann den Installationsanweisungen gefolgt werden. Danach wird die Bibliothek in Qt Creator eingebunden, indem zuerst in einem Terminal der Befehl „sudo apt-get install liblxi“ eingegeben wird. Dadurch werden die erforderlichen Dateien für die liblxi-Bibliothek auf dem System installiert. Anschließend wird der Befehl „sudo apt-get install liblxi-dev“ eingegeben, um die Header-dateien und die Entwicklungspakete zu installieren, die für die Verwendung der liblxi-Bibliothek in der Qt creator Entwickungsumgebung erforderlich sind [14]. Nach der Bestätigung dieser beiden Befehle (vgl. Abb. 4) wird die liblxi-Bibliothek auf dem System installiert und kann nun verwendet werden. Die in der Bibliothek enthaltenen Funktionen können in dem Programm aufgerufen werden, um ZB. das Oszilloskop über Ethernet auszulesen.

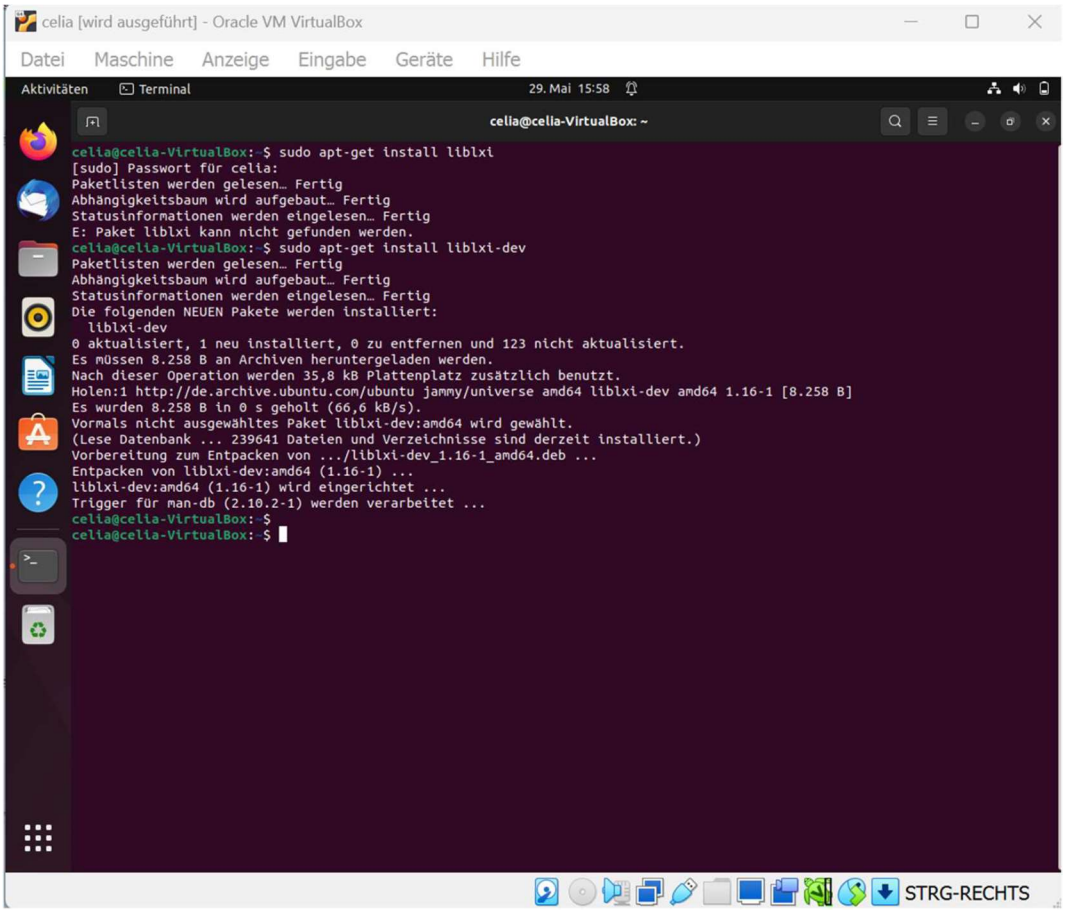


Abbildung 5: Installation von liblxi-Bibliothek

Sobald die LXI-Bibliothek auf dem System installiert ist und die notwendigen Entwicklungspakete installiert sind, wird die Datei mit der Erweiterung ".pro" (text_lxi.pro) des Projekts in Qt Creator geöffnet. Die Einbindung der LXI-Bibliothek wird konfiguriert, indem die Abschnitte "CONFIG" und "LIBS" in der ".pro"-Datei xxxxxtext_lxi.pro geändert werden. Die Konfiguration C++11 wird dem Abschnitt "CONFIG" hinzugefügt, also `CONFIG += c++11`. Durch die Hinzufügung von "CONFIG += c++11" wird Qt Creator angewiesen, den Code gemäß den Regeln und Funktionen des C++11-Standards zu kompilieren und zu behandeln. Die LXI-Bibliothek kann dann im Projekt verwendet werden, wenn die Einträge für die LXI-Bibliothek im Abschnitt "LIBS" hinzugefügt werden. Die entsprechende Codezeile zum Verlinken der Bibliothek mit dem Projekt lautet `LIBS += -llxi`. Die LXI-Bibliothek besitzt jedoch noch eine Abhängigkeit zur TIRPC-Bibliothek, die ebenfalls in die .pro Datei eingetragen werden muss. Die TIRPC-Bibliothek enthält Funktionen, Welche die Verwendung von Remote Procedure Calls (RPC) vereinfachen [15]. Deswegen lautet der vollständige LIBS-Eintrag in der .pro Datei LIBS += -llxi -ltirpc, wodurch im Projekt sowohl die LXI-Bibliothek (-llxi) als die

TIRPC-Bibliothek (-ltirpc) verwendet werden kann. Die Datei text_lxi.pro muss gespeichert werden, um die Änderungen wirksam werden zu lassen. Im Hauptfenster von Qt Creator wird der Build-Prozess ausgeführt, um sicherzustellen, dass die Änderungen am Projekt übernommen werden. Sobald der Build erfolgreich abgeschlossen ist, können die LXI-Funktionen und -Strukturen im Code verwendet werden, indem die entsprechenden Header-Dateien eingebunden und die gewünschten Funktionen aufgerufen werden.

3.4 SCPI-Befehle für die Datenaufnahme

In dieser Arbeit werden spezifische SCPI-Befehle verwendet, um das Oszilloskop R&S®RTB2004 über eine LXI-Verbindung anzusteuern und Messdaten abzurufen. Nachfolgend sind die SCPI-Befehle, die in dieser Arbeit zum Einsatz kommen: DISPlay: CLEar [: SCReen] löscht alle Messkurven, Beschriftungen und die Messergebnisse von gelöschten Messkurven. Alle Einstellungen bleiben unverändert [2]. *IDN?: gibt die Gerätekennung zurück. [2] CHANnel<m>: STATe <State> schaltet das Kanalsignal ein oder aus. Das Suffix <m> wählt den Eingangskanal aus. Die Anzahl der Kanäle hängt von dem Gerät ab.[2] MEASurement<m> [: ENABle] <State> aktiviert oder deaktiviert die ausgewählte Messung.[2] MEASurement<m>: RESult [: ACTual]? [<MeasType>] gibt das Ergebnis der angegebenen Messart zurück. Das Suffix:<m>: 1...6 wählt die Messstelle aus.[2] SPECtrum [: STATe] <State> schaltet die Spektrumanalyse mit Parameter ON bzw. OFF ein bzw. aus.[2] SPECtrum: SOURce <Source> wählt die Quelle für die Spektralanalyse-Diagramme aus (CH1 | CH2 | CH3 | CH4). [2] SPECtrum: FREQuency:STARt <StartFrequency>: Bestimmt die StartFrequenz des angezeigten Frequenzbereichs am linken Displayrand: Center - Span/2. Sie können Start- und Stoppfrequenz einstellen, anstatt eine Mittenfrequenz und einen Frequenzbereich zu definieren. <StartFrequenz> Bereich: Hängt von verschiedenen anderen Einstellungen ab, hauptsächlich von Zeit Basis, Span/RBW-Verhältnis und Mittenfrequenz.[2] SPECtrum: FREQuency: STOP <StopFrequency> bestimmt die Stoppfrequenz des angezeigten Frequenzbereichs am rechten Displayrand: Mitte + Spanne/2.[2] SPECtrum:FREQuency:WINDow:TYPE <WindowFunction> Fensterfunktionen werden mit den Eingangswerten multipliziert und können so die Darstellung der Spektrumanalyse verbessern. Die Parameter sind: RECTangular, HAMMing, HANNing, BLACkmanharris und FLATtop.[2]

SPECtrum: FREQuency: MAGNitude: SCALe< MagnitudeScale > bestimmt die Skalierungseinheit der y-Achse. Die Parameter sind: LINear, DBM, DBV, DBUV. [2] SPECtrum:FREQuency:CENTer <CenterFrequency> legt die Position des angezeigten Frequenzbereichs fest, d. h. (Center - Span/2) bis (Zentrum + Span/2). Die Breite des Bereichs wird mit dem Befehl SPECtrum: FREQuency: SPAN eingestellt. [2] SPECtrum: FREQuency:SPAN<Span> Der Frequenzbereich wird in Hertz angegeben und definiert die Breite des angezeigten Frequenzbereichs, der von (Mitte - Span/2) bis (Mitte + Span/2) reicht. SPECtrum: FREQuency:BANDwidth:AUTO ON aktiviert die automatische Einstellung der Auflösungsbandbreite (RBW) im Spektrummodus. Dies bedeutet, dass das Oszilloskop die RBW automatisch anpassen wird, um die beste Auflösung für das aktuelle Frequenzspektrum zu erzielen. [2] SPECtrum: TIME:RANGe <TimeRange> legt den Zeitbereich für das Zeitbereichsdiagramm fest.[2] DISPlay:CBAR:FFT: [POSition]]<DividerPosition> Bestimmt die Position des Trennstrichs zwischen normaler Wellenform und FFT-Fenster. Der Parameter: <DividerPosition> Vertikale Position in Pixel, gemessen vom oberen Rand. Die vertikale Anzeigegröße beträgt 800 px. Std.-Einheit: p SPECtrum: TIME:POSition <TimePosition> legt die Zeitposition des analysierten Zeitbereichs fest. [2]

4 Messgrößen und Analyse

In diesem Abschnitt werden die grundlegenden Aspekte der Signalanalyse vorrangig behandelt, insbesondere im Hinblick auf Rechtecksignale. Diese Signalform zeichnet sich durch ihre periodische und impulsartige Natur aus und wird in vielen technischen Anwendungen eingesetzt. Um diese Signale genau charakterisieren zu können, ist ein fundiertes Wissen über die verschiedenen Messgrößen und Analysemethoden unerlässlich. Im Folgenden werden wir uns mit den folgenden Schlüsselbegriffen beschäftigen.

4.1 Frequenz

Die Frequenz (von lateinisch frequentia ‚Häufigkeit'; auch Schwingungszahl genannt) ist in Physik und Technik ein Maß dafür, wie schnell bei einem periodischen Vorgang die Wiederholungen aufeinander folgen, z.B. [29] bei einer fortdauernden Schwingung. Im Falle eines Rechtecksignals beschreibt die Frequenz (siehe Abb. 5), wie oft das Signal innerhalb einer bestimmten Zeitspanne

zwischen seinen beiden Pegeln wechselt. [18] Die Einheit der Frequenz ist die abgeleitete SI-Einheit mit dem besonderen Namen Hertz (Einheitenzeichen Hz). Bei manchen Vorgängen werden auch die Bezeichnungen Folgefrequenz, Impulsfolgefrequenz oder Hubfrequenz verwendet, bei Drehbewegungen Drehzahl.[18] Die korrekte Messung der Frequenz ermöglicht es, Signale gezielt zu analysieren und zu manipulieren. Darüber hinaus ist die Kenntnis der Frequenz wichtig, um Störungen zu identifizieren und zu eliminieren, insbesondere in Situationen, in denen mehrere Signale aufeinandertreffen.

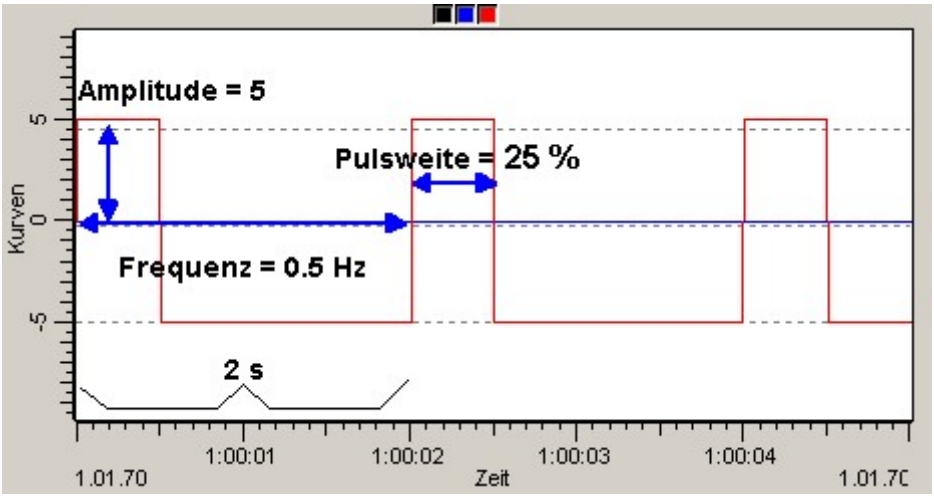


Abbildung 6: Rechtecksignal mit der angegebenen Frequenz (Hz), der vorgegebenen Amplitude und Pulsweite (in Prozent) [19].

4.2 Effektivwert und Mittelwert

Der Effektivwert oder auch im Englischen RMS (root Mean Square) -Wert gibt für elektrische Wechselspannungen und Wechselströme den Wert an, den eine Gleichspannung, beziehungsweise Gleichstrom haben müsste, um dieselbe Wärmeleistung in einem rein ohmschen Verbraucher umzusetzen.[17] Der Effektivwert eines Signals ist ein entscheidender Messwert, der die tatsächliche Leistung oder Stärke eines Signals angibt. Im Kontext eines Rechtecksignals bezieht sich der Effektivwert auf den Wert einer konstanten Gleichspannung, der die gleiche Leistung wie das Rechtecksignal hätte. Ein Rechteck mit demselben Flächeninhalt und derselben Breite hat die mittlere Höhe der Fläche; diese Höhe ist der Mittelwert.[20] Der Mittelwert ist ein wichtiger Kennwert eines Signals, der angibt, wie stark ein Signal im Durchschnitt von einem festgelegten Wert abweicht. Im Kontext eines Rechtecksignals bezieht sich der Mittelwert auf den Durchschnitt

der Signalwerte über eine bestimmte Zeitperiode.[30] Für Rechtecksignale ist der Mittelwert direkt mit der Amplitude und der Periode des Signals verbunden.

4.3 Duty-Cycle Abtastgrad

Bei Duty-Cycle-Messungen wird das Tastverhältnis, auch Pulsgrad, Modulationsgrad oder Duty-Cycle genannt, von Impulsen gemessen. Jeder Impuls besteht aus einem Dach und einem Boden des Impulses. Die Duty-Cycle-Messung bestimmt das Verhältnis zwischen diesen beiden Größen, das auch als Verhältnis zwischen der Impulsdauer und der Periodendauer ausgedrückt werden kann. Ein Duty Cycle von 50 % wird auch symmetrischer Puls genannt. [21] Es gibt zwei grundlegende Arten:
Positive Duty-Cycle: Dies bezieht sich auf den Prozentsatz des Zeitraums, in dem das Signal auf einem hohen Pegel (Ein-Zustand) ist. [22]
Negative Duty-Cycle: Hierbei handelt es sich um den Prozentsatz des Zeitraums, in dem das Signal auf einem niedrigen Pegel (Aus-Zustand) ist. [22] Beide Duty-Cycle-Größen zusammen ergeben 100%, da sie den gesamten Signalzyklus abdecken.

4.4 FFT-Analyse

Die Frequenzanalyse mit dem Oszilloskop analysiert und interpretiert das Zeitsignal. Als Werkzeug dient die Fast-Fourier-Transformation (FFT), um damit das Zeitsignal aufzuschlüsseln.[23] Sie zerlegt ein Signal in einzelne Spektralkomponenten und gibt dadurch Aufschluss über seine Zusammensetzung. FFTs werden zur Fehleranalyse, in der Qualitätskontrolle und in der Zustandsüberwachung von Maschinen oder Systemen eingesetzt. Genau genommen handelt es sich bei der FFT um einen optimierten Algorithmus zur Implementierung der „Diskreten Fourier Transformation“, kurz DFT [24]. Dabei wird ein zeitlich begrenzter Ausschnitt eines Signals in seine Bestandteile zerlegt. Diese Bestandteile sind einzelne Sinus-Schwingung bei diskreten Frequenzen, deren Amplitude und Phase bestimmt werden. Die FFT erlaubt also die Sicht auf ein Signal im Frequenzbereich.[24] Bei einer FFT-Analyse wird eine zeitbasierte Messkurve in ein Frequenzspektrum umgewandelt. Als Ergebnis wird die Magnitude der ermittelten Frequenzen Leistung/Frequenz Diagramm (Spektrum)

angezeigt. (siehe Abb.6). FFT-Ergebnisse sind nützlich, um eine Übersicht über das Eingangssignal im Frequenzbereich zu erhalten und ungewöhnliche Signaleffekte (z. B. Nebenaussendungen oder Verzerrungen) visuell zu erkennen [2] Die FFT-Analyse wird für Signale durchgeführt, die an einem der aktiven Eingangskanäle oder einer der aktiven mathematischen oder Referenzmesskurven erfasst werden. Es ist immer nur an einem einzigen Kanal eine Analyse durchführbar. [2] Um die Berechnungszeit zu verkürzen, stellt das Gerät automatisch die Zeitskala (und damit die Erfassungszeit) und einen Ausschnitt der Zeitbasis ein, für welche die FFT berechnet wird. Dieser Zeitabschnitt wird durch weiße Linien im Zeitdiagramm dargestellt. Die Ergebnisse der FFT-Analyse können auf einen bestimmten Frequenzbereich beschränkt werden. Dazu wird eine Mittenfrequenz und eine Frequenzdarstellbreite oder die Start- und Stoppfrequenzen angegeben. Alternativ kann die Auflösebandbreite (RBW) manuell vergrößert werden. [2] Wenn die Frequenzparameter geändert werden, werden die Zeitbasis und das Zeitfenster in der Messkurve automatisch angepasst. Eine RBW-Änderung wirkt sich auf das Zeitfenster aus.[2]

Abbildung 7: FFT-Spektrum eines Rechtecksignal

5 Implementierung

Im Rahmen dieser Arbeit wurde das Oszilloskop R&S®RTB2004 erfolgreich über Ethernet mit Hilfe der Programmierumgebung Qt Creator und der LXI-Bibliothek

ausgelesen. Dazu wurden die Grundlagen der LXI-Bibliothek erläutert und gezeigt, wie sie in die Programmierumgebung integriert werden kann. Zudem wurde C-Programmierung genutzt, um die benötigten SCPI-Befehle für die Kommunikation mit dem Oszilloskop zu implementieren. Die Implementierung wurde in verschiedenen Schritten durchgeführt, beginnend mit der Einrichtung der Netzwerkverbindung bis hin zur erfolgreichen Datenübertragung und -verarbeitung. Die Ergebnisse zeigen, dass die Implementierung erfolgreich war, da das Oszilloskop zuverlässig ausgelesen werden konnte.

5.1 Wahl der Programmierumgebung

Die Wahl der Programmierumgebung war ein wichtiger Faktor bei der Umsetzung dieses Projekts. Qt Creator wurde ausgewählt, da es sich um eine benutzerfreundliche und leistungsstarke IDE handelt, mit der die Bibliotheken wie LXI leicht integriert werden können. Qt Creator bietet eine vollständige integrierte Entwicklungsumgebung mit Funktionen wie Code-Editor, Debugger, Assistent für das Design der Benutzeroberfläche und Projektmanagement. Dies erleichtert den Prozess der Anwendungsentwicklung und die Erstellung einer benutzerfreundlichen Benutzeroberfläche. Die Kombination aus Qt Creator und C hat dazu beigetragen, die Auslesung des Oszilloskops über Ethernet zu vereinfachen.

5.2 Implementierung der SCPI-Befehle und Steuerung des Oszilloskops über Ethernet

Die Implementierung der SCPI-Befehle und die Steuerung des Oszilloskops über Ethernet erfolgen in mehreren Schritten. Zunächst wurde die physische Konfiguration der Netzwerkverbindung behandelt. Hier wird überprüft, ob das Ethernet-Kabel korrekt mit dem Computer und dem Oszilloskop verbunden ist. Sobald die Verbindung hergestellt ist, leuchten die Verbindungs- oder Aktivitäts-LEDs (Vgl. Abb.5).

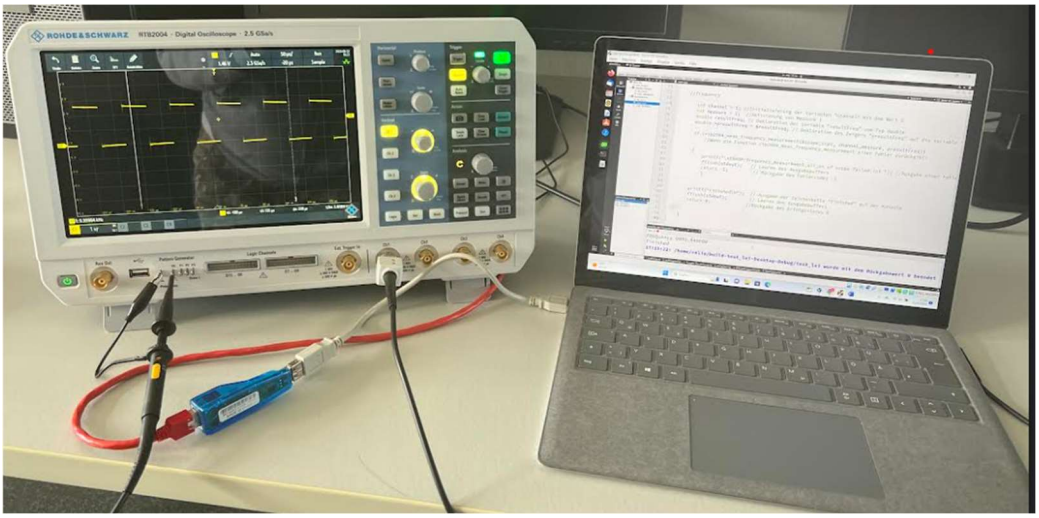


Abbildung 8: Physische Konfiguration der Netzwerkverbindung

Anschließend werden Einstellungen und Konfigurationen für das Projekt vorgenommen. In Qt-creator wird ein Ordner „test_lxi“ erstellt, der eine Datei „test_lxi.pro“ und zwei Unterordner „Header-Dateien“ und „Quelldateien“ enthält. In der Datei „test_lxi.pro“ werden die Einstellungen und Konfigurationen für des Projekts abgelegt. Die in dieser Datei geschriebene Code definiert die Build-Einstellungen und Abhängigkeiten des Projekts. Er gibt an, welche Quelldateien, Header-Dateien und Bibliotheken in das Projekt einbezogen werden sollen. Die Header-Dateien „lxi_con.h“ und „scpi_rtb2004.h“, die im Ordner „Header-Dateien“ befinden, enthalten die Deklarationen der Funktionen und Strukturen, die für die Kommunikation mit dem Oszilloskop erforderlich sind. Im Ordner „Quelldateien“ befinden sich die Dateien „main.cpp“, „lxi_con.c“ und „scpi_rbt2004.c“. die Quelldateien „lxi_con.c“ und „scpi_rbt2004.c“ enthalten die Implementierung dieser Funktionen. Diese Funktionen werden dann in „main.cpp“ aufgerufen. Die Bibliotheken „liblxi“ und „libtirpc“ werden von der Linken verwendet, um die erforderlichen Funktionen der LXI-Bibliothek und eventuell weiterer Abhängigkeiten bereitzustellen. Als nächstes wird die Verbindung zum Oszilloskop hergestellt. Die Dateien „lxi_con.c“ und „lxi_con.h“ arbeiten zusammen, um die Steuerung und Kommunikation mit LXI-Verbindung zu ermöglichen, wobei die Header-Datei „lxi_con.h“ die Funktionssignaturen und Strukturdefinitionen enthält. Durch die Verwendung der Header-Datei können die Funktionen und Strukturen in verschiedenen Quellcodedateien verwendet werden, ohne den Code jedes Mal erneut schreiben zu müssen. Die Quelldatei „lxi_con.c“ enthält die Implementierung der Funktionen, die in der Header-Datei deklariert

wurden. Folgende Strukturen, Funktionen und Konstanten werden in der Header-Datei „lxi_con.h“ deklariert:

DEFAULT_TIMEOUT mit dem Wert 100. Diese Konstante repräsentiert den Standard-Timeout-Wert für die Verbindung.

typedef struct {...} lxi_connection_t; diese Struktur hat den Namen lxi_connection_t und enthält verschiedene Felder, mit der die LXI-Verbindung repräsentiert wird. Die Felder enthalten einen Handler zur Verbindung, die IPv4-Adresse des Geräts, den Timeout-Wert, die Portnummer und das LXI-Protokoll.

lxi_con_init (lxi_connection_t * con) ist die Funktionssignatur für die Initialisierung der LXI-Verbindung. Die Funktion erhält einen Zeiger auf eine lxi_connection_t-Struktur als Parameter und gibt einen Integer-Wert zurück.

lxi_con_deinit (lxi_connection_t * con) ist die Funktionssignatur für das Deinitialisieren der LXI-Verbindung. Die Funktion nimmt einen Zeiger auf eine lxi_connection_t-Struktur als Parameter und gibt einen Integer-Wert zurück.

lxi_con_send (lxi_connection_t * con, char * msg, int len); Das ist die Funktionssignatur zum Senden von Daten über die LXI-Verbindung. Die Funktion erhält einen Zeiger auf eine lxi_connection_t-Struktur, einen Zeiger auf einen Puffer mit den zu sendenden Daten und die Länge der Daten als Parameter. Sie gibt einen Integer-Wert zurück.

lxi_con_receive (lxi_connection_t * con, char * msg, int * len, int max_len) ist die Funktionssignatur zum Empfangen der Daten über die LXI-Verbindung. Die Funktion nimmt einen Zeiger auf eine lxi_connection_t-Struktur, einen Zeiger auf einen Puffer zum Speichern der empfangenen Daten, einen Zeiger auf eine Variable zur Speicherung der tatsächlichen Länge der empfangenen Daten und die maximale Länge des Puffers als Parameter. Sie gibt einen Integer-Wert zurück.

Die Dateien "scpi_rbt2004.c" und "scpi_rbt2004.h" enthalten Funktionen und Definitionen, die spezifisch für das R&S RTB2004-Oszilloskop und das SCPI-Protokoll sind. Diese Dateien implementieren SCPI-Befehle und Funktionen, um das Oszilloskop zu steuern und Messungen durchzuführen. Der Code in „scpi_rtb2004.h" definiert die Schnittstelle und Funktionen für die Steuerung des R&S RTB2004-Oszilloskops über das SCPI-Protokoll. Zu Beginn des Codes werden zwei Direktiven eingefügt, damit der Code nur einmal in die Headerdatei "scpi_rtb2004.h" eingefügt wird, auch wenn es mehrere Stellen gibt, an denen die Header-dateien über eine include Einweisung eingebunden wird, und zwar „#ifndef SCPI_RTB2004_H“ und „#define SCPI_RTB2004_H“. Dann werden die Definitionen und Funktionen der Datei „lxi_con.h“ in "scpi_rtb2004.h" durch die

Einbindung der Datei „lxi_con.h“ mit der Präprozessor-Direktive „#include "lxi_con.h" “ zur Verfügung gestellt. Dadurch können Funktionen, Typdefinitionen und andere Inhalte aus "lxi_con.h" verwendet werden, als ob sie direkt im aktuellen Code geschrieben wären. Diese Direktive ermöglicht die modulare Organisation von Code und die Wiederverwendung von Funktionen und Strukturen in verschiedenen Dateien. Es wird auch über die Direktiven #define RTB2004_ERROR -1, #define RTB2004_INVALID_PARAMETER -2, usw.: Fehlercodes definiert, die im Zusammenhang mit dem R&S RTB2000-Oszilloskop auftreten können. Diese Fehlercodes werden verwendet, um Fehlerzustände zu identifizieren und zu behandeln. Es werden hier zwei Funktionen verwendet. init_rtb2004 führt die Initialisierung des Oszilloskops durch, indem ein Kommando gesendet wird, um den Bildschirm zu löschen und alte Messungen zu entfernen. rtb2004_read_id führt eine Abfrage des Identifikationsstrings des Oszilloskops durch, indem sie ein Kommando sendet, um den IDN-String abzurufen, und dann die Antwort vom Oszilloskop empfängt. In den Funktionen werden verschiedene Fehlercodes zurückgegeben, z.B. wenn die Verbindung fehlschlägt oder die Antwort des Oszilloskops unerwartet ist. Die main-Funktion ist somit eine Art Steuerzentrale, die den Ablauf des Programms koordiniert und die verschiedenen Funktionen und Module miteinander verbindet. Im Folgenden befindet sich die Liste der Module und Funktionen, die in Main verwendet werden: Die Dateien lxi_con.h und lxi_con.c enthalten Funktionen zur Initialisierung und Verwaltung der LXI-Verbindung. In der main-Funktion wird die Funktion lxi_con_init () aufgerufen, um die LXI-Verbindung zu initialisieren. Die LXI-Verbindungsstruktur "scope_lxi" wird initialisiert und die Verbindungsparameter wie IP-Adresse, Timeout, Portnummer und Protokoll werden festgelegt. Anschließend wird die Funktion "lxi_con_init" aufgerufen, um die LXI-Verbindung herzustellen. Wenn die Verbindung fehlschlägt, wird eine Fehlermeldung ausgegeben und das Programm wird beendet. Die Dateien scpi_rtb2004.h und scpi_rtb2004.c enthalten Funktionen zur Steuerung des RTB2004 Oszilloskops über SCPI-Befehle. In der main-Funktion wird die Funktion init_rtb2004() aufgerufen, um das Oszilloskop zu initialisieren, und die Funktion rtb2004_read_id () wird aufgerufen, um die Geräte-ID des Oszilloskops auszulesen. Die SCPI-Struktur "scope_scpi" wird erstellt und der Zeiger auf die LXI-Verbindungsstruktur "scope_lxi" wird zugewiesen. Dies ermöglicht die Kommunikation zwischen der SCPI-Bibliothek und der LXI-Verbindung. Die SCPI-Befehle zur Steuerung des Oszilloskops werden über die LXI-Verbindung gesendet. Durch die Zuweisung des

Zeigers auf die LXI-Verbindungsstruktur an die SCPI-Struktur kann die SCPI-Bibliothek auf die erforderlichen Verbindungsinformationen wie IP-Adresse, Portnummer und Protokoll zugreifen. Diese Befehle werden über die LXI-Verbindung an das Oszilloskop gesendet, und die Antwortdaten werden über die LXI-Verbindung zurückgegeben. Die SCPI-Struktur enthält den Zeiger auf die LXI-Verbindungsstruktur, sodass die SCPI-Bibliothek die Verbindungsressourcen verwenden und die Daten über die Verbindung senden und empfangen kann. Durch diese Kommunikationsschnittstelle zwischen der SCPI-Bibliothek und der LXI-Verbindung können SCPI-Befehle zur Steuerung des Oszilloskops über die Ethernet-Verbindung gesendet werden und die entsprechenden Antworten und Messdaten empfangen werden. Es ist sehr wichtig, dass die LXI-Verbindung korrekt initialisiert wird, bevor sie der SCPI-Struktur zugewiesen wird. Dadurch wird eine korrekte Kommunikation zwischen der SCPI-Bibliothek und dem Oszilloskop über die Ethernet-Verbindung sichergestellt. Die Funktion "init_rtb2004" wird aufgerufen, um das Oszilloskop zu initialisieren. Wenn die Initialisierung fehlschlägt, wird eine Fehlermeldung ausgegeben und das Programm wird beendet. Die Aufgabe dieser Funktion besteht darin, alle erforderlichen Einstellungen und Konfigurationen vorzunehmen, um das Oszilloskop betriebsbereit zu machen. Diese Funktion sendet die entsprechenden SCPI-Befehle an das Oszilloskop, um es zu initialisieren. Dies kann das Zurücksetzen auf Standardwerte, die Aktivierung bestimmter Funktionen oder die Konfiguration von Messparametern umfassen. Durch die Initialisierung wird das Oszilloskop in einen definierten Zustand versetzt, um korrekte und zuverlässige Messungen durchzuführen. Die Funktion gibt einen Statuswert zurück, der den Erfolg oder Misserfolg der Initialisierung des Oszilloskops widerspiegelt. Wenn die Initialisierung erfolgreich war, wird ein Wert größer als 0 zurückgegeben. Andernfalls wird ein Fehlercode zurückgegeben, der auf das Problem bei der Initialisierung hinweist. Wenn die Initialisierung des Oszilloskops fehlschlägt, wird eine entsprechende Fehlermeldung auf der Konsole ausgegeben und das Programm wird beendet. Dies dient dazu, sicherzustellen, dass das Oszilloskop korrekt initialisiert wurde, um genaue Messungen durchzuführen. Die Funktion "rtb2004_read_id" wird aufgerufen, um die Geräte-ID des Oszilloskops zu lesen. Wenn das Lesen der ID fehlschlägt, wird eine Fehlermeldung ausgegeben und das Programm wird beendet. Andernfalls wird die Geräte-ID auf der Konsole ausgegeben. Die main-Funktion verwendet auch Standard-C-Funktionen aus stdio.h für Ein- und Ausgabefunktionen (printf ()), string.h für

Zeichenkettenoperationen (strcpy ()), iostream für Ein- und Ausgabefunktionen (std::cout), und QCoreApplication für die Qt-Anwendung.

5.3 Durchführung von Messung mit Rechtecksignal

Ein Rechtecksignal bzw. eine Rechteckschwingung bezeichnet ein periodisches Signal, das zwischen zwei Werten hin und her schaltet und in einem Diagramm über der Zeit einen rechteckigen Verlauf aufweist [7]. Die Frequenz eines periodischen Signals wie einem Rechtecksignal gibt an, wie oft das Signal pro Zeiteinheit wiederholt wird [8]. Sie wird in Hertz (Hz) gemessen und gibt an, wie viele vollständige Perioden des Signals pro Sekunde auftreten. Dazu kommt die Messungen von Effektivwert, Mittelwert, Duty-cycle Abtastgrad und die Durchführung einer FFT-Analyse. Um die Frequenz, den Effektivwert, den Mittelwert, und den Duty-cycle Abtastgrad eines Signals mit einem Oszilloskop zu messen, sollten theoretisch die folgenden Schritte befolgt werden: Das Oszilloskop wird ordnungsgemäß angeschlossen und eingeschaltet. Das Signal, dessen Frequenz (hier ein Rechtecksignal) gemessen werden soll, wird am Eingangskanal des Oszilloskops angeschlossen (vgl. Abb. 6). Dafür wird hier einen Tastkopf (vgl. Abb. 6) mit Krokodilklemme (vgl. Abb. 7) und Kabelklemme (vgl. Abb. 8) benutzt. Die Krokodilklemme wird an den Masseanschluss angeschlossen und an die Kabelklemme wird das Rechtecksignal (P0) geführt.

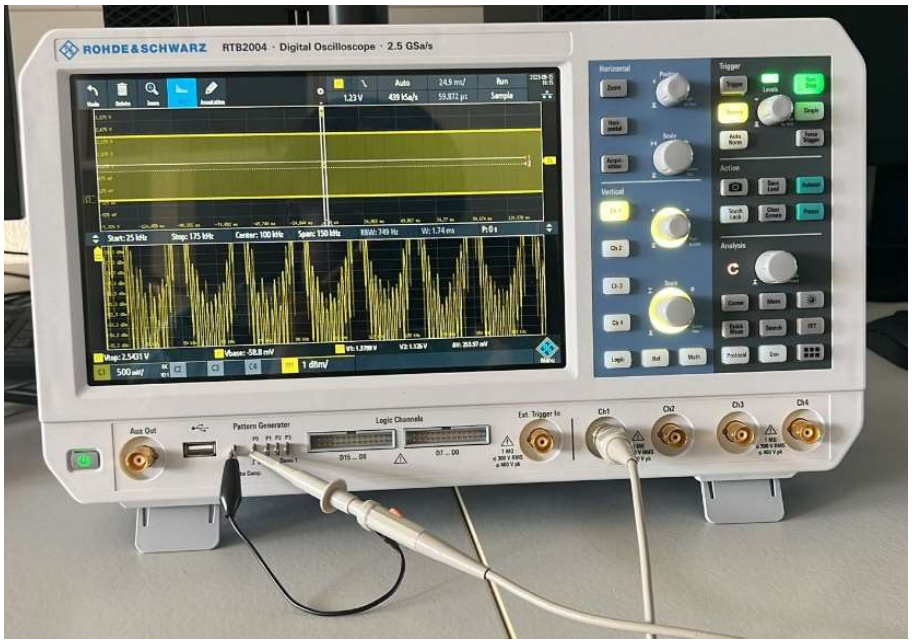


Abbildung 9: Anschluss des Tastkopfs an das Oszilloskop

Abbildung 10: Krokodilklemme [6].

Abbildung 11: Kabelklemme [6].

Sobald das Rechtecksignal ordnungsgemäß am Eingangskanal des Oszilloskops anliegt, können die Messungen durchführt werden. Der Kanal, an dem das Signal anliegt, wird ausgewählt, indem die entsprechende Taste auf dem Oszilloskop

gedrückt wird (hier channel 1). Die "Measure"-Taste auf dem Oszilloskop wird gedrückt, um das Messmenü aufzurufen. Die Liste der auf dem Oszilloskop verfügbaren Messungen wird auf dem Bildschirm angezeigt und von dort aus kann die Messung ausgewählt werden, die durchführt wird. Dann wird sichergestellt, dass die Trigger Parameter für das Rechtecksignal geeignet sind. Sobald alle erforderlichen Einstellungen vorgenommen wurden, wird die Messung durch einen Druck der Taste Run/Start gestartet. Das Oszilloskop analysiert das Signal und berechnet die Messdaten basierend auf den erfassten Daten und die Ergebnisse werden auf dem Oszilloskop-Bildschirm angezeigt. [2] Die FFT-Analyse wird durchgeführt, indem zunächst die [FFT]-Taste betätigt wird. Anschließend wird das FFT-Menü (siehe Abb. 11) durch erneutes Drücken der [FFT]-Taste geöffnet. Der Kanal, für den die erfassten Daten mit FFT analysiert werden, wird angegeben (siehe Abb. 11). Einen der aktiven Eingangskanäle oder eine mathematische oder Referenzmesskurve können ausgewählt werden.[2] Der passende FFT-Fenstertyp (siehe Abb.11) wird ausgewählt, der den Eigenschaften des Signals entspricht. Im Menü „Messkurve“ werden die gewünschten Messkurventypen ausgewählt, die angezeigt werden sollen. Wenn die Auflösebandbreite manuell angepasst werden soll, wird die Option „Automatische Auflösebandbreite“ deaktiviert.

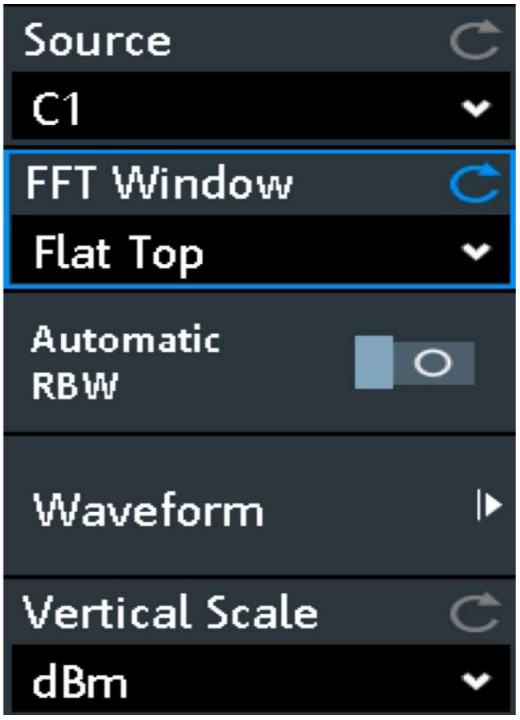


Abbildung 12 : FFT-Fenster [2]

Im FFT-Fenster wird der gewünschte Frequenzbereich angegeben, der im Spektrum Fenster angezeigt werden soll. Dabei haben Sie die Möglichkeit, entweder Start- und Stoppfrequenzen oder Mittenfrequenz und Frequenzdarstellbreite einzustellen (siehe Abb.12). Dieser Bereich wird als (Mitte - Span/2) bis (Mitte + Span/2) definiert. Die FFT-Analyse wird durch Betätigen der [Run Stop] -Taste gestartet oder gestoppt.

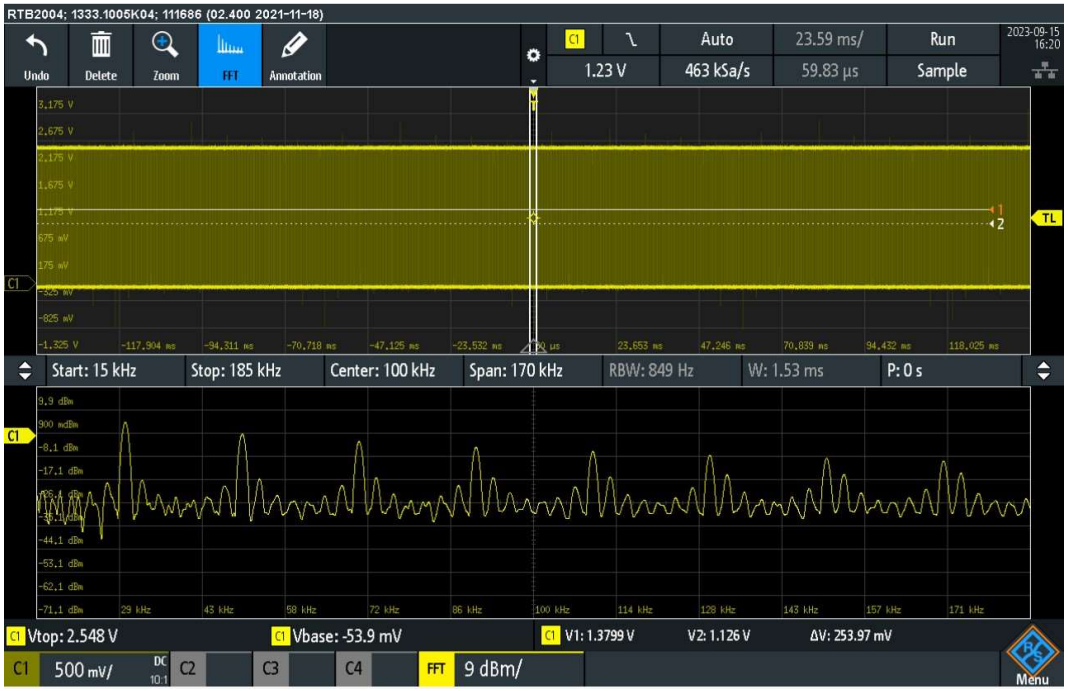


Abbildung 13: FFT-Spektrum mit Frequenzeinstellungen

In diesem Projekt werden automatisch verschiedene Messungen durch den programmierten Code durchgeführt. Bei der Ausführung des Codes erfolgt automatisch die Aktivierung der Kanaltaste, gefolgt von der Auswahl der Messtaste und der Durchführung von Messungen, wie Frequenz, Effektivwert, Mittelwert und Duty-Cycle-Abtastgrad. Sowie die Durchführung von FFT-Analyse. Insbesondere ist es möglich, diese Messungen für einen spezifischen Kanal (Kanal 1) zu aktivieren. Hierbei werden die entsprechenden Funktionen wie "rtb2004_meas_frequency_measurement(…)","rtb2004_meas_MEAN_measurement(…)","rtb2004_meas_RMS_measurement(…)","rtb2004_meas_PDCYcle_measurement(…)","rtb2004_meas_NDCYcle_measurement(…)" und “FFTmeasure(…)“ aufgerufen, um die jeweiligen Messungen zu vollziehen. Im Falle eines

fehlgeschlagenen Messvorgangs wird eine Fehlermeldung ausgegeben, woraufhin das Programm beendet wird. Bei erfolgreicher Messung werden die ermittelten Werte auf der Konsole angezeigt. "Die Frequenz, der Effektivwert, der Mittelwert, der Duty-Cycle-Abtastgrad und die FFT-Analyse werden wie folgt gemessen: Zunächst wird die Variable 'channel' auf den Wert 1 initialisiert, um den Kanal festzulegen. Dies signalisiert, dass die Messungen für Kanal 1 des Oszilloskops durchgeführt werden sollen. Im nächsten Schritt wird die Variable 'measure' auf den Wert 1 gesetzt, um die entsprechende Messfunktion zu aktivieren. Diese Funktion ermöglicht eine schnelle und automatische Messung der wichtigsten Signalkenndaten, ohne manuelle Einstellungen erforderlich zu machen.[2] Das Oszilloskop erkennt das Signal automatisch und führt die Messungen basierend auf den aktuellen Einstellungen und dem ausgewählten Messmodus durch. Es werden Variablen wie 'resultFreq' für die Frequenz, 'resultRms' für den Effektivwert, 'resultMEAN' für den Mittelwert und 'resultPDCYcle' und 'resultNDCYcle' für den positiv- bzw. negativ-Duty-Cycle deklariert. Zeiger wie 'presultFreq' werden auf die entsprechenden Variablen gesetzt, um auf ihre Speicherorte zu verweisen.
Die zugehörigen Funktionen wie 'rtb2004_meas_frequency_measurement' und 'rtb2004_meas_MEAN_measurement' werden aufgerufen, um die jeweiligen Messungen durchzuführen. Hierbei werden die SCPI-Struktur 'scope_scpi', die Kanalnummer 'channel', die aktivierte Messung 'measure' und die entsprechenden Zeiger übergeben. Falls eine Funktion einen Fehler zurückgibt, wird eine entsprechende Fehlermeldung auf der Konsole ausgegeben. Der Ausgabepuffer wird geleert und das Programm gibt den entsprechenden Fehlercode aus. Im Falle erfolgreicher Messungen wird die entsprechende Nachricht auf der Konsole angezeigt. Der Code nimmt eine Reihe von Einstellungen vor, um das Verhalten des Oszilloskops für die Frequenzanalyse zu konfigurieren. Die Spektrumanalyse wird aktiviert und der zu analysierende Kanal definiert. Anschließend wird der Fenstertyp für die FFT auf "HAMMing" gesetzt, um die Spektralanalyse zu verbessern. Die Skalierung der y-Achse für die FFT erfolgt logarithmisch, um genauere Messungen zu ermöglichen. Die vertikale Position des Spektrums wird auf 30 Divisionen festgelegt, und die vertikale Skala entsprechend angepasst. Die Position der Trennlinie zwischen der normalen Wellenform und dem FFT-Fenster wird definiert, um eine klare Darstellung zu gewährleisten. Der Startpunkt der Frequenz wird auf 0 Hz festgelegt, während die Stoppfrequenz auf 200.000 Hz eingestellt wird. Die Mitte der Frequenz wird auf 100.000 Hz gesetzt und der

Frequenzbereich für die FFT auf 200.000 Hz festgelegt. Die Bandbreite (RBW) wird automatisch festgelegt, um optimale Ergebnisse zu erzielen. Der Zeitbereich wird auf 2,6 Millisekunden gesetzt, und die Zeitposition wird auf 0 Sekunden eingestellt.

6 Anwendung und Ergebnisse

In diesem Kapitel wird die Auslesung von Messdaten und die Darstellung der Ergebnisse behandelt.

6.1 Auslesen von Messdaten

Sobald die LXI-Verbindung zum Oszilloskop erfolgreich initialisiert wurde, führt der Code die Messungen aus. Dafür werden der zu verwendende Kanal und der Messmodus festgelegt. In diesem Fall wird der Kanal mit dem Wert 1 initialisiert, was bedeutet, dass der erste Eingangskanal des Oszilloskops für die Messung verwendet wird. Der Messmodus wird auf 1 gesetzt, um die Messung zu aktivieren. Anschließend wird die Funktion rtb2004_meas_(…)_measurement aufgerufen und die erforderlichen Parameter übergeben, einschließlich der Verbindungsinformationen des Kanals, auf dem die Messungen durchgeführt werden sollen. Das Messergebnis wird in der Variable result(…) gespeichert, auf die über den Zeiger presult(…) zugegriffen wird. Der Rückgabewert der Funktion wird überprüft, um festzustellen, ob die Messungen erfolgreich waren. Wenn die Funktion einen Fehler zurückgibt, wird eine entsprechende Fehlermeldung auf der Konsole ausgegeben, um den Benutzer darüber zu informieren. In diesem Fall wird beispielweise die Zeichenkette "ERROR: frequency_measurement_all_on of scope failed" zusammen mit dem Typ "int" ausgegeben. Es wird auch der Ausgabepuffer geleert, um sicherzustellen, dass die Fehlermeldung sofort angezeigt wird. Wenn die Messungen erfolgreich waren, wird die Zeichenkette "Finished" auf der Konsole ausgegeben, um anzuzeigen, dass der Messvorgang abgeschlossen ist. Der Ausgabepuffer wird erneut geleert, um die Ausgabe sofort anzuzeigen. Schließlich wird der Erfolgsstatus 0 zurückgegeben, um anzuzeigen, dass das Programm ordnungsgemäß beendet wurde. Diese Schritte ermöglichen die Auslesung von bestimmten Messdaten von einem Oszilloskop über eine LXI-

Verbindung und zeigen den Ablauf der Messung sowie die Behandlung von Fehlern an.

6.2 Darstellung der Ergebnisse

Nachdem der Programmcode die Verbindung zwischen Oszilloskop und Computer hergestellt hat, misst das Oszilloskop die Messdaten des Recksignals. Und führt die FFT-Analyse durch. Wenn die Messungen abgeschlossen sind, werden die Messergebnisse auf dem Computerbildschirm direkt angezeigt (Vgl. Abb.9 Abb. 10). Damit ist die Aufgabestellung vollständig gelöst.

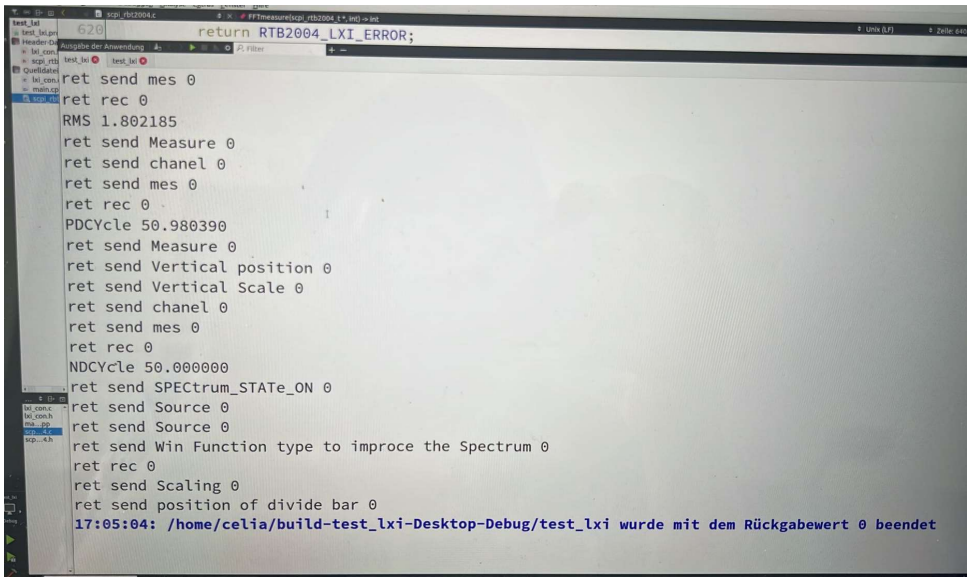


Abbildung 14:Anzeige des Messergebnisses der Frequenz auf dem Bildschirm des Computers.

Am Ende der Codeausführung wird das FFT-Spektrum auf dem Bildschirm des Oszilloskops angezeigt. (siehe Abb. 14)

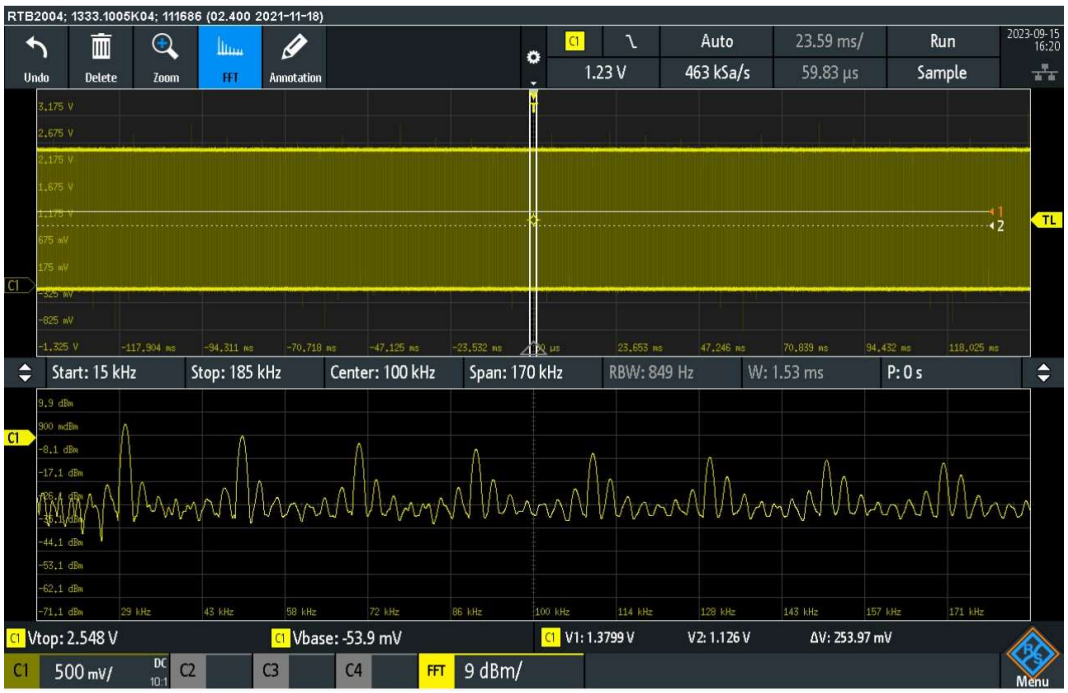


Abbildung 15: FFT-Spektrum mit Frequenzeinstellungen

7 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurden umfassende Messungen an einem Oszilloskop durchgeführt. Dabei wurden nicht nur die Frequenz und der Effektiv- sowie Mittelwert gemessen, sondern auch der Duty-Cycle-Abtastgrad ermittelt. Zusätzlich wurde eine FFT-Analyse durchgeführt, um detaillierte Einblicke in das Frequenzspektrum des Signals zu erhalten. Diese Messungen wurden erfolgreich durch die Auslesung des Oszilloskops über Ethernet mithilfe von Qt Creator und der LXI-Bibliothek realisiert. Ein wichtiger Schritt war die Einbindung der LXI-Bibliothek, welche die Kommunikation mit dem Oszilloskop über Ethernet ermöglicht. Durch die Verwendung der Bibliothek konnten Verbindungen hergestellt, Befehle gesendet und Messdaten abgerufen werden. Zusätzlich erfolgte eine Einarbeitung in die SPCI-Befehlsdefinitionen des Oszilloskops vertraut gemacht, um die korrekten Befehle zur Steuerung und Auslesung der gewünschten Messdaten zu verwenden. Die erfolgreiche Auslesung des Oszilloskops über Ethernet eröffnet vielfältige Möglichkeiten und Anwendungsbereiche. Durch die Fernsteuerung und Fernüberwachung von Oszilloskopen können Messungen und Analysen in entfernten Umgebungen durchgeführt werden. Dies kann besonders nützlich in Bereichen wie der Industrieautomatisierung, Fehlerdiagnose und Netzwerkanalyse sein. Insgesamt bietet die Auslesung eines Oszilloskops über Ethernet mithilfe von Qt Creator und der LXI-Bibliothek eine flexible und leistungsstarke Lösung zur Fernsteuerung und Datenerfassung. Die Kombination aus Ethernet, SCPI und einer leistungsfähigen Programmierumgebung bietet ein großes Potenzial für die Weiterentwicklung von Oszilloskop-Anwendungen. In verschiedenen Bereichen wie Elektronikentwicklung, Signalanalyse, Telekommunikation und mehr können präzise Messungen, detaillierte Signalanalysen und automatisierte Tests durchgeführt werden.

8 Quellenverzeichnis

[1]https://www.rohde-schwarz.com/de/produkte/messtechnik/oszilloskope_63663.html?mid=10913&midx=generic-scopes-measure_generic-scopes_search_text-ad_de_&kw=messen%20mit%20dem%20oszilloskop

[2]https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/pdm/cl_manuals/user_manual/1333_1611_01/RTB_UserManual_de_12.pdf

[3] https://www.giga.de/artikel/was-ist-ethernet-einfach-erklaert/

[4] HTTPS://GITHUB.COM/LXI-TOOLS/LIBLXI

[5] HTTPS://PIXABAY.COM/DE/PHOTOS/NETZWERKKABEL-NETZWERK-KABEL-DATEN-2245837/

[6] https://www.oszilloskope.net/tastkopf/

[7]https://de.wikipedia.org/wiki/Rechteckschwingung#:~:text=Das%20Rechtecksignal%20bzw.,Zeit%20einen%20rechteckigen%20Verlauf%20aufweist.

[8] https://www.elektronik-kompendium.de/sites/com/2503191.htm

[9] https://www.avast.com/de-de/c-virtual-machine#topic-4

[10]https://de.wikipedia.org/wiki/Standard_Commands_for_Programmable_Instruments

[11] https://www.redhat.com/de/topics/api/what-are-application-programming-interfaces#:~:text=APIs%20(Application%20Programming%20Interfaces%20oder,zu%20kommunizieren%20und%20Daten%20auszutauschen.

[12] https://ubuntu.com/tutorials/how-to-run-ubuntu-desktop-on-a-virtual-machine-using-virtualbox#1-overview

[13] https://vitux.com/compiling-your-first-qt-program-in-ubuntu/

[14] https://github.com/lxi-tools/liblxi

[15] https://www.fr.linuxfromscratch.org/view/blfs-svn/basicnet/libtirpc.html

[16] https://sourceforge.net/projects/libtirpc/

[17] https://studyflix.de/ingenieurwissenschaften/frequenz-5716

[18] https://www.biancahoegel.de/physik/welle/frequenz.html

[19] https://micon-l.de/FB_DE_Rechtecksignal.html

[20] https://de.wikipedia.org/wiki/Gleichrichtwert

[21]https://www.smart-testsolutions.de/de/glossar/begriff/duty-cycle-messung.html

[22]https://www.tmatlantic.com/encyclopedia/index.php?ELEMENT_ID=9192

[23]https://www.elektronikpraxis.de/fast-fourier-transformation-frequenzanalyse-mittels-fft-in-der-praxis-a-9c9758b581b272c528972ad86eb59cef/

[24]https://www.nti-audio.com/de/service/wissen/fast-fourier-transformation-fft

[25] http://www.stefanfrings.de/qt_lernen/index.html

[26] https://wiki.qt.io/File:Qtcreator-welcome.png

[27] https://www.elektrotechnik-einfach.de/oszilloskop/

[28]https://www.redhat.com/de/topics/api/what-are-application-programming-interfaces#:~:text=APIs%20(Application%20Programming%20Interfaces%20oder,zu%20kommunizieren%20und%20Daten%20auszutauschen.

[29] https://www.wikiwand.com/de/Frequenz

[30]https://www.schullv.de/mathe/basiswissen/bw/analysis/integral/mittelwert

9 Anhang

9.1 Code

9.1.1 Header-Dateien

- Lxi_con.h

```
#ifndef LXI_CON_H_
#define LXI_CON_H_

#include <lxi.h>   // Einbinden der LXI-Bibliothek


#define DEFAULT_TIMEOUT 100   // Standard-Timeout-Wert für die
Verbindung

typedef struct
{
    int handler;  // Handler zur Verbindung
    char ipv4[16]; // IPv4-Adresse des Geräts
    int timeout;   // Timeout-Wert für die Verbindung
    int port;      // Portnummer
    lxi_protocol_t protocol;  // LXI-Protokoll

} lxi_connection_t;      // Struktur zur Repräsentation der LXI-
Verbindung


int lxi_con_init(lxi_connection_t * con);       //Diese Funktion
initialisiert eine LXI-Verbindung
int lxi_con_deinit(lxi_connection_t * con);     //Diese Funktion
deinitialisiert eine LXI-Verbindung.
int lxi_con_send(lxi_connection_t * con, char * msg, int len);
int lxi_con_receive(lxi_connection_t * con, char * msg, int * len,
int max_len);


#endif /* LXI_CON_H_ */
```

- scpi_rtb2004.h

```
#ifndef SCPI_RTB2004_H
#define SCPI_RTB2004_H

#include "lxi_con.h"

#define RTB2004_ERROR -1
#define RTB2004_INVALID_PARAMETER -2
#define RTB2004_LXI_ERROR -3
```

```
#define RTB2004_COM_ERROR -4
#define RTB2004_CHANNEL_SELECDT_ERROR -5


typedef struct {
    lxi_connection_t * lxi_connection;
} scpi_rtb2004_t;

int init_rtb2004(scpi_rtb2004_t* dev);

int  rtb2004_read_id(scpi_rtb2004_t*  dev,  char  *  id_message,  int
max_length);


int  rtb2004_meas_frequency_measurement(scpi_rtb2004_t*  dev,  int
channel,int measure, double *result);

int rtb2004_meas_MEAN_measurement(scpi_rtb2004_t* dev, int channel,
int measure, double *result);
int   rtb2004_meas_RMS_measurement   (scpi_rtb2004_t*   dev,   int
channel,int measure, double *result);
int   rtb2004_meas_PDCYcle_measurement(scpi_rtb2004_t*   dev,   int
channel,int measure, double *result);
int   rtb2004_meas_NDCYcle_measurement(scpi_rtb2004_t*   dev,   int
channel,int measure, double *result);

int FFTmeasure(scpi_rtb2004_t* dev, int channel);
#endif // SCPI_RTB2004_H
```

9.1.2 Quelldateien

- Lxi_con.c

```
#include <assert.h>
#include <stdlib.h>
#include <lxi.h>
#include "lxi_con.h"



int lxi_con_init(lxi_connection_t * con)
{
assert(con != NULL);
// Überprüfen, ob der Zeiger 'con' nicht NULL ist

assert(con->port >= 0);
// Überprüfen, ob der Port-Wert nicht negativ ist

assert(con->timeout >= 0);
// Überprüfen, ob der Timeout-Wert nicht negativ ist


lxi_init();
// LXI-Bibliothek initialisieren

// Verbindung zum Gerät herstellen
```

```
// Verbindet sich mit dem Gerät über die angegebenen Parameter in
der lxi_connection_t-Struktur.
// Der Rückgabewert wird in der Variable "res" gespeichert.

int res = lxi_connect(con->ipv4, con->port, "inst0", con->timeout,
con->protocol);

if(res != LXI_ERROR)
// Überprüft, ob die Verbindung erfolgreich hergestellt wurde.
{

con->handler = res;
//   Speichert   den   Handler-Wert   der   Verbindung   in   der
lxi_connection_t-Struktur.
return 0;
//  Gibt  0  zurück,  um  anzuzeigen,  dass  die  Initialisierung
erfolgreich war.

}
else

{

// Bei Verbindungsfehler den Handler-Wert auf -1 setzen und -1
zurückgeben
con->handler = -1;
//  Setzt  den  Handler-Wert  auf  -1,  um  anzuzeigen,  dass  die
Initialisierung fehlgeschlagen ist.
return -1;
//  Gibt  -1  zurück,  um  einen  Fehler  bei  der  Initialisierung
anzugeben.
}
}

int lxi_con_deinit(lxi_connection_t * con)
//Diese Funktion deinitialisiert eine LXI-Verbindung.
{
// Überprüfen, ob der Zeiger 'con' nicht NULL ist
assert(con != NULL);
//  Überprüft,  ob  der  Zeiger  auf  eine  gültige  lxi_connection_t-
Struktur zeigt.

// Versuchen, die Verbindung zu trennen
if(lxi_disconnect(con->handler) != LXI_OK)
// Trennt die Verbindung über den gespeicherten Handler-Wert in der
lxi_connection_t-Struktur.
{
// Wenn die Trennung fehlschlägt, einen erneuten Versuch unternehmen
if(lxi_disconnect(con->handler) != LXI_OK)
{
return -1;
// Bei Fehlschlag -1 zurückgeben
}
else
{
con->handler = -1;
// Setzt den Handler-Wert auf -1, um anzuzeigen, dass die Verbindung
getrennt wurde.
return 0;
```

```
// Gibt 0 zurück, um anzuzeigen, dass die Deinitialisierung
erfolgreich war.
}
}else
{
con->handler = -1;
// Den Handler-Wert auf -1 setzen
return 0;
// Bei Erfolg 0 zurückgeben
}
}

//Diese Funktion sendet Daten über eine LXI-Verbindung
int lxi_con_send(lxi_connection_t * con, char * msg, int len)
{
assert(con != NULL);
// Überprüfen, ob der Zeiger 'con' nicht NULL ist
assert(con->handler >= 0);
// Überprüfen, ob der Handler-Wert nicht kleiner als 0 ist
assert(msg != NULL);
// Überprüfen, ob der Zeiger 'msg' nicht NULL ist
assert(len > 0);
// Überprüfen, ob die Länge 'len' größer als 0 ist

// Sendet die Daten aus dem Puffer "msg" über die Verbindung mit
dem gespeicherten Handler-Wert in der lxi_connection_t-Struktur.
// Der Rückgabewert wird in der Variablen "ret" gespeichert.

int ret = lxi_send(con->handler, msg, len, con->timeout);
// Überprüfen, ob beim Senden ein Fehler aufgetreten ist oder die
Anzahl der gesendeten Länge nicht mit 'len' übereinstimmt
if( ret == LXI_ERROR || ret < len || ret > len+1)
{
return -1;
// Bei Fehlschlag -1 zurückgeben
}else
{
return 0;
// Bei Erfolg 0 zurückgeben
}
}

//Diese Funktion empfängt Daten über eine LXI-Verbindung.

int lxi_con_receive(lxi_connection_t * con, char * msg, int * len,
int max_len)
{
assert(con != NULL);
// Überprüft, ob der Zeiger auf eine gültige lxi_connection_t-
Struktur zeigt.
assert(con->handler >= 0);
// Überprüft, ob der Handler-Wert in der lxi_connection_t-Struktur
gültig ist.
assert(msg != NULL);
// Überprüft, ob der Zeiger auf einen gültigen Puffer zeigt.
assert(len != NULL);
// Überprüft, ob der Zeiger auf eine gültige Variable zeigt.
assert(max_len > 0);
// Überprüft, ob max_len größer als 0 ist.
```

```
// Empfängt Daten über die Verbindung mit dem gespeicherten Handler-Wert in der lxi_connection_t-Struktur.
// Die Rückgabe des Empfangsvorgangs wird in der Variable "ret" gespeichert.
int ret = lxi_receive(con->handler, msg, max_len, con->timeout);
// Ruft die lxi_receive-Funktion auf, um Daten über die Verbindung zu empfangen.
if(ret == LXI_ERROR)
// Überprüft, ob ein Fehler beim Empfangen aufgetreten ist.
{
*msg = 0;
// Setzt den Wert des Puffers auf 0, um ihn zu leeren.
*len = 0;
// Setzt den Wert der Länge auf 0, um anzuzeigen, dass keine Daten empfangen wurden.
return -1;
// Gibt -1 zurück, um einen Fehler beim Empfangen der Daten anzugeben
}
else
{
*len = ret;
// Aktualisiert den Wert der Länge mit der tatsächlichen Länge der empfangenen Daten.
return 0;
// Gibt 0 zurück, um anzuzeigen, dass der Empfangsvorgang erfolgreich war.
}

}
```

- scpi_rtb2004.c

```
#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "scpi_rtb2004.h"
#include "lxi_con.h"

// diese Funktion wird verwendet , um eine Initialisierung eines Geräts
//oder einer Hardware-Komponente namens "rtb2004" durchzuführen,
//indem sie einen Pointer auf die entsprechende Struktur "dev" verwendet.

int init_rtb2004(scpi_rtb2004_t* dev){
    assert(dev != NULL);     // überprüft, ob der Zeiger dev nicht auf NULL zeigt.
    assert(dev->lxi_connection != NULL);   // überprüft, ob der Zeiger dev->lxi_connection nicht auf NULL zeigt
    assert(dev->lxi_connection->handler >= 0);  //überprüft, ob der Wert von dev->lxi_connection->handler größer oder gleich 0 ist.


    char msg[60]; //deklaration eines lokalen Zeichenpuffer namens msg mit einer Größe von 60 Zeichen.
```

```
    int length; //deklaration einer lokale Variable namens length
vom Datentyp int.
    int ret;  // deklaration einer lokale Variable namens ret vom
Datentyp int

    // Bildschirm löschen und alte Messungen entfernen

    length = snprintf(msg, 60,"DISPlay:CLEar:SCReen");
    ret = lxi_con_send(dev->lxi_connection, msg, length);
    if(ret != 0)
        return RTB2004_LXI_ERROR;

    return 0;
}

//Die Funktion liest die ID des RTB2004-Geräts, indem sie den
entsprechenden Befehl an das LXI-Gerät sendet.
//Die empfangene ID wird nicht direkt zurückgegeben, sondern in den
übergebenen id_message-Puffer geschrieben,
//damit der Aufrufer darauf zugreifen kann.

int rtb2004_read_id(scpi_rtb2004_t* dev, char * id_message, int
max_length){
    assert(dev != NULL);
// überprüft, ob der Zeiger dev nicht auf NULL zeigt.
    assert(dev->lxi_connection != NULL);
// überprüft, ob der Zeiger dev->lxi_connection nicht auf NULL zeigt
    assert(dev->lxi_connection->handler >= 0);
//überprüft, ob der Wert von dev->lxi_connection->handler größer
oder gleich 0 ist.

    char lxi_msg[10];
//deklaration einen lokalen Zeichenpuffer namens lxi_msg mit einer
Größe von 10 Zeichen
    int length;
//deklaration einer lokale Variable namens length vom Datentyp int.
    int ret;
// deklaration einer lokale Variable namens ret vom Datentyp int


//senden IDN-Anfrage an das Gerät und überprüfen anschließend den
Erfolg des Sendevorgangs.
    length = snprintf(lxi_msg, 10,"*IDN?");
    ret = lxi_con_send(dev->lxi_connection, lxi_msg, length);
    if(ret != 0)
        return RTB2004_LXI_ERROR;

    //read response from scope
    ret = lxi_con_receive(dev->lxi_connection, id_message, &length,
max_length);

    //check return value
    if(ret != 0)
        return RTB2004_LXI_ERROR;
    if(length <= 1)
        return RTB2004_COM_ERROR;
    if(length == 60)
        return RTB2004_COM_ERROR;


    return 0;
```

```
}

//FREQuency

int  rtb2004_meas_frequency_measurement(scpi_rtb2004_t*  dev,  int
channel, int measure, double *result){

    // Überprüfen der Vorbedingungen
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(result != NULL);

    // Deklaration der benötigten Variablen


    char lxi_msg01[40];
    int length01;
    int ret01;
   // Erstellen des Befehls zum Aktivieren der angegebenen Messung
   //MEASurement<m>[:ENABle] <State>
   length01   =   snprintf(lxi_msg01,   40,   "MEASurement%d:ENABle
ON",measure);

   // send command to scope
    ret01 = lxi_con_send(dev->lxi_connection, lxi_msg01, length01);

    printf("ret send Measure %i\n", (ret01));
    if(ret01 != 0)
        return RTB2004_LXI_ERROR;


    char lxi_msg0[40];
    int length0;
    int ret0;

    // Erstellen des Befehls zum Aktivieren des angegebenen Kanals
    //  Der  Kanalindex  wird  in  den  Befehl  eingefügt,  um  den
spezifischen Kanal anzusprechen
    length0 = snprintf(lxi_msg0, 40, "CHANnel%d:STATe ON",channel);

    // send command to scope
    ret0 = lxi_con_send(dev->lxi_connection, lxi_msg0, length0);

    printf("ret send chanel %i\n", (ret0));
    if(ret0 != 0)
        return RTB2004_LXI_ERROR;

    char lxi_msg[40];
    int length;
    int ret;

    // Setzt den Inhalt des char-Arrays 'lxi_msg' mithilfe von
snprintf fest
    // Der Inhalt wird durch den Wert von 'channel' ersetzt


    length           =           snprintf(lxi_msg,           40,
"MEASurement%i:RESult:ACTual?FREQuency",channel);
```

```
    // send command to scope
    ret = lxi_con_send(dev->lxi_connection, lxi_msg, length);
    printf("ret send mes %i\n", (ret));

    //read response from scope
    ret = lxi_con_receive(dev->lxi_connection, lxi_msg, &length,
40);

    printf("ret rec %i\n", ret);
    if(ret != 0)
        return RTB2004_LXI_ERROR;

    //convert response to double
    *result = atof(lxi_msg);
    printf("FREQuency %f\n", atof(lxi_msg));


    return 0;
}

//Mean

int rtb2004_meas_MEAN_measurement(scpi_rtb2004_t* dev, int channel,
int measure, double *result){

    // Überprüfen der Vorbedingungen
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(result != NULL);

    // Deklaration der benötigten Variablen


    char lxi_msg01[40];
    int length01;
    int ret01;
   // Erstellen des Befehls zum Aktivieren der angegebenen Messung
   //MEASurement<m>[:ENABle] <State>
    length01 = snprintf(lxi_msg01, 40, "MEASurement%d:ENABle
ON",measure);

    // send command to scope
    ret01 = lxi_con_send(dev->lxi_connection, lxi_msg01, length01);

    printf("ret send Measure %i\n", (ret01));
    if(ret01 != 0)
        return RTB2004_LXI_ERROR;


    char lxi_msg0[40];
    int length0;
    int ret0;

    // Erstellen des Befehls zum Aktivieren des angegebenen Kanals
    // Der Kanalindex wird in den Befehl eingefügt, um den
spezifischen Kanal anzusprechen
    length0 = snprintf(lxi_msg0, 40, "CHANnel%d:STATe ON",channel);

    // send command to scope
    ret0 = lxi_con_send(dev->lxi_connection, lxi_msg0, length0);
```

```
    printf("ret send chanel %i\n", (ret0));
    if(ret0 != 0)
        return RTB2004_LXI_ERROR;

    char lxi_msg[40];
    int length;
    int ret;

    // Setzt den Inhalt des char-Arrays 'lxi_msg' mithilfe von
snprintf fest
    // Der Inhalt wird durch den Wert von 'channel' ersetzt
    length          =          snprintf(lxi_msg,          40,
"MEASurement%d:RESult:ACTual?MEAN",channel);


    // send command to scope
    ret = lxi_con_send(dev->lxi_connection, lxi_msg, length);
    printf("ret send mes %i\n", (ret));

    //read response from scope
    ret  =  lxi_con_receive(dev->lxi_connection,  lxi_msg,  &length,
40);

    printf("ret rec %i\n", ret);
    if(ret != 0)
        return RTB2004_LXI_ERROR;

    //convert response to double
    *result = atof(lxi_msg);
    printf("MEAN %f\n", atof(lxi_msg));


    return 0;
}


//RMS

int rtb2004_meas_RMS_measurement(scpi_rtb2004_t* dev, int channel,
int measure, double *result){

    // Überprüfen der Vorbedingungen
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(result != NULL);

    // Deklaration der benötigten Variablen


    char lxi_msg01[40];
    int length01;
    int ret01;
   // Erstellen des Befehls zum Aktivieren der angegebenen Messung
     //MEASurement<m>[:ENABle] <State>
    length01   =   snprintf(lxi_msg01,   40,   "MEASurement%d:ENABle
ON",measure);

    // send command to scope
    ret01 = lxi_con_send(dev->lxi_connection, lxi_msg01, length01);
```

```
    printf("ret send Measure %i\n", (ret01));
    if(ret01 != 0)
        return RTB2004_LXI_ERROR;


    char lxi_msg0[40];
    int length0;
    int ret0;

   // Erstellen des Befehls zum Aktivieren des angegebenen Kanals
  //  Der  Kanalindex  wird  in  den  Befehl  eingefügt,  um  den
spezifischen Kanal anzusprechen
    length0 = snprintf(lxi_msg0, 40, "CHANnel%d:STATe ON",channel);

   // send command to scope
    ret0 = lxi_con_send(dev->lxi_connection, lxi_msg0, length0);

    printf("ret send chanel %i\n", (ret0));
    if(ret0 != 0)
        return RTB2004_LXI_ERROR;

    char lxi_msg[40];
    int length;
    int ret;

 // Setzt den Inhalt des char-Arrays 'lxi_msg' mithilfe von snprintf
fest
 // Der Inhalt wird durch den Wert von 'channel' ersetzt
    length           =           snprintf(lxi_msg,           40,
"MEASurement%d:RESult:ACTual?RMS",channel);


   // send command to scope
    ret = lxi_con_send(dev->lxi_connection, lxi_msg, length);
    printf("ret send mes %i\n", (ret));

    //read response from scope
    ret  =  lxi_con_receive(dev->lxi_connection,  lxi_msg,  &length,
40);

    printf("ret rec %i\n", ret);
    if(ret != 0)
        return RTB2004_LXI_ERROR;

    //convert response to double
    *result = atof(lxi_msg);
    printf("RMS %f\n", atof(lxi_msg));


    return 0;
}

//PDCYcle

int   rtb2004_meas_PDCYcle_measurement(scpi_rtb2004_t*   dev,   int
channel, int measure, double *result){

    // Überprüfen der Vorbedingungen
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
```

```
    assert(dev->lxi_connection->handler >= 0);
    assert(result != NULL);

    // Deklaration der benötigten Variablen


    char lxi_msg01[40];
    int length01;
    int ret01;
     // Erstellen des Befehls zum Aktivieren der angegebenen Messung
    //MEASurement<m>[:ENABle] <State>
    length01   =   snprintf(lxi_msg01,   40,   "MEASurement%d:ENABle
ON",measure);

    // send command to scope
    ret01 = lxi_con_send(dev->lxi_connection, lxi_msg01, length01);

    printf("ret send Measure %i\n", (ret01));
    if(ret01 != 0)
        return RTB2004_LXI_ERROR;


    char lxi_msg0[40];
    int length0;
    int ret0;

    // Erstellen des Befehls zum Aktivieren des angegebenen Kanals
    //  Der  Kanalindex  wird  in  den  Befehl  eingefügt,  um  den
spezifischen Kanal anzusprechen
    length0 = snprintf(lxi_msg0, 40, "CHANnel%d:STATe ON",channel);

    // send command to scope
    ret0 = lxi_con_send(dev->lxi_connection, lxi_msg0, length0);

    printf("ret send chanel %i\n", (ret0));
    if(ret0 != 0)
        return RTB2004_LXI_ERROR;

    char lxi_msg[40];
    int length;
    int ret;

    //  Setzt  den  Inhalt  des  char-Arrays  'lxi_msg'  mithilfe  von
snprintf fest
    // Der Inhalt wird durch den Wert von 'channel' ersetzt
    length           =           snprintf(lxi_msg,           40,
"MEASurement%d:RESult:ACTual?PDCYcle",channel);


    // send command to scope
    ret = lxi_con_send(dev->lxi_connection, lxi_msg, length);
    printf("ret send mes %i\n", (ret));

    //read response from scope
    ret  =  lxi_con_receive(dev->lxi_connection,  lxi_msg,  &length,
40);

    printf("ret rec %i\n", ret);
    if(ret != 0)
        return RTB2004_LXI_ERROR;
```

```
    //convert response to double
    *result = atof(lxi_msg);
    printf("PDCYcle %f\n", atof(lxi_msg));


    return 0;
}


// NDCYcle

int   rtb2004_meas_NDCYcle_measurement(scpi_rtb2004_t*   dev,   int
channel, int measure, double *result){

    // Überprüfen der Vorbedingungen
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(result != NULL);

    // Deklaration der benötigten Variablen


    char lxi_msg01[40];
    int length01;
    int ret01;
     // Erstellen des Befehls zum Aktivieren der angegebenen Messung
     //MEASurement<m>[:ENABle] <State>
    length01   =   snprintf(lxi_msg01,   40,   "MEASurement%d:ENABle
ON",measure);

    // send command to scope
    ret01 = lxi_con_send(dev->lxi_connection, lxi_msg01, length01);

    printf("ret send Measure %i\n", (ret01));
    if(ret01 != 0)
        return RTB2004_LXI_ERROR;


    char lxi_msg0[40];
    int length0;
    int ret0;

   // Erstellen des Befehls zum Aktivieren des angegebenen Kanals
  //  Der  Kanalindex  wird  in  den  Befehl  eingefügt,  um  den
spezifischen Kanal anzusprechen
    length0 = snprintf(lxi_msg0, 40, "CHANnel%d:STATe ON",channel);

    // send command to scope
    ret0 = lxi_con_send(dev->lxi_connection, lxi_msg0, length0);

    printf("ret send chanel %i\n", (ret0));
    if(ret0 != 0)
        return RTB2004_LXI_ERROR;

    char lxi_msg[40];
    int length;
    int ret;

    // Setzt den Inhalt des char-Arrays 'lxi_msg' mithilfe von
snprintf fest
```

```
    // Der Inhalt wird durch den Wert von 'channel' ersetzt
    length           =           snprintf(lxi_msg,           40,
"MEASurement%d:RESult:ACTual?NDCYcle",channel);


    // send command to scope
    ret = lxi_con_send(dev->lxi_connection, lxi_msg, length);
    printf("ret send mes %i\n", (ret));

    //read response from scope
    ret  =  lxi_con_receive(dev->lxi_connection,  lxi_msg,  &length,
40);

    printf("ret rec %i\n", ret);
    if(ret != 0)
        return RTB2004_LXI_ERROR;

    //convert response to double
    *result = atof(lxi_msg);
    printf("NDCYcle %f\n", atof(lxi_msg));


    return 0;
}



//FREQuency */
int FFTmeasure(scpi_rtb2004_t* dev, int channel)
{
    // Überprüfen der Vorbedingungen
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);

    // Deklaration der benötigten Variablen
    char SpectrumON[30];
    int lengthspON;
    int retspON;


    //FRQSPEKTRUM

    //1-FFT-Menü öffnen
    lengthspON = snprintf(SpectrumON, 30, "SPECtrum:STATe ON");

    // send command to scope
      retspON   =   lxi_con_send(dev->lxi_connection,   SpectrumON,
      lengthspON);
      retspON   =   lxi_con_send(dev->lxi_connection,   SpectrumON,
      lengthspON);
  //retspON    =    lxi_con_send(dev->lxi_connection,    SpectrumON,
lengthspON);

    printf("ret send SPECtrum_STATe_ON %i\n", (retspON));
    if(retspON != 0)
        return RTB2004_LXI_ERROR;


    //2- Typ der FFT-Fenster auswählen
```

```
    char lxi_msgS[40];
    int lengthS;
    int retS;


    // Erstellen des Befehls zum Aktivieren des angegebenen Kanals:
SPECtrum:SOURce
    //  Der  Kanalindex  wird  in  den  Befehl  eingefügt,  um  den
spezifischen Kanal anzusprechen
    lengthS  =  snprintf(lxi_msgS,  40,  "SPECtrum:SOURce  CH%d",
channel);

    // send command to scope
    retS = lxi_con_send(dev->lxi_connection, lxi_msgS, lengthS);

    printf("ret send Source %d\n", (retS));
    if(retS != 0)
        return RTB2004_LXI_ERROR;
    lengthS  =  snprintf(lxi_msgS,  40,  "SPECtrum:SOURce  CH%d",
channel);

    // send command to scope
    retS = lxi_con_send(dev->lxi_connection, lxi_msgS, lengthS);

    printf("ret send Source %d\n", (retS));
    if(retS != 0)
        return RTB2004_LXI_ERROR;


    // Fenstertyp für die Frequenzbereichsanalyse (FFT) auf einem
Oszilloskop festlegen
    //window functions to improve the spectrum analysis
    char lxi_msgW[40];
    //mögliche Funktion Type
    char TypeR[30] = "RECTangular";
    char TypeHam[30] ="HAMMing";
    char TypeHan[30] = "HANNing";
    char TypeB[30] ="Blackmanharris";
    char TypeF[30] = "FLATtop";
    int lengthW;
    int retW;

        // Setzt den Inhalt des char-Arrays 'lxi_msg' mithilfe von
      snprintf fest
    // Der Inhalt wird durch den Wert von 'channel' ersetzt
    lengthW          =          snprintf(lxi_msgW,          40,
"SPECtrum:FREQuency:WINDow:TYPE %s",TypeHam);


    // send command to scope
    retW = lxi_con_send(dev->lxi_connection, lxi_msgW, lengthW);
    printf("ret  send  Win  Function  type  to  improce  the  Spectrum
%i\n", (retW));

    //read response from scope

    printf("ret rec %i\n", retW);
    if(retW != 0)
        return RTB2004_LXI_ERROR;

    //Defines the Scaling unit of the y-axis
```

```
    char MagScal1[10] ="LINear";
//Skalierung, zeigt den Effektivwert der Spannung an.
    char MagScal2[10] ="DBM";
//Logarithmische Skalierung, bezogen auf 1 mW.
    char MagScal3[10] ="DBV";
//Logarithmische Skalierung, bezogen auf 1 Veff.
    char MagScal4[10] ="DBUV";
//Logarithmische Skalierung; bezogen auf 1 µVeff.
    int lengthMS;
    int retMS;
    int retMSr;
    char lxi_msgMS[40];

//  Skalierung  der  Y-Achse  für  die  FFT  auf  "logarithmische
Skalierung"
    lengthMS          =          snprintf(lxi_msgMS,          40,
"SPECtrum:FREQuency:MAGNitude:SCALe %d",MagScal2);

    // send command to scope
    retMS = lxi_con_send(dev->lxi_connection, lxi_msgMS, lengthMS);

    printf("ret send Scaling %d\n", (retMS));
    if(retMS != 0)
        return RTB2004_LXI_ERROR;

    //Defines the vertical position of the spectrum
    int Position = 30;
    int lengthVP;
    int retVP;
    char lxi_msgVP[40];

    lengthVP = snprintf(lxi_msgVP, 40, "SPECtrum:FREQuency:POSition
%d div" , Position);

    // send command to scope
    retVP = lxi_con_send(dev->lxi_connection, lxi_msgVP, lengthVP);

    printf("ret send Vertical position %d\n", (retVP));
    if(retMS != 0)
        return RTB2004_LXI_ERROR;

    //Defines the vertical Scale of the spectrum
    int lengthSP;
    int retSP;
    char lxi_msgSP[40];

    //Parameter:  <Scale>  Range  values  and  unit  depend  on
SPECtrum:FREQuency: MAGNitude:SCALe.

    //lengthSP = snprintf(lxi_msgSP, 40, "SPECtrum:FREQuency:SCALe
%d", ScaleValue); mit int ScaleValue = 40; a Tester

    lengthSP  =  snprintf(lxi_msgSP,  40,  "SPECtrum:FREQuency:SCALe
%d", (retMS/10));

    // send command to scope
    retSP = lxi_con_send(dev->lxi_connection, lxi_msgSP, lengthSP);

    printf("ret send Vertical Scale %d\n", (retSP));
    if(retSP != 0)
        return RTB2004_LXI_ERROR;
```

```
    //Defines the position of the divide bar between normal waveform
and FFT window
    //Position der Trennlinie einstellen. einheit für die Position
ist Pixel.
    // die vertikale anzeigegröße beträgt 800px
    int DividerPosition = 400;
    int lengthPP;
    int retPP;
    char lxi_msgPP[40];

     lengthPP = snprintf(lxi_msgPP, 40, "DISPlay:CBAR:FFT:POSition
%d", DividerPosition );

    // read response von  scope

       retPP    =    lxi_con_send(dev->lxi_connection,    lxi_msgPP,
      lengthPP);
      printf("ret send position of divide bar %i\n", retPP);
        if(retPP != 0)
        return RTB2004_LXI_ERROR;

    int startFrequency = 0;
    char lxi_msgStart[40];
    int lengthStart;
    int retStart;

    lengthStart        =        snprintf(lxi_msgStart,        40,
"SPECtrum:FREQuency:STARt %d", startFrequency);

    // Befehl zur Übertragung an das Oszilloskop senden
    retStart   =   lxi_con_send(dev->lxi_connection,   lxi_msgStart,
lengthStart);

    if (retStart != 0) {
        return RTB2004_LXI_ERROR;
    }

    // Setzen der Stoppfrequenz auf 100.000 Hz
    int stopFrequency = 200000;
    char lxi_msgStop[40];
    int lengthStop;
    int retStop;

    lengthStop = snprintf(lxi_msgStop, 40, "SPECtrum:FREQuency:STOP
%d", stopFrequency);

    // Befehl zur Übertragung an das Oszilloskop senden
    retStop    =    lxi_con_send(dev->lxi_connection,    lxi_msgStop,
lengthStop);

    if (retStop != 0) {
        return RTB2004_LXI_ERROR;
    }

    // center
    int centerFrequency = 100000;
    char lxi_msgCenter[40];
    int lengthCenter;
    int retCenter;
```

```
    lengthCenter        =        snprintf(lxi_msgCenter,        40,
"SPECtrum:FREQuency:CENTer %d", centerFrequency);

    // Befehl zur Übertragung an das Oszilloskop senden
    retCenter  =  lxi_con_send(dev->lxi_connection,  lxi_msgCenter,
lengthCenter);

    if (retCenter != 0) {
        return RTB2004_LXI_ERROR;
    }

    //Span

    int spanFrequency = 200000;
    char lxi_msgSpan[40];
    int lengthSpan;
    int retSpan;

    lengthSpan = snprintf(lxi_msgSpan, 40, "SPECtrum:FREQuency:SPAN
%d", spanFrequency);

    // Befehl zur Übertragung an das Oszilloskop senden
    retSpan   =   lxi_con_send(dev->lxi_connection,   lxi_msgSpan,
lengthSpan);

    if (retSpan != 0) {
        return RTB2004_LXI_ERROR;
    }

   // RBW (Auto)

    int lengthRBWAuto;
    char lxi_msgRBWAuto[40];
    int retRBWAuto;

    lengthRBWAuto       =       snprintf(lxi_msgRBWAuto,       40,
"SPECtrum:FREQuency:BANDwidth:AUTO ON");

    // Befehl zur Übertragung an das Oszilloskop senden
    retRBWAuto = lxi_con_send(dev->lxi_connection, lxi_msgRBWAuto,
lengthRBWAuto);

    if (retRBWAuto != 0) {
        return RTB2004_LXI_ERROR;
    }

    // W=2,6 ms

    int timeRangeValue = 2.6;  // Zeit in Millisekunden
    int lengthTimeRange;
    char lxi_msgTimeRange[40];
    int retTimeRange;

    lengthTimeRange      =      snprintf(lxi_msgTimeRange,      40,
"SPECtrum:TIME:RANGe %fms", timeRangeValue);

    // Befehl an das Oszilloskop senden
    retTimeRange        =        lxi_con_send(dev->lxi_connection,
lxi_msgTimeRange, lengthTimeRange);
```

```
    if (retTimeRange != 0) {
        return RTB2004_LXI_ERROR;
    }

    //P=0

    int lengthTimePosition;
    char lxi_msgTimePosition[40];
    int retTimePosition;

    lengthTimePosition    =    snprintf(lxi_msgTimePosition,    40,
"SPECtrum:TIME:POSition %fs", 0.0);

    // Befehl an das Oszilloskop senden
    retTimePosition       =       lxi_con_send(dev->lxi_connection,
lxi_msgTimePosition, lengthTimePosition);

    if (retTimePosition != 0) {
        return RTB2004_LXI_ERROR;
    }


            return 0;
        }
```

- Main.cpp

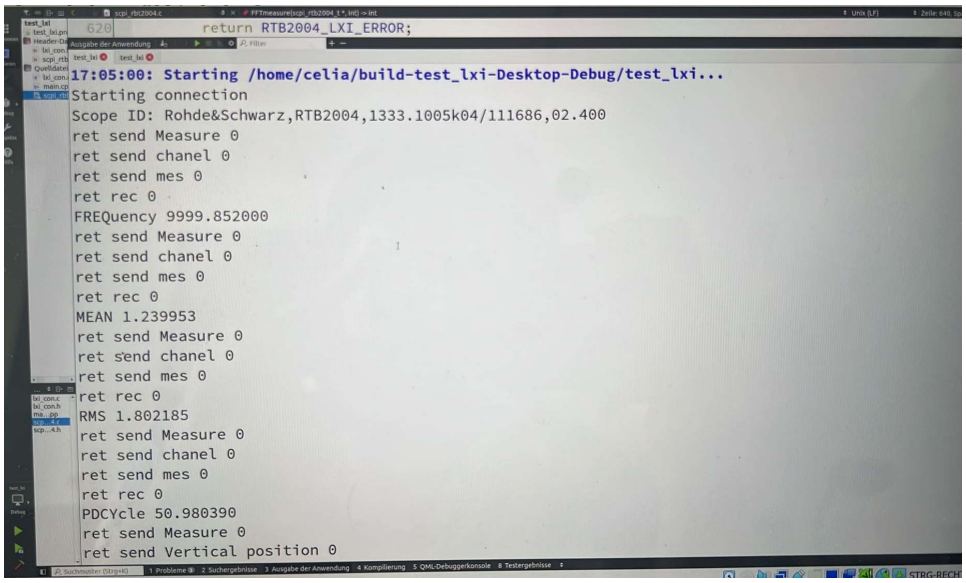
```
#include <QCoreApplication>

#include <iostream>
#include <stdio.h>
#include <string.h>

extern "C" {
#include "lxi_con.h"
#include "scpi_rtb2004.h"
}


int main(int argc, char *argv[]){
printf("Starting connection\n");
// Ausgabe auf der Konsole: "Starting connection
fflush(stdout);     // Ausgabepuffer leeren, um die Ausgabe sofort
auf der Konsole anzuzeigen


// Deklaration der erforderlichen Strukturen (lxi + spezifisch für
das Scope)

lxi_connection_t scope_lxi;
// LXI-Verbindungsstruktur
scpi_rtb2004_t scope_scpi;
// SCPI-Struktur spezifisch für das RTB2004 Scope

// Setzen der LXI-Verbindungsparameter

strcpy(scope_lxi.ipv4, "192.168.0.25");
```

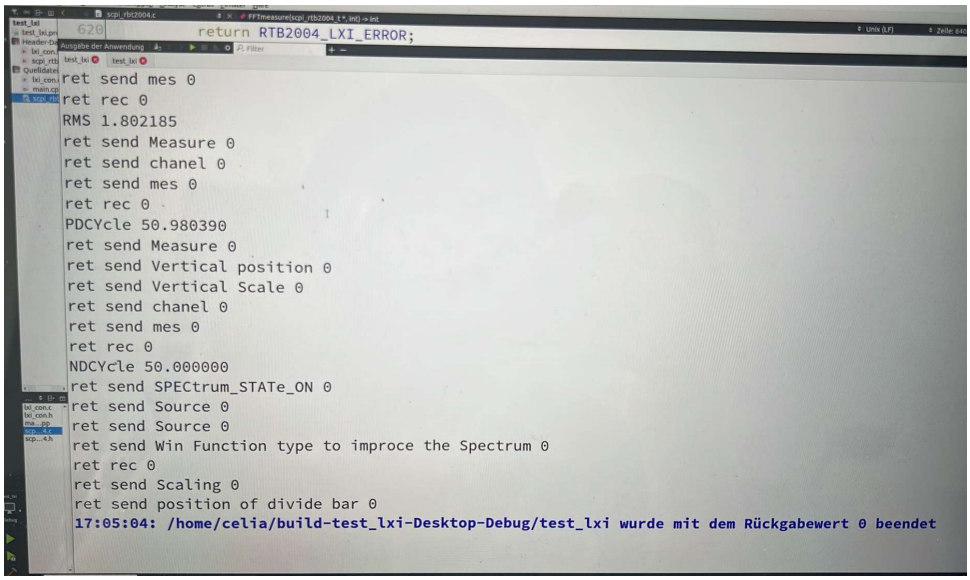
```
// IP-Adresse des Geräts setzen  *
scope_lxi.timeout = 1000;
// Timeout-Wert setzen
scope_lxi.port = 0;
// Portnummer setzen
scope_lxi.protocol = VXI11;
// Verwendetes Protokoll setzen
scope_lxi.handler = 0;
// Handler initialisieren (Standardwert)


//init lxi connection

if(lxi_con_init(&scope_lxi)) {
printf("\nERROR: LXI connection to scope failed.");
fflush(stdout);
return -1;
}

//attach lxi connection to scope struct
scope_scpi.lxi_connection = &scope_lxi;
// Der Zeiger auf das lxi_connection_t-Objekt "scope_lxi" wird der
Variable "scope_scpi.lxi_connection" zugewiesen.

//  Die Initialisierungsfunktion des Scopes wird aufgerufen.
if(init_rtb2004(&scope_scpi)) //Wenn die Initialisierungsfunktion
init_rtb2004 einen Fehler zurückgibt:
{
printf("\nERROR: Init function of scope failed.");
// Fehlermeldung auf der Konsole ausgeben
fflush(stdout);
// Den Ausgabepuffer leeren
return -1;
// Den Fehlercode -1 zurückgeben
}

//read scope id

char id_value[50];
// Deklaration eines Zeichenarrays "id_value" mit einer Größe von
50 Zeichen
if(rtb2004_read_id(&scope_scpi, id_value, 50))
// Wenn die Funktion rtb2004_read_id einen Fehler zurückgibt:
{
printf("\nERROR: ID read of scope failed.");
// Ausgabe einer Fehlermeldung auf der Konsole
fflush(stdout);
// Leeren des Ausgabepuffers
return -1;
// Rückgabe des Fehlercodes -1
}

// Ausgabe der Geräte-ID auf der Konsole

printf("Scope ID: ");
// Ausgabe der Zeichenkette "Scope ID: " auf der Konsole
printf(id_value);
// Ausgabe des Inhalts der Variablen "id_value" als Zeichenkette
auf der Konsole
fflush(stdout);
```

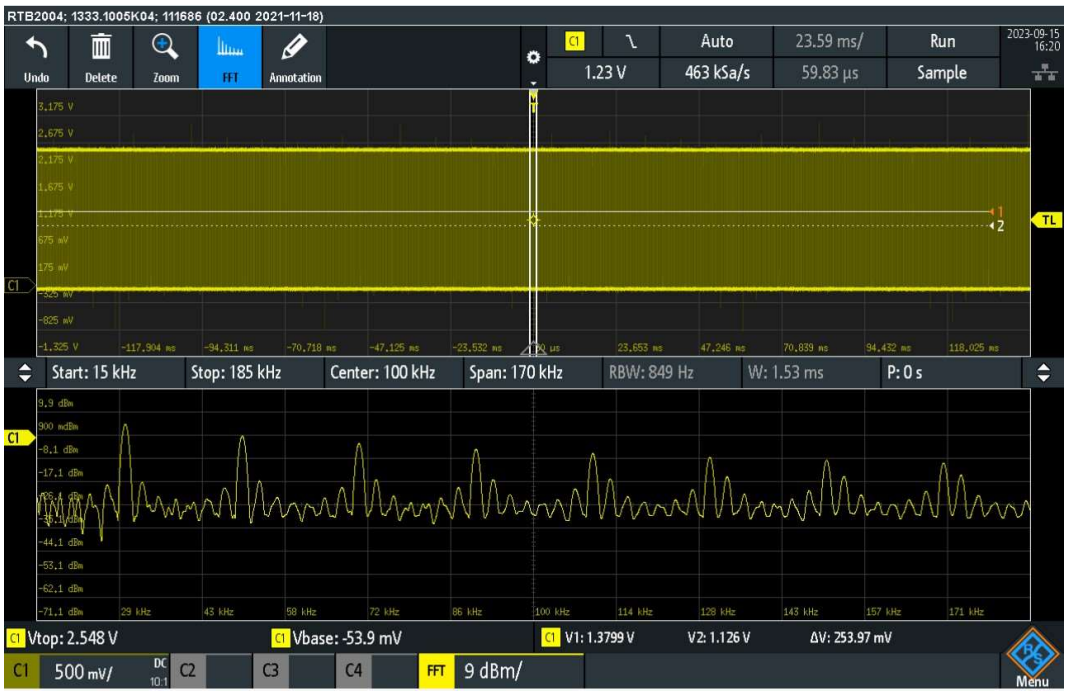
```
// Leeren des Ausgabepuffers, um sicherzustellen, dass die Ausgabe
angezeigt wird

// Messung der Frequenz durchführen
int channel = 1;
int measure = 1;
double resultFreq;
if    (rtb2004_meas_frequency_measurement(&scope_scpi,    channel,
measure, &resultFreq)){
printf("\nERROR:frequency_measurement_all_on  of  scope  failed.int
");
//Ausgabe einer Fehlermeldung auf der Konsole
fflush(stdout);   // Leeren des Ausgabepuffers
return -1;
}

// Messung von MEAN durchführen


double resultMEAN;
if  (rtb2004_meas_MEAN_measurement(&scope_scpi,  channel,  measure,
&resultMEAN)){
printf("\nERROR:MEAN_measurement_all_on  of  scope  failed.int  ");
//Ausgabe einer Fehlermeldung auf der Konsole
fflush(stdout);   // Leeren des Ausgabepuffers
return -1;
}

// Messung von RMS durchführen


double resultRms;
if(rtb2004_meas_RMS_measurement(&scope_scpi,    channel,  measure,
&resultRms)) {
printf("\nERROR:RMS_measurement_all_on  of  scope  failed.int  ");
//Ausgabe einer Fehlermeldung auf der Konsole
fflush(stdout);   // Leeren des Ausgabepuffers
return -1;
}
// Messung von PDCYcle durchführen

double resultPDCYcle;
if   (   rtb2004_meas_PDCYcle_measurement   (&scope_scpi,   channel,
measure, &resultPDCYcle)) {
printf("\nERROR:PDCYcle_measurement_all_on of scope failed.int ");
//Ausgabe einer Fehlermeldung auf der Konsole
fflush(stdout);   // Leeren des Ausgabepuffers
return -1;

}

// Messung von NDCYcle durchführen

double resultNDCYcle;
if    (rtb2004_meas_NDCYcle_measurement    (&scope_scpi,    channel,
measure, &resultNDCYcle)) {
printf("\nERROR:NDCYcle_measurement_all_on of scope failed.int ");
//Ausgabe einer Fehlermeldung auf der Konsole
fflush(stdout);   // Leeren des Ausgabepuffers
return -1;
}
```

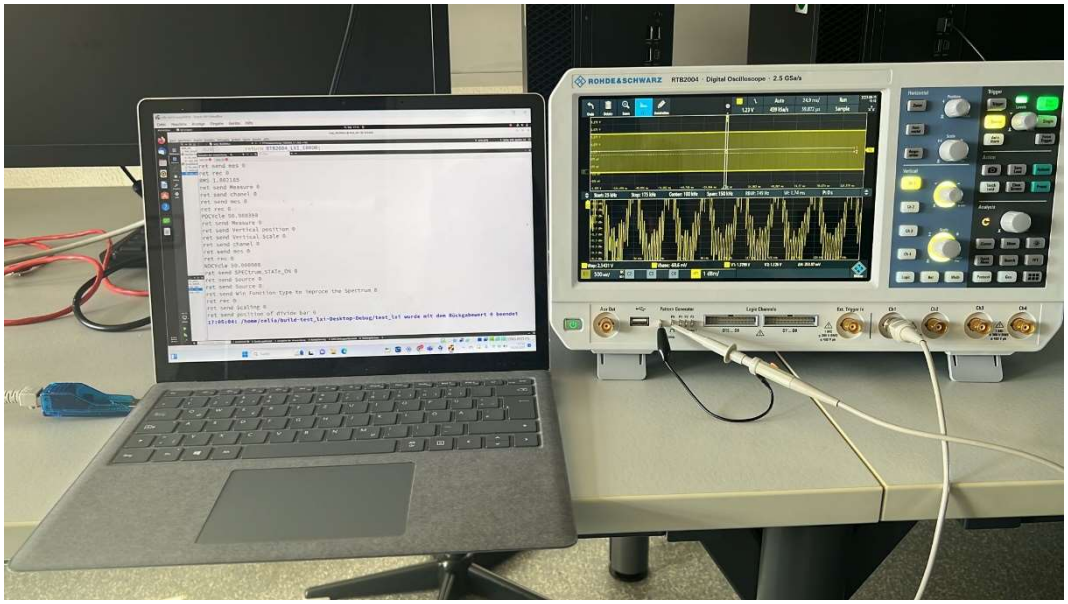
```
if (FFTmeasure(&scope_scpi, channel )) {
printf("\nERROR:FFTmeasure_all_on of scope failed.int ");
//Ausgabe einer Fehlermeldung auf der Konsole
fflush(stdout);
// Leeren des Ausgabepuffers
return -1;}


}
```

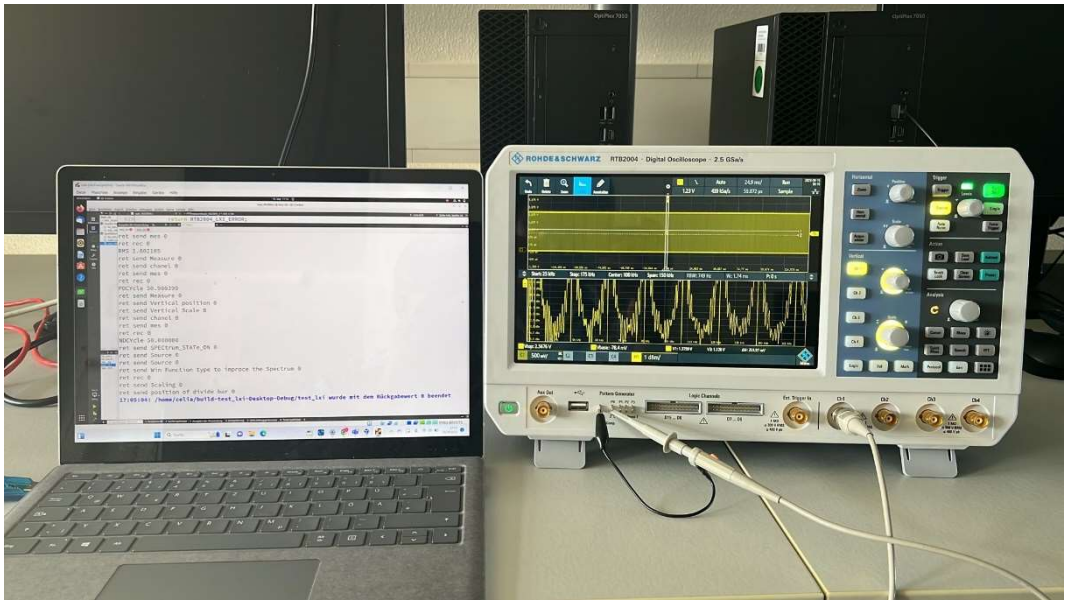
9.2 Screenshots der Anwendung


return RTB2004_LXI_ERROR;
17:05:00: Starting /home/celia/build-test_lxi-Desktop-Debug/test_lxi...
Starting connection
Scope ID: Rohde&Schwarz,RTB2004,1333.1005k04/111686,02.400
ret send Measure 0
ret send chanel 0
ret send mes 0
ret rec 0
FREQuency 9999.852000
ret send Measure 0
ret send chanel 0
ret send mes 0
ret rec 0
MEAN 1.239953
ret send Measure 0
ret send chanel 0
ret send mes 0
ret rec 0
RMS 1.802185
ret send Measure 0
ret send chanel 0
ret send mes 0
ret rec 0
PDCYcle 50.980390
ret send Measure 0
ret send Vertical position 0


return RTB2004_LXI_ERROR;
ret send mes 0
ret rec 0
RMS 1.802185
ret send Measure 0
ret send chanel 0
ret send mes 0
ret rec 0
PDCYcle 50.980390
ret send Measure 0
ret send Vertical position 0
ret send Vertical Scale 0
ret send chanel 0
ret send mes 0
ret rec 0
NDCYcle 50.000000
ret send SPECtrum_STATe_ON 0
ret send Source 0
ret send Source 0
ret send Win Function type to improce the Spectrum 0
ret rec 0
ret send Scaling 0
ret send position of divide bar 0
17:05:04: /home/celia/build-test_lxi-Desktop-Debug/test_lxi wurde mit dem Rückgabewert 0 beendet


RTB2004; 1333.1005K04; 111686 (02.400 2021-11-18)
Undo
Delete
Zoom
FFT
Annotation
Auto
23.59 ms/
Run
1.23 V
463 kSa/s
59.83 µs
Sample
2023-09-15
16:20
TL
Start: 15 kHz
Stop: 185 kHz
Center: 100 kHz
Span: 170 kHz
RBW: 849 Hz
W: 1.53 ms
P: 0 s
C1 Vtop: 2.548 V
C1 Vbase: -53.9 mV
C1 V1: 1.3799 V
V2: 1.126 V
ΔV: 253.97 mV
C1 500 mV/
C2
C3
C4
FFT 9 dBm/
Menu


ROHDE&SCHWARZ
RTB2004 · Digital Oscilloscope · 2.5 GSa/s


ROHDE&SCHWARZ
RTB2004 · Digital Oscilloscope · 2.5 GSa/s

10 Abkürzungsverzeichnis

Vgl.: vergleiche

Abb.: Abbildung

VMM: Virtual Maschine Monitor oder Manager

VM: Virtual Maschine

API: Application Programming Interfaces oder Programmierschnittstellen

Z.B.: zum Beispiel

TIRPC-Bibliothek : port of Suns Transport-Independent RPC Library [16].

11 Abbildungsverzeichnis

12 Tabellenverzeichnis