Fachhochschule Dortmund
University of Applied Sciences and Arts

Cologne Chip

Charakterisierung des Einflusses von Temperatur und Versorgungspannung auf das Verhalten einer SRAM basierten Physically Unclonable Function im GateMate FPGA

Bachelorthesis

im Studiengang „Elektrotechnik Automation & Antriebe“

vorgelegt von

Aya Riahi

Am 18.06.2024

an der Fachhochschule Dortmund

Erstprüfer: Herr Prof. Dr. Karagounis

Zweitprüfer: Herr Felix Schneider

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 henntlich gemacht.

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

Dortmund,18.06.2024 _____

Ort, Datum Unterschrift

Abstract

Die vorliegende Studie behandelt den Einfluss von Temperatur und Versorgungspannung des FPGAs auf die Antwort einer PUF Physically Unclonable Function (PUF) auf Basis von SRAM. Um dieses Ziel zu erreichen, sind mehrere Schritte erforderlich. Zunächst wird ein VHDL-Design zur Ansteuerung dieser Bauteile erstellt und zur Konfiguration des FPGAs verwendet. Weiterhin wird eine Software entwickelt, um das VHDL Design zu nutzen und zu implementieren. Anschließend wird eine virtuelle Maschine mit der Linux-Distribution Ubuntu eingerichtet. Danach wird die Entwicklungsumgebung Qt Creator installiert. Zur Ausführung der gewünschten Aufgabe wird schließlich die Programmiersprache C in Qt Creator verwendet. Die Daten werden über die serielle Schnittstelle mit Hilfe von QSerialPort eingelesen und mit Hilfe von QFile in einer definierten Datei gespeichert. Anschließend werden die Daten ausgewertet, um zu überprüfen, ob die Temperatur und die Versorgungsspannung einen Einfluss auf die PUF haben.

Abstract

This study deals with the influence of temperature and supply voltage of the FPGA on the response of a PUF Physically Unclonable Function (PUF) based on SRAM . To achieve this goal, several steps are necessary. Firstly, a VHDL design is created to control these components and used to configure the FPGA. Furthermore, software is developed to utilise and implement the VHDL design. A virtual machine is then set up with the Linux distribution Ubuntu. The Qt Creator development environment is then installed. Finally, the C programming language is used in Qt Creator to execute the desired task. The data is read in via the serial interface with the help of QSerialPort and saved in a defined file with the help of QFile. The data is then analysed to check whether the temperature and supply voltage have an influence on the PUF.

Inhalt

1.Einleitung 10
2.Aufgabenbeschreibung 11
3.PUFs 11
3.1 Digitale intrinsische physikalisch nicht klonierbare Funktionen 12
3.2 Speicherbasierte intrinsische PUFs 13
3.2.1 SRAM PUFs 13
4.Verwendete Hardware 15
4.1 Pmod USB UART 15
4.2 Pmod BTN für 4 Benutzerdrucktasten 16
4.3 Pmod A UM232H 16
4.4 R&S®HMC8043 18
4.5 Thermostream 18
5. Verwendete Software-Pakete 21
5.1 Ubuntu 21
5.2 VHDL 21
5.2 Qt Creator (GUI) 21
6.Beschreibung des Schaltungsentwurfs 23
6.1 Blockdiagramm 23
6.2 Module 25
6.2.1 BRAM_BA 25
6.2.2 Counter_BA 25
6.2.3 MUX_BA 26
6.2.4 FSM_BA 27
6.2.5 Serial_tx 29
6.2.6 Serial_rx 30
6.2.7 Counter_neu 30
6.2.8 Register 31
6.2.9 PUF_BA 31
6.2.10 PUF_BA_REVERSE 32
6.2.11 PUF_BAT 32
6.3 Versuchsbeschreibung 33
6.3.1Versuchsaufbau 33
7. Erstellung einer grafischen Benutzeroberfläche in Qt mit Definition der Parameter für die Messungen 36
7.1 Aufbau und Funktion der graphischen Benutzeroberfläche 36

7.2 Q-serial-Port 37
7.3 FTDI Device 37
7.4 Code Beschreibung 39
7.5 Ergebnisse 42
7.5.1 Betrachtung auf Zuverlässigkeit 42
7.5.3 Zufälligkeit 44
7.5.4 Betrachtung auf Einzigartigkeit 45
7.5.5 Betrachtung auf Bitfehler-Analyse 45
8.Fazit 46
9.Literaturverzeichnis 47
10.Anhang 50
A.1 VHDL Codes 50
A.1.1 BRAM_BA 50
A.1.2 Counter_BA 57
A.1.3 Counter_neu 58
A.1.4 FSM_BA 60
A.1.5 MUX_BA 65
A.1.6 Register_RX 66
A.1.7 Serial_rx 67
A.1.8 serial_tx 70
A.1.9 PUF_BAT 74
A.1.10 PUF_BA 80
A.1.11 PUF_BA_REVERSE 83
A.1.12PUF_BAT.ccf 85
A.2 Code (Qt Creator) 86
A.2.1 Header-Dateien 86
A.2.2 Quelldateien 96

Abbildungsverzeichnis

Abbildung 1: Logische Schaltung einer SRAM PUF-Zelle [19] 14
Abbildung 2: Elektrische Schaltung einer SRAM (PUF)-Zelle in Standard-CMOS-Technologie. [20] 14
Abbildung 3: Pmod USBUART [21] 15
Abbildung 4: Pmod USBUART [22] 15
Abbildung 5: Pmod_Button. [23] 16
Abbildung 6: UM232H USB to Serial/FIFO Development Module. [24] 17
Abbildung 7: UM232H USB to Serial/FIFO Development Module. [25] 17
Abbildung 8: R&S® HMC804x layout. [26] 18
Abbildung 9: Thermostream Display. [27] 19
Abbildung 10: Darstellung des Thermostream Geräts. [28] 20
Abbildung 11: Blockdiagramm der Schaltung für die Beschreibung und Auslesung des BRAMs 24
Abbildung 12: BRAM_BA Modul 25
Abbildung 13: Counter_BA Modul 26
Abbildung 14: MUX_BA Modul 26
Abbildung 15: FSM_BA Modul 27
Abbildung 16: Zustandsdiagramm 28
Abbildung 17: serial_tx Modul 29
Abbildung 18: Serial_rx Modul 30
Abbildung 19: Counter_neu Modul 30
Abbildung 20: Register Modul 31
Abbildung 21: PUF_BA Modul 31
Abbildung 22: PUF_BA_REVERSE Modul 32
Abbildung 23: PUF_BAT Modul 32
Abbildung 24: Aufbau der gesamten Versuch 33
Abbildung 25: Beschaltung des UM232H FTDI Baustein auf einem Steckboard 34
Abbildung 26: Grafische Benutzeroberfläche für die Durchführung des Versuchs 35
Abbildung 27: Konfiguration des FPGA's 35
Abbildung 28: Design der Benutzeroberfläche 36
Abbildung 29: UM232H Impulse 38
Abbildung 30: Oszilloskop Screenshot, welcher den Übergang eines FTDI Ausgangssignals von Low nach High zeigt. 38
Abbildung 31: Intra-Chip Hamming-Distanz der untersuchten SRAM PUF als Funktion der Temperatur für verschiedene Versorgungsspannungen 43
Abbildung 32: Einzigbarkeit vs Temperatur-SRAM PUFs 44
Abbildung 33: Bitfehleranalyse vs Temperatur SRAM PUFs 46

Tabellenverzeichnis

Tabelle 1: Einstellbarer Temperaturbereich je nach Gerätemodell [28] 20

Verzeichnis der Quellcodes

Quellcode 1: Verwendung der QSerialPort Bibliothek 37
Quellcode 2: Header Dateien in der worker.cpp 37
Quellcode 3: Q-Serial-Port Instanz 37
Quellcode 4: setParameters Methode 39
Quellcode 5: readSerialData Methode 40
Quellcode 6: reprogramAndSetVoltage 41
Quellcode 7: setMyFileName Methode 41
Quellcode 8: Stop Methode 41

Abkürzungsverzeichnis

FH: Fachhochschule Dortmund

PUF: Physically Unclonable Function

FPGA: Field Programmable Gate Array

GPIO: General

Purpose Input/Output

BRAM: Block Random Access Memory

Pmod: Peripheral Modul

VHDL: Very High Hardware Description Language

SPI: Serial Peripheral Interface

USB: Universal Serial Bus

UART: Universal Asynchronous Receiver Transmitter

DC: direct current

AC: alternating current

PCB: Power Supply

SLP: Super Low Power

SRAM: Static random access memory

CPE: combining programmable element

FTDI: Future Technology devices international

R&S: Rohde und Schwarz

HMC: Hameng Instrument

GUI: Graphical User Interface

CPP: C++

QIO Device: Qt Input/Output

QFile: QtFile

1.Einleitung

Diese Arbeit konzentriert sich auf den Einfluss der Temperatur und der Versorgungspannung auf die Antwort einer SRAM-basierten physikalisch nicht klonbaren Funktion (engl. Physical Unclonable Function, eng.), die in einem GateMate A1 FPGA implementiert wurde. SRAM ist ein statischer Direktzugriffsspeicher, der Datenbits im Speicher behält, solange die Versorgungsspannung anliegt. SRAM ist meist in „System on Chips“ (SoC) integriert und kann deswegen schneller zugegriffen werden als dynamische Direktzugriffsspeicher, die meist als externe Bauelemente in eingebetteten Systemen verwendet werden. GateMate FPGA ist eine Familie kleiner bis mittelgroßer FPGAs, welche von der Cologne Chip AG mit Sitz in Köln produziert werden und alle typischen Anwendungsanforderungen digitaler Systeme erfüllen.

Programmierbares Silizium kann in Hochgeschwindigkeitsanwendungen mit geringem Stromverbrauch verwendet werden und ermöglicht den Einsatz in vielen Bereichen wie z.B Industrie, Automatisierung, Kommunikation, Sicherheit, Automobil, Internet der Dinge(IoT) und künstliche Intelligenz (KI). Das GateMate FPGA-Programm wird vom Bundesministerium für Wirtschaft und Energie im Rahmen der Important Project of Common European Interest (IPCEI) Initiative der Europäischen Kommission gefördert. In dieser Arbeit wird zunächst der Begriff „Physical Unclonable Function“ geklärt und das Funktionsprinzip der SRAM-basierten Physical Unclonable Function vorgestellt. Anschließend wird die verwendete Hardware inklusive FPGA und benötigter Zusatzmodule beschrieben. Beim Entwurf digitaler Schaltkreise spielt auch EDA-Software (Elektronic Design Automation) eine wichtige Rolle. Das Besondere an diesem Projekt ist die Verwendung der Open-Source-Synthesesoftware Yosys, die zusammen mit einer proprietären Place & Route-Software zur Implementierung digitaler Schaltkreise auf dem GateMate FPGA verwendet wird. Anschließend wird der Schaltungsentwurf in VHDL vorgestellt, der für die Auslesung von Daten aus dem BRAM des FPGA und die Übertragung dieser Daten über die UART-Schnittstelle verantwortlich ist. Abschließend wurde eine GUI in Qt erstellt, mit der die Parameter für Messungen festgelegt werden können. Ein FTDI Baustein wird verwendet, um die Daten, welche über eine UART Schnittstelle aus dem FPGA ausgelesen werden über USB an einen PC zu übertragen. Ein weiterer FTDI Baustein wird verwendet, um aus der auf dem PC laufenden Software die Auslesung der PUF durch das Setzen von Signalen zu initiieren. Außerdem ist in Qt die Einbindung der Klasse Qserial erforderlich, um die Daten über die serielle Schnittstelle auszulesen und später mit Hilfe der Klasse QFile in die festgelegte Datei gespeichert werden.

2.Aufgabenbeschreibung

Ziel des Projektes ist es, den Einfluss von Temperatur und Versorgungsspannung auf das Verhalten eines SRAMs in einem FPGA zu charakterisieren. Dazu soll ein Qt-Programm geschrieben werden, das ein Klimagerät so ansteuert, dass die Temperatur des Bauteils innerhalb eines vorgegebenen Temperaturbereichs schrittweise verändert. Für jede Temperatur soll die Versorgungsspannung innerhalb eines vorgegebenen Spannungsbereichs schrittweise angelegt werden. Für jedes Parameterpaar aus Temperatur und Versorgungsspannung wird die SRAM-Antwort ausgelesen und in einer Datei gespeichert. Anschließend wird die Auslesung des PUF durch das entsprechende logische Signal mit Hilfe des FTDI-Bausteins initiiert. Die Daten werden über QSerialPort über die serielle Schnittstelle eingelesen und über QFile in die festgelegte Datei gespeichert. Die Messdaten werden ausgewertet und geprüft, ob die Temperatur und die Versorgungsspannung einen Einfluss auf den Oszillator-PUF Antwort haben.

3.PUFs

Der Begriff Physical Unclonable Function (PUF) lässt sich nur schwer in einer einzigen geschlossenen Definition erfassen. Eine PUF führt eine funktionelle Operation durch. Diese funktionelle Operation erzeugt eine messbare Ausgabe, wenn sie mit einer bestimmten Eingabe angeregt wird. In den meisten Fällen ist eine PUF keine echte Funktion im mathematischen Sinne, da eine Eingabe in eine PUF mehr als eine mögliche Ausgabe haben kann. Es ist angemessener, eine PUF als eine Funktion im technischen Sinne zu betrachten, d. h. als eine Prozedur, die von einem bestimmten physikalischen System ausgeführt wird oder auf dieses System einwirkt. Im Zusammenhang mit PUFs wird die Eingabe als Challenge und die Ausgabe als Response bezeichnet. Eine angewandte Eingabe und die gemessene Antwort wird im Allgemeinen als Challenge-Response-Paar (CRP) und die von einer bestimmten PUF erzwungene Beziehung zwischen Eingabe und Ausgabe als ihr CRP-Verhalten bezeichnet [1]. Bei der Charakterisierung von PUFs werden die folgenden beiden Metriken verwendet.

- ✓ Für eine bestimmte Eingabe ist die Inter-Distanz zwischen zwei verschiedenen PUF-Instanziierungen der Abstand zwischen den beiden Antworten, die sich aus der einmaligen Anwendung dieser Eingabe auf beide PUFs ergeben.
- ✓ Für eine bestimmte Eingabe ist die Intra-Distanz zwischen zwei Bewertungen auf einer einzigen PUF-Instanziierung die Distanz zwischen den beiden Antworten, die sich aus der zweimaligen Anwendung dieser Eingabe auf eine PUF ergeben.

Die erste Metrik beschreibt die Einzigartigkeit des generierten Bitmusters. Die Einzigartigkeit des Bitmusters setzt voraus, dass nur zufällige Fertigungsschwankungen Schlüsselmerkmale bestimmen, während der Einfluss systematischer Ursachen ausgeschlossen werden kann. Zur quantitativen Bewertung der Einzigartigkeit der generierten Schlüssel wird die Inter-Hamming Distanz herangezogen, die wie folgt definiert ist:

$$D_{Pi,Pj}^{inter} = \frac{2}{k(k-1)} \sum_{j\equiv 1}^{k-1} \sum_{j=i+1}^{k} \frac{HD(Pi,Pj)}{n} \,.100\%$$

Die Formelgröße n ist dabei die Bitlänge des Schlüssels und k die Anzahl der Primitive, welche miteinander verglichen werden. Die Funktion HD bezeichnet die Hamming-Distanz zwischen den Schlüsseln, welche von den PUV Primitiven Pi bzw. Pj generiert werden und der Anzahl der Bitstellen entspricht, an denen sich zwei Codewörter voneinander unterscheiden. Im Idealfall ist die Inter-Hamming Distanz binomial verteilt und besitzt einen Erwartungswert von E(DinterPi;Pj) = 50%. Die zweite Metrik beschreibt die Reproduzierbarkeit des Bitmusters, die besonders wichtig ist, weil bei der Verwendung von PUF-Zellen das Bitmuster bei Bedarf immer wieder neu generiert wird. Zur Bewertung der Reproduzierbarkeit der generierten Schlüssel wird die Intra-Hamming Distanz herangezogen, die wie folgt definiert ist:

$$D_{P_i,P_i'}^{\text{intra}} = \frac{1}{k} \sum_{j=i+1}^{k} \frac{HD(P_i,Pi')}{n} \cdot 100\%$$

Die Formelgröße n ist dabei wiederum die Bitlänge des Schlüssels und k die Anzahl der Vergleiche, welche zwischen dem Schlüssel Pi und dem zu einem späteren Zeitpunkt aber von der gleichen Zelle generierten Schlüssel P'i durchgeführt wurde. Im Idealfall besitzt die Intra-Hamming Distanz den Wert Dintra Pi;P'i= 0, was bedeutet, dass einmal generierte Schlüssel jederzeit reproduziert, werden können.

3.1 Digitale intrinsische physikalisch nicht klonierbare Funktionen

PUFs gelten als Alternative zu On-Chip Speichern für Schlüssel, die in der Kryptographie eingesetzt werden. Kryptographischen Verfahren wie z.B. RSA, SHA und AES werden verwendet, um die Firmware von eingebetteten Systemen vor unautorisiertem Zugriff und Manipulation zu schützen. Die Sicherheit des Systems wird erhöht, wenn wie im ISO/IEC 11889 Standard spezifiziert, der Schlüssel in einem nicht-flüchtigen Speicher On-Chip abgelegt ist und keine I/O Schnittstellen genutzt werden, um den Schlüssel aus externen Speicherbausteinen einzulesen. Bei der Nutzung von PUF-Primitiven wird der Schlüssel nicht gespeichert, sondern das Bitmuster immer wieder bei Bedarf neu generiert. Dadurch können teure Produktionsschritte für integrierte ROM, EEPROM oder Flash-Module eingespart werden. PUF Primitive nutzen dabei stochastische Prozessvariationen, die sich inhärent bei der Produktion von Halbleiterbauelementen ergeben. Obwohl moderne CMOS-Technologien einer strengen Prozesskontrolle unterliegen, können Hersteller zufällige Variationen auf der mikroskopischen Ebene weder verhindern noch beeinflussen, was beispielsweise zu lokalen Änderungen der Oxiddicken und Dotierungskonzentrationen der Bauelemente führt. Für ein hohes Maß an Sicherheit ist die Verwendung von intrinsischen PUFs anzustreben. Hierbei werden zwei Voraussetzungen unterschieden, damit ein PUF als intrinsisch bezeichnet werden kann. Die PUF muss einschließlich aller notwendigen Messeinrichtungen vollständig in der Einbettungsvorrichtung z.B. einem Mikrochips integriert sein. Zweitens sollte die vollständige PUF-Konstruktion aus Primitiven bestehen, die für den Herstellungsprozess des Einbettungsgeräts natürlich verfügbar sind. Die erste Bedingung bedeutet, dass das Gerät seine eigene PUF abfragen und auslesen kann, ohne dass externe Instrumente erforderlich sind und ohne dass die Anfrage und die Antwort das Gerät verlassen müssen. Die zweite Bedingung besagt, dass die komplette PUF- Konstruktion außer dem vom PUF belegten Platz praktisch

keine zusätzlichen Kosten verursacht, d. h keine zusätzlichen Fertigungsschritte oder speziellen Komponenten erforderlich sind. Diese Bedingung wird beispielsweise von Beschichtungs-PUF und integrierten optischen PUFs nicht erfüllt, da beide hochspezialisierte Verarbeitungsschritte erfordern. [2]

3.2 Speicherbasierte intrinsische PUFs

Im Rahmen dieser Thesis wird eine Art von intrinsischer PUFs erörtert, die auf dem Einschwingzustand digitaler Speicherprimitive basiert. Eine digitale Speicherzelle ist typischerweise eine digitale Schaltung mit mehr als einem logischen stabilen Zustand. Wenn sie sich in einem ihrer stabilen Zustände befindet, kann sie Informationen speichern, z. B. eine Binärziffer im Fall von zwei möglichen stabilen Zuständen. Wenn das Element jedoch in einen instabilen Zustand gebracht wird, ist nicht klar, was passiert. Es könnte anfangen, zwischen instabilen Zuständen zu oszillieren, oder es könnte wieder in einen seiner stabilen Zustände konvergieren. Im letzteren Fall ist zu beobachten, dass bestimmte Zellen bestimmte stabile Zustände gegenüber anderen stark bevorzugen. Dieser Effekt kann oft nicht durch die logische Implementierung der Zelle erklärt werden. Stattdessen liegt eine interne physikalische Fehlanpassung vor, die z. B. durch Fertigungsschwankungen verursacht wird. Aus diesem Grund ist der stabile Einschwingzustand einer destabilisierten Speicherzelle ein guter Kandidat für eine PUF-Antwort.[3]

3.2.1 SRAM PUFs

SRAM-PUFs wurden in [5] vorgeschlagen und ein sehr ähnliches Konzept wurde in [6] zeitgleich vorgestellt. SRAM oder statischer Direktzugriffsspeicher ist eine Art digitaler, aus Zellen bestehender Speicher, wobei jede Zelle jeweils eine Binärziffer speichern kann. Eine SRAM-Zelle, wie in Abb. 1 gezeigt, ist aus zwei kreuzgekoppelten Invertern aufgebaut, was zu zwei stabilen Zuständen führt. In normaler CMOS-Technologie wird diese Schaltung mit 4 MOSFETs implementiert, wobei zwei zusätzliche MOSFETs, wie in Abbildung 2 gezeigt, für den Lese- und Schreibzugriff verwendet werden. Aus diesen Gründen wird die physikalische Diskrepanz zwischen den beiden symmetrischen Hälften der Schaltung, die jeweils einen Inverter implementieren, so gering wie möglich gehalten. Aus der logischen Beschreibung der Zellen geht nicht hervor, in welchem Zustand sich der Speicher direkt nach dem Einschalten befindet, d. h. was passiert, wenn die Versorgungsspannung angelegt wird. Es ist zu beobachten, dass einige Zellen beim Einschalten vorzugsweise eine Null speichern, andere vorzugsweise eine Eins und einige Zellen wiederum keine wirkliche Präferenz aufweisen. Die Verteilung dieser drei Arten von Zellen über den gesamten Speicher ist zufällig. Wie sich herausstellt, bestimmt die zufällige physikalische Fehlanpassung in der Zelle, welche durch Fertigungsschwankungen verursacht wird, das Einschaltverhalten. Sie zwingt eine Zelle je nach Vorzeichen der Fehlanpassung während des Einschaltens auf Null oder Eins. Ist die Fehlanpassung sehr gering, wird der Einschaltzustand durch stochastisches Rauschen in der Schaltung bestimmt und ist zufällig, ohne dass eine echte Präferenz vorliegt.

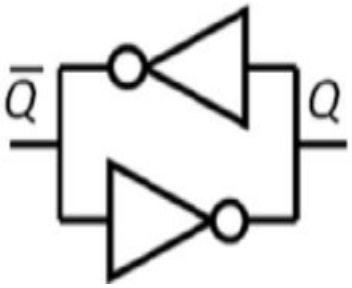

Abbildung 1: Logische Schaltung einer SRAM PUF-Zelle [19]

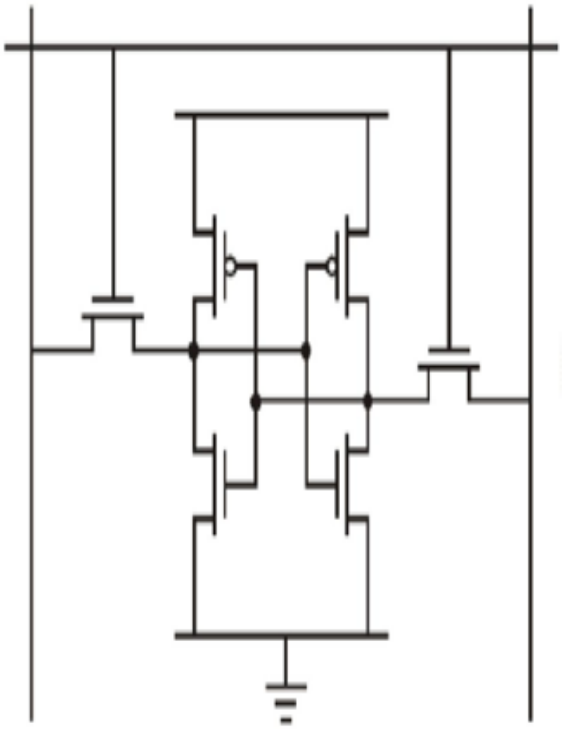

Abbildung 2: Elektrische Schaltung einer SRAM (PUF)-Zelle in Standard-CMOS-Technologie. [20]

In [5] wurden umfangreiche Experimente mit SRAM-PUFs durchgeführt. Es wurde der Einschaltzustand von 8190 Byte SRAM aus verschiedenen Speicherblöcken auf verschiedenen FPGAs erfasst. Die Ergebnisse zeigen eine durchschnittliche Inter-Distanz zwischen zwei verschiedenen Blöcken von Dinter = 49.97% und die durchschnittliche Intra-Distanz innerhalb mehrerer Messungen eines einzelnen Blocks beträgt Dintra = 3,57% für eine feste Umgebung und Dintra < 12% für große Temperaturabweichungen. In [7] schätzen die Autoren den Entropiegehalt der SRAM-Einschaltzustände auf 0,76 Bit pro SRAM-Zelle. In [6, 8] wurde das Einschaltverhalten von SRAM auf zwei verschiedenen Plattformen untersucht. Für 5120 Blöcke von 64 SRAM-Zellen, die auf 8 kommerziellen SRAM-Chips gemessen wurden, erhielten sie Dinter = 43.16% und Dintra =3.8%, und für 15 Blöcke von 64 SRAM-Zellen aus dem eingebetteten Speicher in 3 Mikrocontroller-Chips erhielten sie Dinter = 49.34% und Dintra = 6.5%.

4.Verwendete Hardware

4.1 Pmod USB UART

In diesem Projekt sollen Daten aus dem FPGA ausgelesen und über eine USB-Schnittstelle an den PC gesendet werden. Für die Übertragung der Daten wird das USBUART PMOD-Modul verwendet, welches eine USB-zu-UART-Kreuzkonvertierung durch den FTDI FT232R Baustein bietet (siehe Abbildung3 und 4). Benutzer können Daten mit dem Pmod senden und empfangen, nachdem sie in das entsprechende Format konvertiert wurden.[9]

Abbildung 3: Pmod USBUART [21]

Abbildung 4: Pmod USBUART [22]

Eigenschaften:

- USB auf serielle UART-Schnittstelle
- Micro-USB-Anschluss
- Möglichkeit, die Systemplatine über den FTDI-Chip zu versorgen
- Kleine Leiterplatte für flexible Ausführungen 1.0“ × 0.8” (2.5 cm × 2.0 cm)
- 6-poliger Pmod-Stecker mit UART-Schnittstelle [9]

4.2 Pmod BTN für 4 Benutzerdrucktasten

Auf einer Systemplatine stellt der Pmod BTN dem Benutzer vier Drucktasten für einfache Benutzereingaben zur Verfügung. In diesem Projekt wird der Pmod BTN für das Signal RST benötigt. Die Drucktaste BTN1 wird als Pin_in mit der GPIO Bank IO_NB_A0 des FPGAs GatemateTM verbunden. [10]

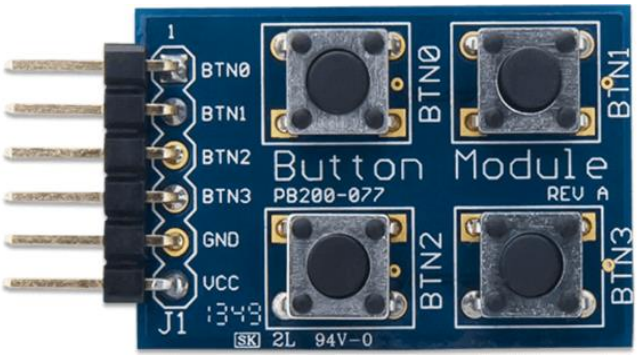

Abbildung 5: Pmod_Button. [23]

Eigenschaften:

- Vier Drucktasten mit Momentanwert
- Entprellende Filter
- Invertierende Schmitt-Trigger
- Kleine PCB-Größe für flexible Designs 3,0 cm × 2,0 cm (1,2" × 0,8")
- 6-poliger Pmod-Anschluss mit GPIO-Schnittstelle
- Entspricht der Digilent Pmod-Schnittstellen-Spezifikation Typ1[10]

4.3 Pmod A UM232H

Das UM232H ist ein USB-seriell/FIFO-Entwicklungsmodul aus der FTDI-Produktpalette, das den FT232H USB Hi-Speed (480Mb/s) Single-Port Bridge Chip für die USB-Signalisierung und Protokolle verwendet. Das UM232H ist ideal für Entwicklungszwecke, um schnell die Funktionalität des USB-Anschlusses in einem Zielsystem zu testen. Das UM232H ist ein Modul, das in einen standardmäßigen 0,6" breiten 28-poligen DIP-Sockel eingesteckt werden kann. Die USB-Verbindung an ein Host-System erfolgt über einen Mini-B-USB-Stecker. Alle verwendeten Komponenten, einschließlich des FT232H, sind Pb-frei (RoHS-konform). [11]

Der FT232H ist ein vielseitiger USB 2.0 Hi-Speed (480Mb/s) zu Seriell/FIFO IC (Integrierter Schaltkreis), der für verschiedene Kommunikationsprotokolle entwickelt wurde. Er kann als Konverter für UART, FIFO und FTDI´s MPSSE-Modus (inklusive JTAG, SPI, und I2C) konfiguriert werden und bietet 14 Bit-Banging-Leitungen. Darüber hinaus umfasst das Leistungsspektrum weitere Modi, darunter den CPU-Style FIFO-Modus für parallele Datenverbindung, einen schnellen seriellen Schnittstellenmodus sowie halbduplexe FT1248-Schnittstelle, die eine flexible und Leistungsstrake Datenkommunikation gewährleisten. Aufgrund seiner Vielseitigkeit eignet sich der FT232H somit ideal für unterschiedlichste Kommunikationsanforderungen und Anwendungsbereiche. Die Signalbezeichnungen und Pinbezeichnungen für die einzelnen Pins des UM232H sind in Abbildung 7 dargestellt. [11]

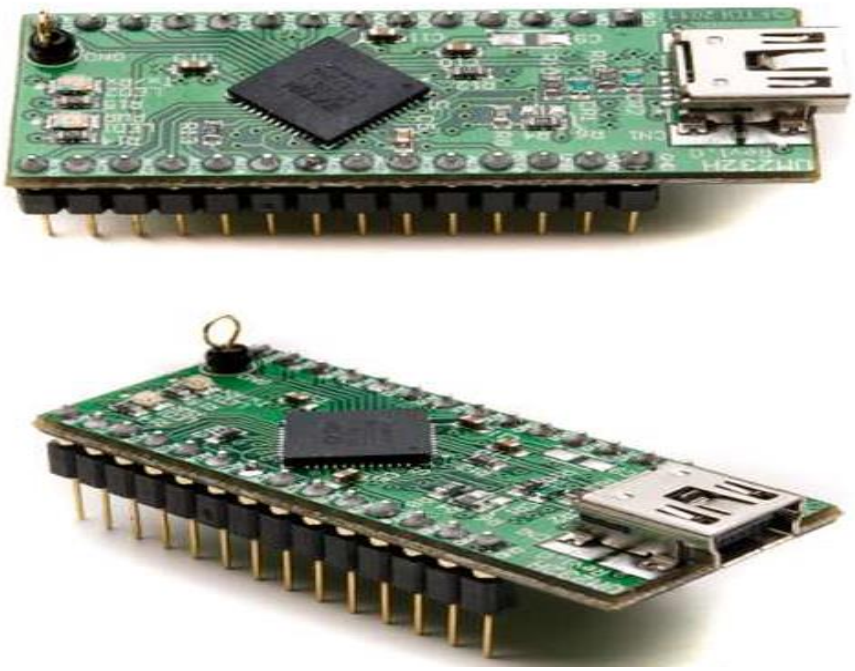

Abbildung 6: UM232H USB to Serial/FIFO Development Module. [24]

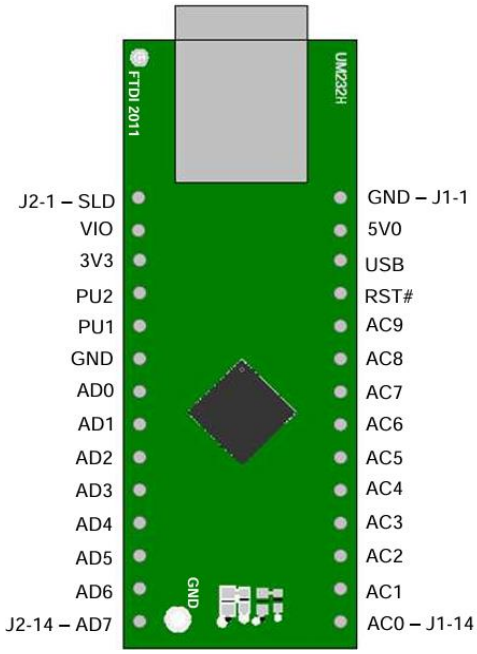

Abbildung 7: UM232H USB to Serial/FIFO Development Module. [25]

4.4 R&S®HMC8043

Das Netzteil R&S®HMC804x eignet sich aufgrund seiner Ausstattung und seines großen Funktionsumfangs ideal für den Einsatz in Forschungslaboren und industriellen Umgebungen. Dank seiner hohen Energieeffizienz bleibt das Netzteil auch bei maximaler Leistung kühl und leise. Praktische Schnittstellen und Anschlüsse ermöglichen ein schnelles und komfortables Arbeiten. [12]

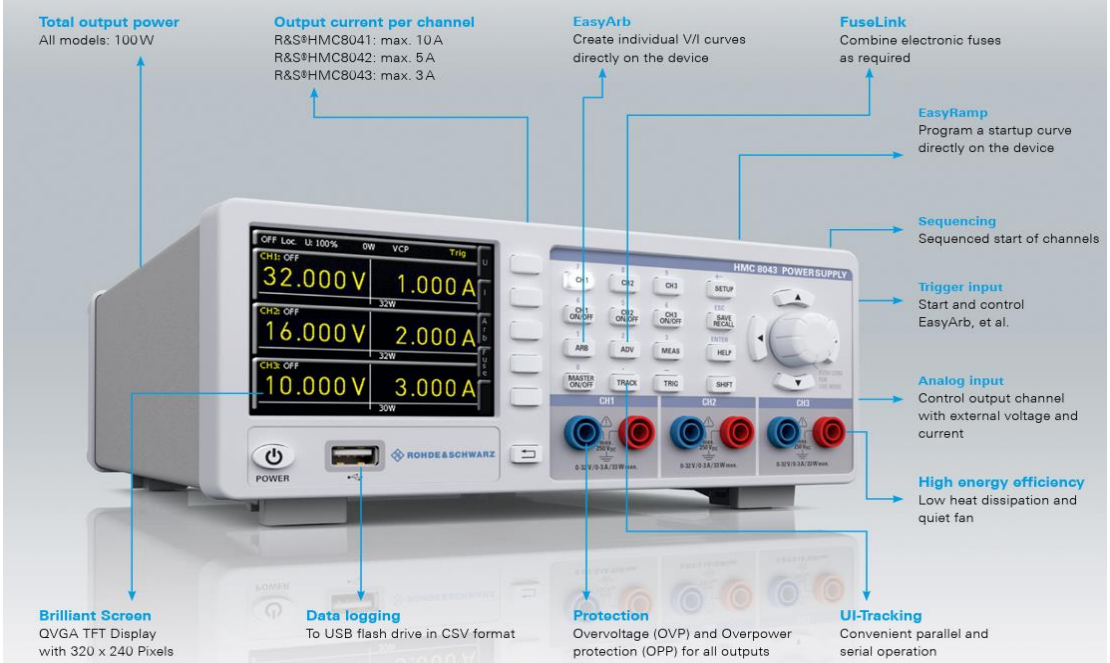

Abbildung 8: R&S®HMC804x layout. [26]

Die R&S®HMC804x-Serie umfasst drei Modelle mit einer maximalen Gesamtleistung von bis zu 100 W und einem Gleichspannungsbereich von 0 V bis 32 V. Der einkanalige R&S®HMC8041 stellt maximal 10 A, der zweikanalige R&S®HMC8042 maximal maximal 5 A und dreikanalig R&S® HMC8043 maximal 3A pro Kanal zu Verfügung, mit Zwei- und Dreikanalmodellen können Benutzer mehrere Ausgänge parallel oder in Reihe schalten, die Spannung oder den Strom zu erhöhen. Die Ausgänge sind galvanisch getrennt, erdfrei und gegen Überlast und Kurzschluss geschützt. Spannungs-, Strom- und Leistungswerte werden auf einem QVGA-Display angezeigt. [12]

Hauptmerkmale:

- Übersichtliche Darstellung aller gemessenen Parameter
- Galvanisch getrennte, erdfreie und kurzschlussfeste Ausgänge
- Schutzfunktionen können für jeden Kanal separat eingestellt werden.
- Ideale Stromversorgungen für Hardwareentwickler und Laborplätze [13]

4.5 Thermostream

Thermostreams sind portable Temperaturkontrollsysteme für Hochgeschwingkeitstests, Konditionierung, Charakterisierung und Verifizierung von elektronischen Komponenten, Sensoren, ICs und PCBs. Die Temperatur wird über den Luftstrom des Thermostream auf die zu testende

Komponente geführt und kann einen Bereich von -100 bis +300°C abdecken. Thermostreams sind ideal für Labor- oder Produktionsanwendungen mit hohem Durchsatz, bei denen Testobjekte mit geringer Masse schnell und präzise temperiert werden müssen. Ein lokale Touchscreen bietet intuitive Bedienelemente. Darüber hinaus existieren Fernbedienungsmöglichkeiten für die Einrichtung und Ausführung komplexer Temperaturwechselprogramme. [14]

Abbildung 9: Thermostream Display. [27]

- Hoher Durchsatz

- Tiefsttemperatur bis zu -100°C

- Höchsttemperatur bis zu +300°C

- Schnelle Temperaturwechsel bis zu 18°C/Sek.

- Gehäuse für Komponenten und PCBs [14]

- Bequeme Bedienung

- Fernsteuerung

- Es ist kein Stickstoff (LN2) erforderlich, um extrem niedrige Temperatur zu erreichen

- Erweiterte Reichweite zur Anpassung an Testköpfen

- Stromsparfunktion bei Nichtbenutzung [14]

ATS-710-M

Abbildung 10: Darstellung des Thermostream Geräts. [28]

MODEL	TEMPERATURE RANGE
ATS - 710	-80°C to +225°C
ATS - 710 WM	-80°C to +225°C water cooled
ATS - 730	-90°C to +225°C high capacity
ATS - 750	-90°C to +300°C high temp, high capacity
ATS - 770	-100°C to +225°C ultra-low temp

Tabelle 1: Einstellbarer Temperaturbereich je nach Gerätemodell [28]

5. Verwendete Software-Pakete

5.1 Ubuntu

Ubuntu ist eine GNU/Linux-Distribution, die auf Debian basiert. Der Name Ubuntu bedeutet auf Zulu „Menschlichkeit“. Ziel der Systementwickler war es, ein einfach zu installierendes und zu verwendendes Betriebssystem mit passender Software zu schaffen. Das Projekt wird vom Softwareentwickler Canonical finanziert, der vom südafrikanischen Geschäftsmann Mark Shuttlerworth gegründet wurde. Ubuntu erfreut sich seit der Veröffentlichung seiner ersten Version im Oktober 2004 stetig wachsender Beliebtheit und ist heute eine der am weitesten verbreiteten GNU/Linux-Distribution. Die Zahl der Nutzer wird auf 25 Millionen geschätzt. Von Version 11.04 bis 17.04 verwendete Ubuntu standarmäßig die von der Ubuntu-Entwicklergemeischft entwickelte graphische Umgebung Unity, ab Version 17.10 basiert die graphische Benutzeroberfläche von Ubuntu jedoch wieder auf Gnome. Es gibt viele verschiedene Versionen von Ubuntu. Zu den offiziellen Teilprojekten gehören Kubuntu mit KDE, Xubuntu mit Xfce, Ubuntu Mate mit Mate, Ubuntu Budgie und Ubuntu Studio, die speziell für Audio-, Grafik und Videobearbeitungsanforderungen entwickelt wurden. Neue Versionen von Ubuntu erscheinen alles sechs Monate im April (Version 04) und Oktober (Version 10). Die aktuelle Version 24.04 LTS „Noble Numbat“ wurde am 25.April 2024 veröffentlicht. [15]

5.2 VHDL

Very High Speed Integrated Circuit Hardware Description Language (auch VHSIC Hardware Description Language), kurz VHDL, ist eine Hardwarebeschreibungssprache, mit der es möglich ist, digitale Systeme textbasiert zu beschreiben. VHDL ist seit 1987 als IEEE-Standard festgelegt und es gibt inzwischen einige ebenfalls standardisierte Spracherweiterungen. Darüber hinaus gibt es Sprachderivate wie zum Beispiel VHDL-AMS, mit deren Hilfe auch analoge oder Mixed-Signal-Systeme beschrieben werden können. VHDL ist als Beschreibungssprache keine Programmiersprache; da sie jedoch Objekte beschreibt, deren Aufgabe meist die Informationsverarbeitung ist, kann über deren Simulation dennoch Datenverarbeitung stattfinden, indem für diesen Simulationslauf mitgegebene „Eingangsdaten“ von der (simulierten) Hardware zu „Ergebnisdaten“ verarbeitet werden. Durch diesen Umweg kann VHDL (in Kombination mit einem Simulator) wie eine Programmiersprache Turing-vollständige Datenverarbeitung beschreiben. Durch fortschrittliche Schaltungsgeneratoren ist es mitunter sogar möglich, anstatt des Hardwareaufbaus für einen Algorithmus nur den Algorithmus selbst anzugeben; die dazugehörige Schaltung wird vollautomatisch erzeugt. Dies nähert VHDL einer Programmiersprache weiter an. [16]

5.2 Qt Creator (GUI)

Qt Creator ist eine plattformübergreifende integrierte Entwicklungsumgebung (IDE), die das Erstellen von Anwendungen mithilfe der Qt-Plattform einfacher und effizienter machen soll. Qt ist eine leistungsstarke C-Plattform für die Entwicklung von Anwendungen mit grafischer Benutzeroberfläche (GUI) sowie Nicht-GUI-Anwendungen. Qt Creator bietet eine Reihe von Funktionen zur Unterstützung der Qt-Programmierung, darunter:

Qt-Creator verfügt über einen Leistungsstarken Code-Editor mit Syntaxhervorhebung, Code-Vervollständigung und Code-Navigation, der das Codieren erleichtert. Dazu enthält Creator einen

Drag-and-Drop-GUI-Designer, mit dem Entwickler Benutzeroberflächen visuell gestalten können. Das Toll generiert Qt-spezifischen Code und erleichtert so die Erstellung komplexer Benutzeroberflächenlayouts.

Außerdem bietet Qt-Creator Projektmanagementfunktionen, einschließlich Unterstützung für die Verwaltung mehrerer Projekte, Projektvorlagen und Integration mit Versionskontrollsystemen wie Git. Der Debugging-Prozess unterstützt Funktionen wie Stopps, Überprüfung von Variablen und freischaltcodes. Zusätzlich zur herkömmlichen Widget-basierten GUI unterstützt Qt Creator auch das Design von Benutzeroberflächen mit Qt Quick Designer, einer deklarativen Designsprache für Benutzeroberflächen. Mit Qt-Quick Designer können Entwickler mithilfe von QML (Qt Modeling language) flexible und optisch ansprechende Benutzeroberflächen erstellen. Außerdem lässt sich Qt-Creator das Toll in Qt Assistant integrieren, einen Qt-Dokumentbetrachter. Dies ermöglicht Entwicklern einen einfachen Zugriff auf die Qt-Dokumentation innerhalb der IDE.

Insgesamt ist Qt-Creator eine leistungsstarke Entwicklungsumgebung zur Erstellung von Qt-Anwendungen, die eine Reihe von Funktionen bietet, um den Entwicklungsprozess zu rationalisieren und die Erstellung hochwertiger grafischer Benutzeroberflächen zu erleichtern.

6.Beschreibung des Schaltungsentwurfs

Dieses Kapitel beschreibt den Schaltungsentwurf mit dessen Hilfe, ein instanziierter BRAMs, beschrieben und ausgelesen wird, während die Daten über eine UART-Schnittstelle versendet werden. Anfangs wird die Entwurfsstruktur anhand eines Blockdiagramms erläutert. Dann folgt eine detaillierte Beschreibung der einzelnen in VHDL entworfenen Module. Der vollständige VHDL-Quellcode ist im Anhang enthalten.

6.1 Blockdiagramm

Die Struktur des Schaltungsentwurfs ist im Blockdiagramm in Abbildung 11 dargestellt. Als zentrales Element befindet sich in der Mitte des Blockschaltbildes eine Instanz des BRAMs.
Die Adresse, welche bestimmt, welches Datenwort in das BRAM geschrieben oder aus dem BRAM ausgelesen werden soll, wird durch einen Counter generiert. Das BRAM besitzt eine Datenwortbreite von 18 Bits, während Daten byteweise über die UART-Schnittstelle übertragen werden. Aus diesem Grund befindet sich zwischen dem BRAM-Eingang und dem Ausgang des UART-Moduls ein Register, welches 3 Bytes vom UART_RX entgegennimmt und zu einem 18 Bit Datenwort zusammensetzt. An welcher Stelle des Registers ein über die UART-Schnittstelle empfangenes Byte platziert wird, entscheidet eine Zustandsmaschine, die am unteren Teil des Blockschaltbildes zu sehen ist. Sobald ein neues Byte am Ausgang des UART_RX-Blocks anliegt, initiiert die Zustandsmaschine die Abspeicherung des Bytes, wobei der Zählerstand CR die Position des Bytes im Register bestimmt. Sobald das Datenwort vollständig empfangen worden ist, löst die Zustandsmaschine die Inkrementierung der BRAM-Adresse aus und das Datenwort wird in den BRAM hineingeschrieben. Nachdem der Prozess der Datenübertragung in das BRAM abgeschlossen worden ist, kann der Auslesesprozess des BRAMS begonnen werden. Ein aus dem BRAM ausgelesenes Datenwort besitzt eine Wortbreite von 18Bit, während über die UART-Schnittstelle nur 8 Bit bzw. einzelne Bytes übertragen werden können. Aus diesem Grund befindet sich zwischen dem BRAM-Ausgang und dem Eingang des UART-Moduls ein Multiplexer, der das Datenwort des SRAMS in Bytes unterteilt. Welches Byte des ausgelesenen BRAM-Datenworts an den UART übergeben wird entscheidet wieder die Zustandsmaschine. Sobald ein neues Byte am Ausgang des Multiplexeres anliegt, initiiert die Zustandsmaschine die Übermittlung des Bytes über die UART-Schnittstelle. Wenn die UART-Schnittstelle die Versendung des Bytes vollzogen hat, wird die Auslesung der nächsten Byte des aktuellen Datenwortes angestoßen. Sobald das Datenwort vollständig versendet worden ist, löst die Zustandsmaschine die Inkrementierung der BRAM-Adresse aus und liest das nächste Datenwort aus dem BRAM aus. Der Vorgang endet, sobald das BRAM vollständig ausgelesen und alle Daten über die UART-Schnittstelle übertragen worden sind. Sollen die Werte ausgelesen werden, die sich ohne Initialisierung nach dem Einschalten im BRAM befinden, kann die Beschreibung des BRAMs übersprungen werden und direkt mit der Auslesung begonnen werden.

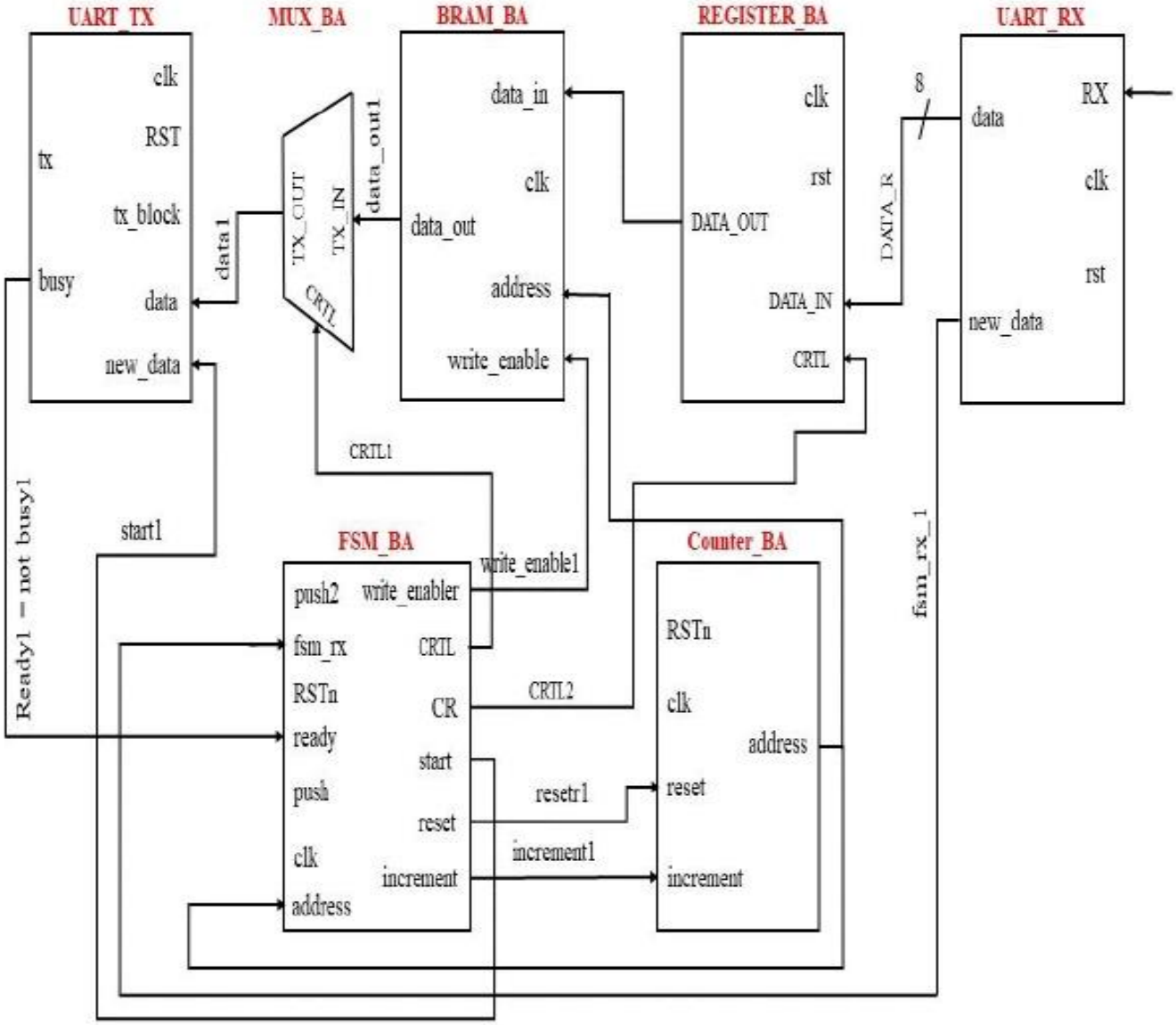

Abbildung 11: Blockdiagramm der Schaltung für die Beschreibung und Auslesung des BRAMs

6.2 Module

6.2.1 BRAM_BA

Im BRAM_BA ist eine BRAM-Instanz des FPGA vorhanden, welche dem integrierten SRAM Block entspricht. Sie stellt die eigentlichen Physically Unclonable Function Primitive dar, die in diesem Projekt untersucht werden soll.

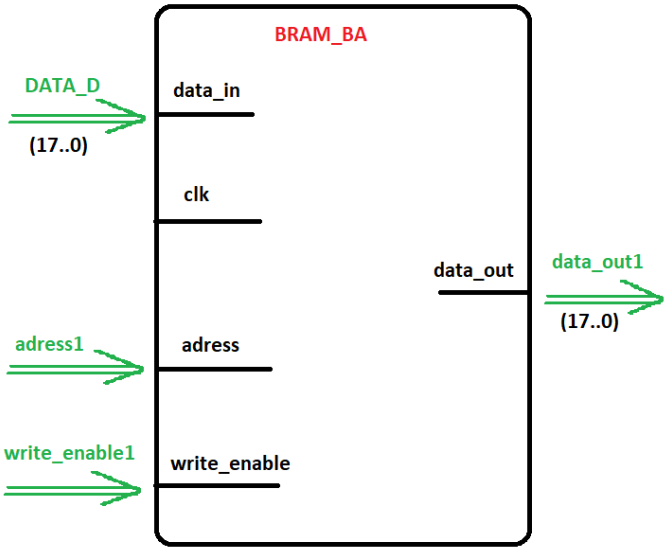

Abbildung 12: BRAM_BA Modul

Das BRAM_BA Modul besitzt die vier Eingänge clk, write_enable, data_in, address und den data_out Ausgang. Am Port adresse erhält das Modul die Adresse des zu schreibenden oder lesenden Datenwortes. Am data_in Port liegen die Daten an, die in den Speicher geschrieben werden. Das Signal write_enable bestimmt, ob Daten in das BRAM geschrieben (write_enable=1) oder gelesen (write_enable=0) werden. Als data_out wird der Ausgangsport bezeichnet, an dem mit der steigenden Flanke des Taktsignals clk die adressierten Datenwörter mit einer Wortlänge von 18 Bit anliegen.

6.2.2 Counter_BA

Count_BA führt eine Zählung von 0 bis 511 durch. Der Zähler wird erhöht, wenn das Inkrementsignal von der Zustandsmaschine gesetzt wird. Die Adresse des BRAM-Datenwortes, das geschrieben bzw. ausgelesen werden soll, besitzt eine Wortbreite von 8 Bit. Mit der steigenden Flanke und entsprechender Ansteuerung aktiviert der Port CLK, der dem Systemtakt entspricht, den Zähler. Das Signal RSTn repräsentiert einen Low-aktiven asynchronen Reset. Darüber hinaus verfügt das Modul auch über das high-aktive synchrone Resetsignal resetr1.

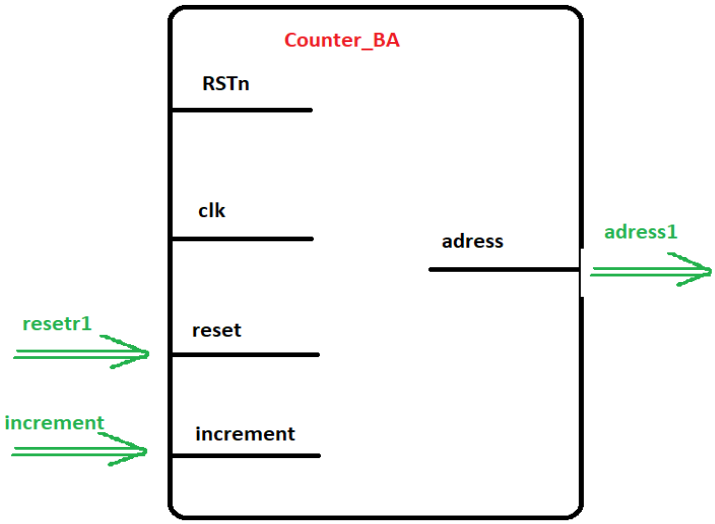

Abbildung 13: Counter_BA Modul

6.2.3 MUX_BA

Die Verarbeitung des 18-Bit-Datenworts an der aktuell ausgelesenen Adresse des BRAMs liegt in der Verantwortung des Moduls MUX_BA. Es besitzt einen Eingang TX_IN, dessen Daten in drei Bytes aufgeteilt und über den Port TX_OUT an den UART weitergeleitet werden. Die Weiterleitung des BRAM-Datenwortes wird durch das Signal CRTL gesteuert. Beim Wert „00“ wird das niederwertigste Byte [7:0] gesendet. Der Wert „01“ wird zum Senden des nächsthöheren Bytes [15:8] verwendet. Die beiden Datenbits mit der höchsten Wertigkeit [17:16] werden rechtsbündig weitergeleitet, wobei die MSB-Bits jedes Bytes mit Nullen aufgefüllt werden.

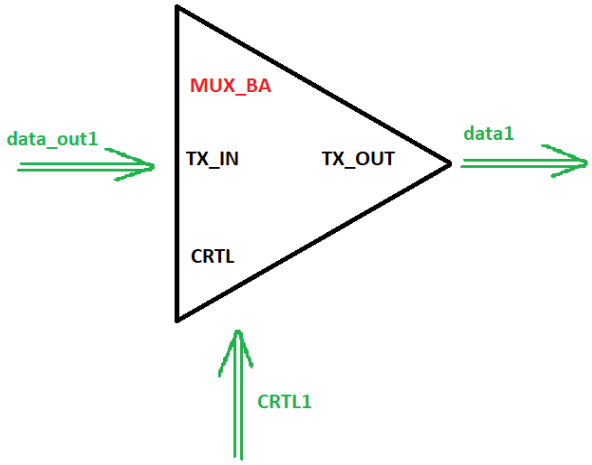

Abbildung 14: MUX_BA Modul

6.2.4 FSM_BA

Die Zustandsmaschine hat die Aufgabe die vollständige Einlesung und Auslesung des BRAMs und die Übermittlung und den Empfang der Daten über die UART Schnittstelle zu steuern. Wie in Abbildung 11 dargestellt ist, werden das Modul Register und BRAM so angesteuert, dass die 8 Bits , die vom Modul Uart_rx empfangen worden sind, an das BRAM in Form von 18 Bits angelegt werden. Der Empfang eines Bytes durch das Uart_rx Modul wird mit Hilfe des Signals fsm_rx_1 signalisiert. Der vollständige Empfang von 3 Bytes und der Zeitpunkt zur vollständigen Speicherung der Registerdaten wird dem BRAM über das Signal write_enable_1 bekannt gemacht. Nach der Übertragung eines Datenwortes in das BRAM wird dem Modul Counter_BA das Signal gegeben, die Adresse zu inkrementieren. Wenn alle 512 Adressen abgearbeitet worden sind, geht die State Maschine in den Ausgangszustand „idle“ zurück. Hier wartet die State Maschine darauf entweder neue Daten über den UART zu empfangen oder auf das Signal „Push“ , welches die Auslesung des BRAMS auslöst. Ist letzteres der Fall, werden die Module MUX_BA und UART so angesteuert, dass die 3 Bytes, die dem Datenwort des BRAMS an der aktuellen Adresse entsprechen, übertragen werden. Die Versendung eines Bytes über die UART Schnittstelle geschieht mit Hilfe des Signals start. Die vollständige Versendung eines Bytes über die UART Schnittstelle signalisiert der UART über das Signal ready. Nach der Übertragung des dritten Bytes wird dem Modul Counter_BA das Signal gegeben, die Adresse zu inkrementieren. Wenn alle 512 Adressen abgearbeitet worden sind, geht die State Maschine in den Ausgangszustand „idle“ zurück.

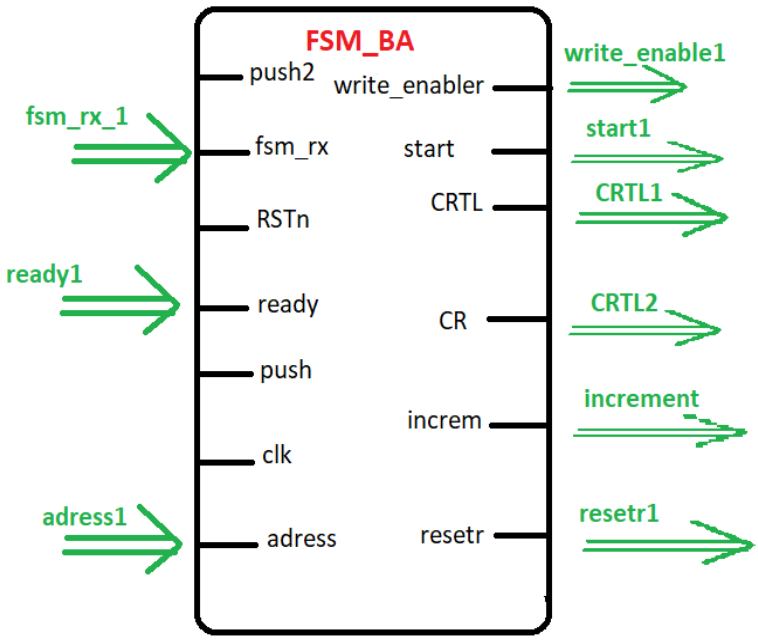

Abbildung 15: FSM_BA Modul

Das FSM_BA Modul besitzt sieben Eingänge push2, fsm_rx, RSTn, ready, push, clk und adress und die sechs Ausgänge write_enabler, Start, CRTL, CR, increment und resetr.

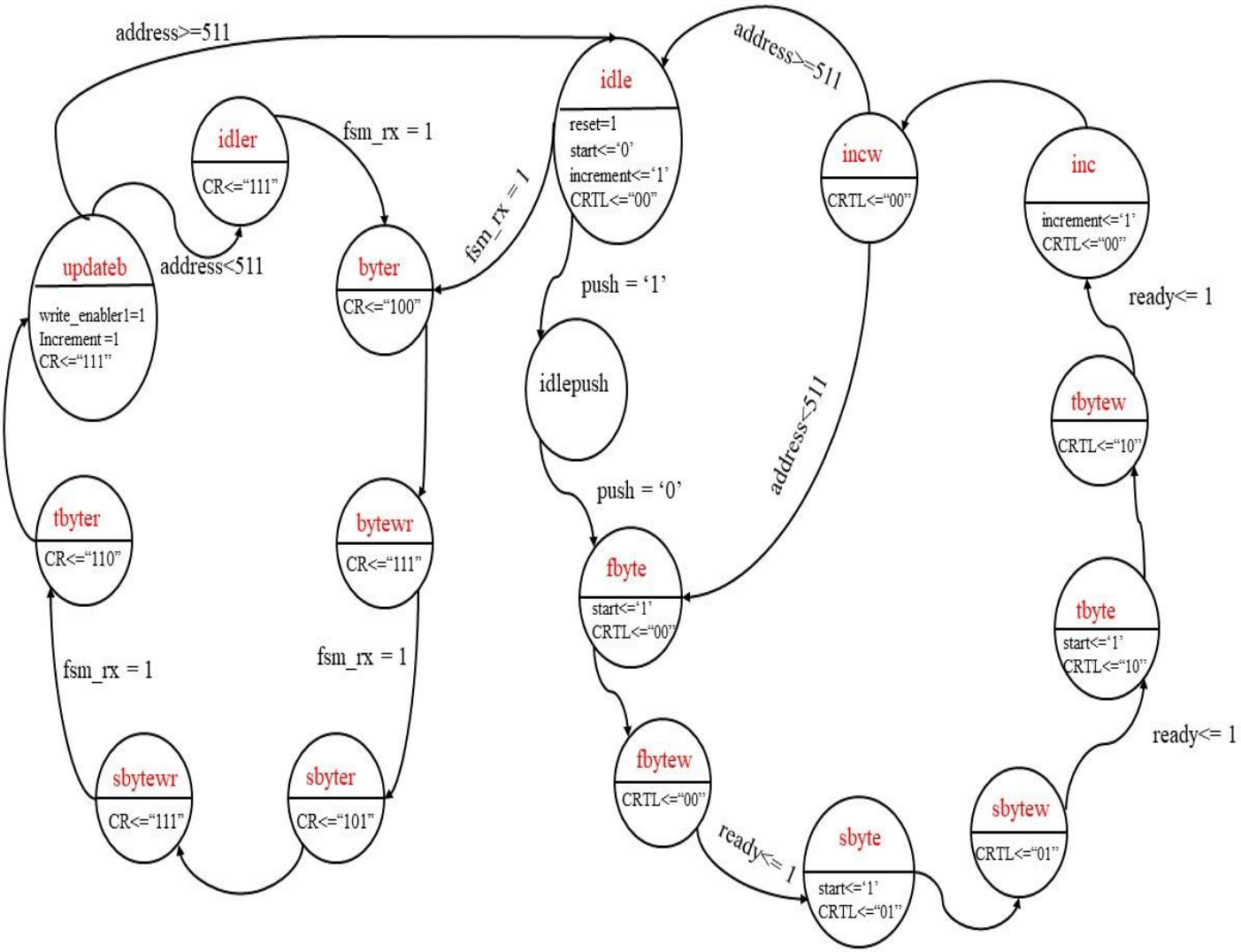

Abbildung 16: Zustandsdiagramm

Das Zustandsdiagramm in Abbildung 16 stellt die Zustände der Zustandsmaschine grafisch dar. Der initiale Zustand ist idle. Wenn der Empfang eines neuen Bytes über die UART Schnittstelle durch fsm_rx =`1´ signalisiert wird, wechselt die Zustandsmaschine in den Zustand byter um das niederwertigste Byte abzuspeichern. Danach wechselt sie in den Zustand bytewr, um auf den vollständigen Empfang des nächsten Bytes zu warten. Bei Empfang des nächsten Bytes wird wieder das Signal fsm_rx =`1´ gesetzt und die Zustandsmaschine wechselt in den Zustand sbyte, um die Abspeicherung des nächsten Bytes zu initiieren. Der Zustand wird danach erneut gewechselt, um im sbytewr Zustand wieder auf den vollständigen Empfang des nächsten Bytes zu warten. Wenn das Signal fsm_rx =`1´ gesetzt ist , wechselt die Zustandsmaschine in den Zustand tbyter und initiiert die Speicherung des dritten Bytes des aktuellen Datenwortes. Dann wechselt die Zustandsmaschine in den Zustand Updateb. Hier wird mit Hilfe der Signalen write_enabler und increment das vollständige Datenwort im BRAM gespeichert und die Adresse inkrementiert. Im Zustand inc wird verglichen, ob die aktuelle Adresse kleiner oder grösser als 511. Wenn die Adresse < 511 ist, ist der Schreibenvorgang noch nicht vollständig vollzogen und die Zustandsmaschine wechselt in den Zustand idelr. Wenn die Adresse > 511 ist, ist der Schreibenvorgang

beendet und die Zustandsmaschine wechselt wieder in den Zustand idle. Dort wartet sie entweder auf einen neuen Schreib- oder Auslesevorgang. Der Auslesevorgang startet wenn das Signal push=`1´ist . Dann geht die Zustandsmaschine in den Zustand idlepush. Dort wird gewartet, bis das Signal push wieder '0' wird. Dadurch soll vermieden werden, dass der Auslesevorgang mehrmals durchgeführt wird, wenn das Signal push zulange gesetzt bleibt. Danach wechselt die Zustandsmaschine in den Zustand fbyte, in dem für einen Takt das Signal start gesetzt wird, um das niederwertigste Byte des aktuellen Datenwortes zu versenden. Dann erfolgt ein Übergang in den fbytew Zustand, wo auf die vollständige Versendung des Bytes gewartet wird. Wenn ready =`1´ ist, wechselt die Zustandsmaschine in den Zustand sbyte, um die Versendung des nächst höherwertigen Bytes zu initiieren. Der Zustand wird danach gewechselt und im sbytew Zustand wird wieder auf die vollständige Versendung des Bytes gewartet. Wenn ready =`1´ ist, wechselt die Zustandsmaschine in den Zustand tbyte und initiiert die Versendung des dritten Bytes des aktuellen Datenworts. Der Zustand ändert sich danach auf tbytew und im tbytew wird drauf gewartet, dass das dritte Byte vollständig versendet worden ist. Wenn ready =`1´ geworden ist, wechselt die Zustandsmaschine in den Zustand inc und löst dort die Inkrementierung der Adresse aus. Dann wechselt die Zustandsmaschine in den Zustand incw. Im Zustand incw wird verglichen, ob die aktuelle Adresse kleiner oder grösser als 511 ist. Wenn die Adresse > 511 ist, ist der Auslesevorgang beendet und die Zustandsmaschine wechselt in den Zustand idle. Wenn die Adresse < 511 ist, ist der Auslesevorgang noch nicht vollständig vollzogen und die Zustandsmaschine wechselt in den Zustand fbyte.

6.2.5 Serial_tx

Das Modul serial_tx verwendet den UART-Standard zum Senden serieller Daten. Neben einem Schieberegister verfügt es über eine Zustandsmaschine. In diesem Abschnitt wird nicht näher auf das Modul eingegangen, da es aus dem Mojo-Projekt übernommen wurde. Das Modul wird im Kontext des Mojo Boards verwendet, um den Datenaustausch zwischen einem FPGA und einem Mikrocontroller über eine UART-Schnittstelle zu implementieren. Das MUX_BA-Modul übergibt jeweils 8 Bits an das serial_tx-Modul, welche dann über den tx-Ausgangspin an das USBUART-Modul übertragen werden, das über einen PMOD-Anschluss mit dem GateMate Evaluation Board verbunden ist.

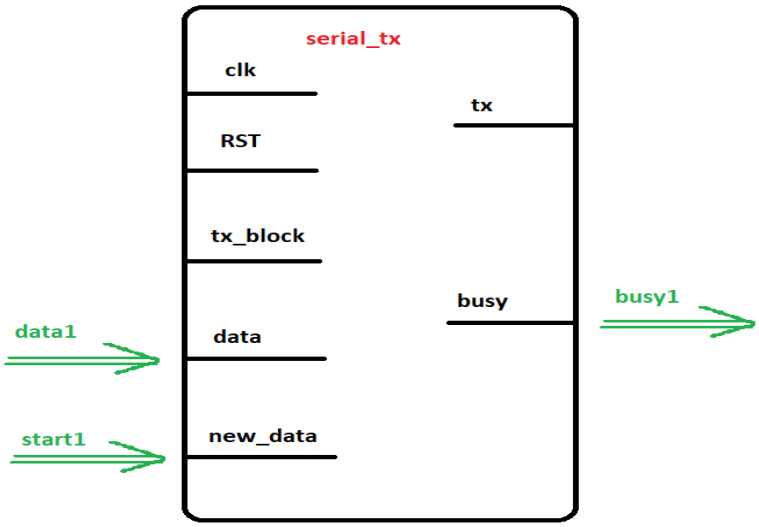

Abbildung 17: serial_tx Modul

Das Modul serial_tx verfügt über zwei Ausgänge und fünf Eingänge, darunter clk, rst, tx_block, new_data und data. Das Signal new_data leitet den Sendevorgang ein. Das Modul zeigt durch das Signal busy an, dass es mit der Versendung von Daten beschäftigt ist.

6.2.6 Serial_rx

Das Modul serial_rx empfängt einen seriellen Datenstrom aus 8 Bits und übergibt diese Daten als paralles Datenwort über das Signal DATA_R. Das serial_rx Modul besitzt die drei Eingänge clk, rst, und rx und die zwei Ausgänge data und new_data. Gestartet wird der Sendevorgang durch die Erkennung eines Startbits auf dem Signal RX. Durch das Signal fsm_rx_1 wird angezeigt, dass das Modul ein neues Byte vollständig erhalten hat.

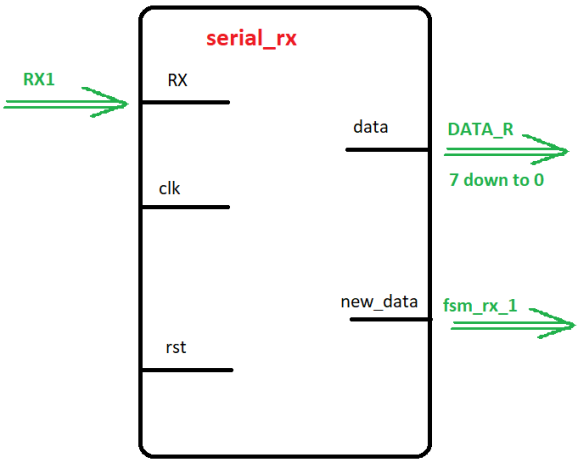

Abbildung 18: Serial_rx Modul

6.2.7 Counter_neu

Zur Entprellung des Tastersignals wurde das Modul counter_neu entwickelt. Es hat drei Eingänge clk, push_button_in und RSTn und der Ausgang push_button_out. Unabhängig vom Zustand des push_button_in-Signals wird das push_button_out-Signal hoch gehalten, sobald der Zähler den Wert 100 erreicht hat, wenn die mit dem push_button_in-Signal verbundene Taste mindestens 500 Takte lang gedrückt wird. Andernfalls wechselt sich das Signal push_button_in auf `1´ und passiert in dem Moment ebenfalls nichts weiter, da es keine Bedingung gibt, die auf eine steigende Flanke von push_button_in reagiert.

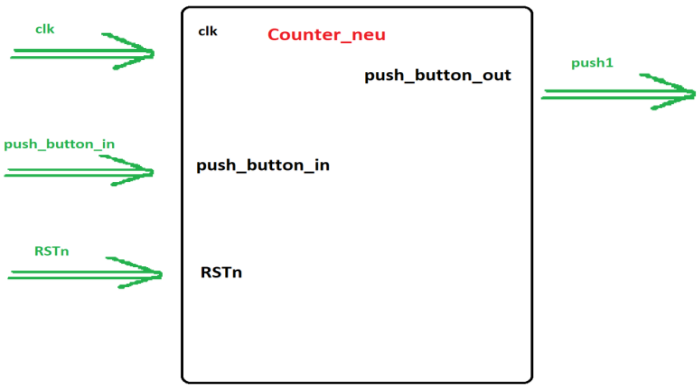

Abbildung 19: Counter_neu Modul

6.2.8 Register

Das Modul Register_BA hat die Aufgabe, die über das Modul Uart_rx empfangenen Bytes zu einem Datenwort zusammenzusetzen, und über den Port DATA_OUT an den Eingang des BRAMs anzulegen. An welcher Stelle das aktuell über das Modul uart_rx empfangene Datenwort abgelegt werden soll, wird durch den Steuereingang CRTL festgelegt. Der Port CRTL besitzt drei Bits, wobei der Wert "100" die Speicherung des niederwertigsten Bytes [7:0] und der Wert "101" die Speicherung des nächst höherwertigen Bytes [15:8] verursacht. Beim Wert "110" werden die hochwertigsten zwei Bits des Daten-wortes [17:16] gespeichert. Beim Wert "111" bleibt der aktuelle Inhalt des Registers erhalten und kann vom BRAM übernommen werden.

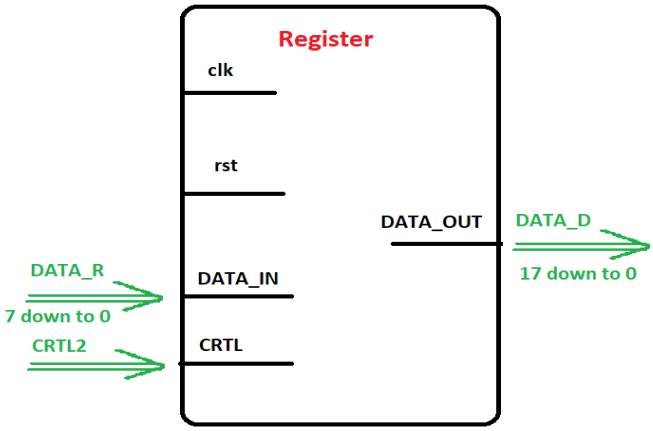

Abbildung 20: Register Modul

6.2.9 PUF_BA

Das Modul PUF_BA besteht aus den zwei Modulen MUX_BA und serial_tx. Das Modul hat die fünf Eingänge clk, rst, TX_IN, new_data und CRTL und den Ausgängen tx und busy. An den Port clk wird dabei der Systemtakt angelegt. Das Signal rst entspricht einem synchronen high-aktiven Reset. Die UART Daten liegen am Signal tx an.

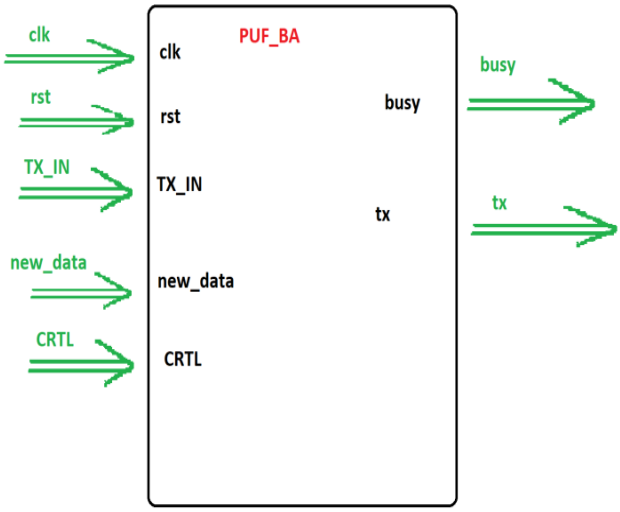

Abbildung 21: PUF_BA Modul

6.2.10 PUF_BA_REVERSE

Der PUF_BA_REVERSE besteht aus den zwei Modulen Register und Uart_rx. Das Modul hat die vier Eingänge clk, rst, CRTL, rx, new_data und DATA_OUT. An den Port clk wird dabei der Systemtakt angelegt. Rst entspricht einem synchronen high-aktiven Reset. Die UART Daten liegen dann am Signal rx an.

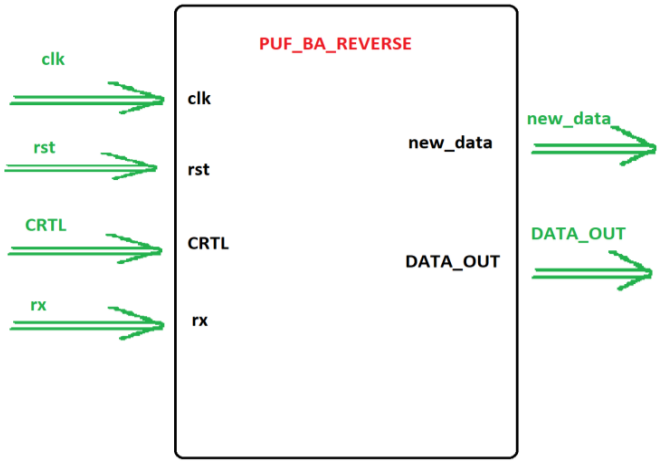

Abbildung 22: PUF_BA_REVERSE Modul

6.2.11 PUF_BAT

PUF_BAT entspricht dem Toplevel Modul des Schaltungsentwurfs und besteht aus den sechs Modulen PUF_BA (auslesen), PUF_BA_REVERSE (einlesen), counter_neu, Counter_BA, FSM_BA und BRAM_BA. Das Modul hat die Eingänge clk, RST, push_button_in, rx und den Ausgang tx.

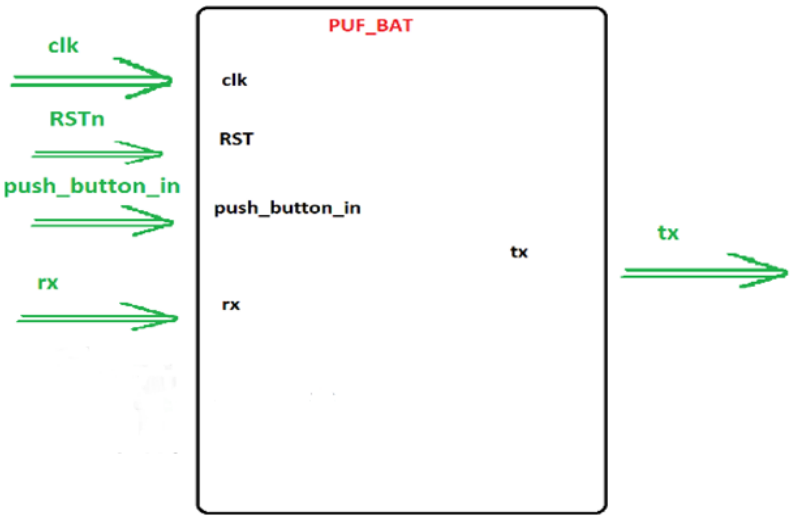

Abbildung 23: PUF_BAT Modul

6.3 Versuchsbeschreibung

6.3.1Versuchsaufbau

Abbildung 24: Aufbau der gesamten Versuch

Im folgenden Abschnitt wird der Versuchsaufbau im Detail beschrieben. Es werden die folgenden Komponenten verwendet.

- ✓ **Komponentenliste:**
- 1x PC
- 1x GateMateTM FPGA Cologne Chip
- 1x FTDI Device UM232H
- 1x PmodUSBUART
- 1x Spannungsversorger R&S®HMC8043
- 1x ThermoStream® Thermal Test Systems
- Verbindungskabel, Breadboard, Male to Male Kabel und Netzwerkkabel
- Qt Creator für die Benutzeroberflächenprogrammierung
- ✓ **Programmierung:**
  - Die Benutzeroberfläche, die mit Hilfe von Qt Creator erstellt wurde, wird kompiliert und ausgeführt.

- ✓ **Durchführung des Versuchs:**

Wie in Abbildung 24 dargestellt, werden alle Komponenten d.h. die Spannungsversorgung, der Thermostream, der PC sowie die FTDI Bausteine eingeschaltet und richtig verbunden. Wie in Abbildung 25 dargestellt, wird der GND Pin des UM232H Bausteins mit dem GND des PMOD A Stecker des FPGA sowie der AD4 Pin des UM232H mit dem entsprechenden GPIO Anschluss des FPGA verbunden. Der hat die Funktion eines Push-Buttons und wird verwendet den Beginn der Messung zu initiieren. Zunächst wird die grafische Benutzeroberfläche (GUI) in Abbildung 26 geöffnet und die minimale Temperatur auf -40 °C und die maximale Temperatur auf 60 °C eingestellt. Die Schrittweite beträgt 10 °C. Des Weiteren wird die Spannung von 0,9 bis 1,1 Volt in Schritten von 0,1 V eingestellt die Anzahl der Wiederholungen sowie die Parameter für die Wartezeiten zur Erreichung der Temperaturstabilität vor der ersten und zwischen den einzelnen Messungen festgelegt. Anschließend wird die Schaltfläche „Down" angeklickt, um den Kopf des Thermostreams nach unten zu bewegen, bevor schließlich die Schaltfläche „Start" betätigt wird. Die Elektronik, einschließlich des FPGA-Boards, befindet sich in einer geschlossenen Box. Diese Box sorgt für eine kontrollierte Testumgebung. Eine genaue Positionierung des thermostreamkopfes über der Elektronik ist notwendig, um eine schnelle und effiziente Anpassung der Komponenten-Temperatur zu gewährleisten. Dadurch wird die Temperaturschleife sowie die Spannungsschleife durchlaufen. Zur Einstellung der Spannung sowie zur Neuprogrammierung im FPGA wird die Funktion „reprogrammUndSetVoltage" aufgerufen (vgl. Abbildung 27). Um die Daten zu lesen und in der Datei „output.dat" zu speichern, wird die Funktion „readSerialData" aufgerufen. Am Ende, wenn alle Wiederholungen abgeschlossen sind, wird das Programm beendet.

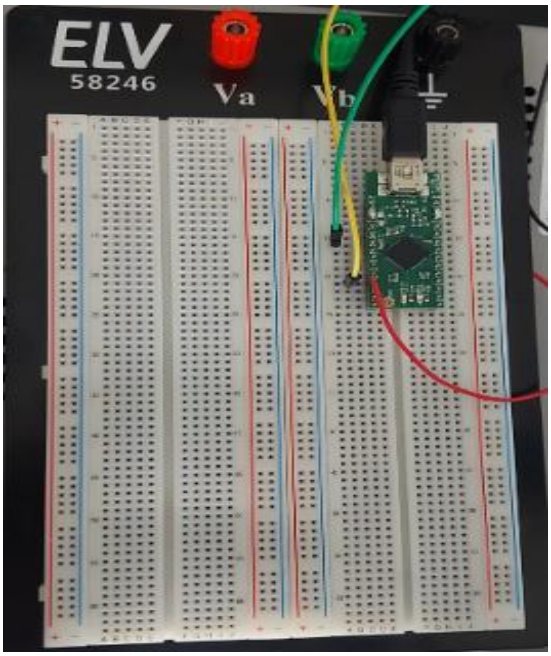

Abbildung 25: Beschaltung des UM232H FTDI Baustein auf einem Steckboard

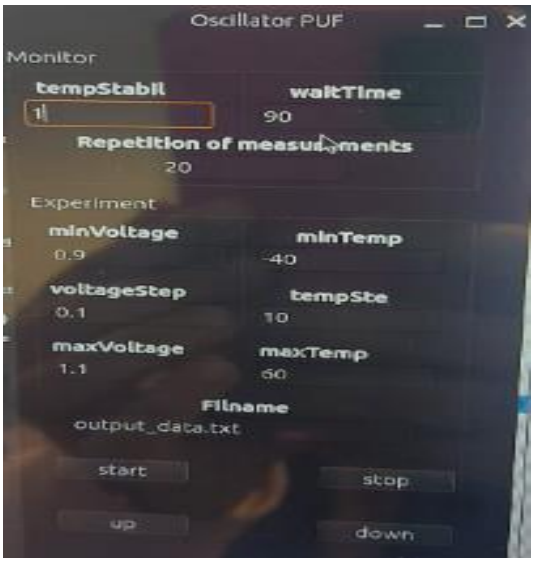

Abbildung 26: Grafische Benutzeroberfläche für die Durchführung des Versuchs

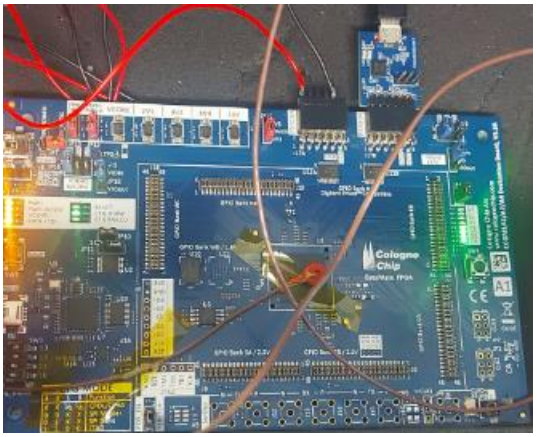

Abbildung 27: Konfiguration des FPGA's

7. Erstellung einer grafischen Benutzeroberfläche in Qt mit Definition der Parameter für die Messungen

7.1 Aufbau und Funktion der graphischen Benutzeroberfläche

Die folgende gestaltete GUI, welche in Abbildung 28 zu sehen ist, gliedert sich in sechs Bereiche. Diese sind nach der Reihenfolge ihrer Nutzung nummeriert. Im Einzelnen haben die Bedienelemente der Bereiche folgende Funktion.

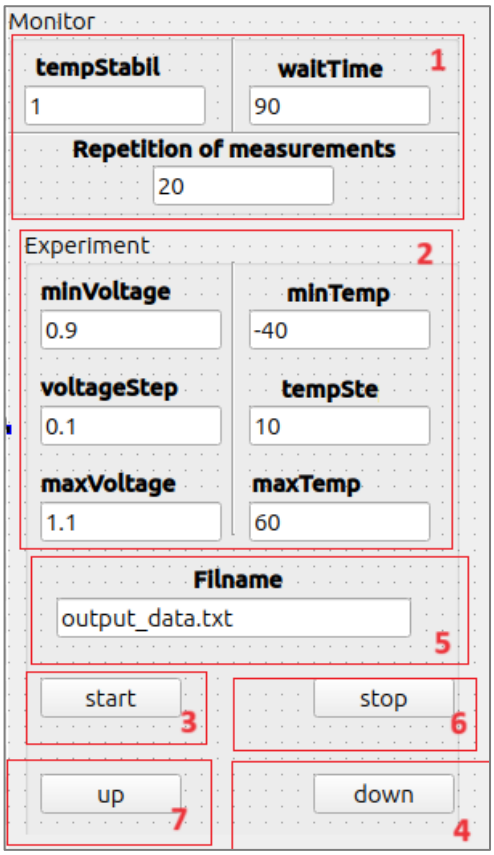

Abbildung 28: Design der Benutzeroberfläche

Im ersten Bereich erfolgt die Festlegung der Wartzeit zur Erreichung der Zieltemperatur vor Durchführung der ersten Messung, der Wartezeit zwischen den einzelnen Messungen und der Zahl der Messwiederholungen. Im zweiten Bereich erfolgt die Festlegung der minimalen Spannung, des Spannungsschritts, der maximalen Spannung, der minimalen Temperatur, des Temperaturschrittweite und der maximalen Temperatur. Im dritten Bereich befindet sich die Start-Taste zum Starten des Experiments. Im vierten Bereich befindet sich die Down-Taste, die dazu dient den Kopf des Thermostreams nach unten zu bewegen. Der Name der Datei zur Speicherung erfasster Messwerte ist im Bereich 5 einzugeben. Die Stop-Taste beendet das Experiment. Die Up-Taste bewegt den Kopf des Thermostream nach dem Ende des Experiments wieder nach oben. Im Anschluss erfolgt die Anpassung der Namen und Eigenschaften aller Objekte auf der grafischen Oberfläche. Daraufhin wird mit der Erstellung des Codes im Editiermodus begonnen.

7.2 Q-serial-Port

Für die Durchführung der seriellen Kommunikation in Qt, muss die Bibliothek für serielle Schnittstellen „QSerialPort“ eingebunden werden. Die Code-Zeile aus Quellcode 1 muss in einer ´´ .pro´´-Datei im Qt-Framework eingetragen werden.

```
QT      += core gui serialport
```

Quellcode 1: Verwendung der QSerialPort Bibliothek

Durch die Verwendung dieser Klasse können die Grundfunktionen der seriellen Schnittstelle genutzt werden. Dazu gehören das Öffnen, das Schreiben, das Lesen und das Schließen der seriellen Schnittstelle. Um die Funktionen der Bibliothek verwenden zu können, muss die folgende Datei inkludiert werden, die Funktionen definiert, welche die serielle Kommunikation ermöglichen.

```
#include <QtSerialPort >
```

Quellcode 2: Header Dateien in der worker.cpp

```
void Worker::run()

{

//..
QSerialPort serial;
serial.setBaudRate(9600);
  serial.setPortName("/dev/serial/by-id/usb-FTDI_FT232R_USB_UART_A503VSLI-if00-port0");
  if (!serial.open(QIODevice::ReadWrite))  {
    qDebug() << "ERROR: opening serial port" << serial.errorString();
  }
//..
serial.close();
}
```

Quellcode 3: Q-Serial-Port Instanz

In Quellcode 3 wird eine Q-Serial-Port-Instanz erzeugt und konfiguriert. Die Baudrate wird auf 9600 eingestellt und der Portname wird festgelegt. Danach wird versucht, die serielle Schnittstelle zu öffnen und zu lesen bzw. zu schreiben. Wenn das Öffnen des Ports fehlschlägt, wird eine Fehlermeldung ausgegeben. Um Daten von einem seriellen Gerät zu lesen, wird die Instanz Serial innerhalb der Methode run verwendet. Die serielle Schnittstelle wird am Ende der Methode geschlossen.

7.3 FTDI Device

Das ftdi device setzt ein Eingangssignal des FPGAs, welches die die Versendung der PUF Daten über die UART Schnittstelle verursacht. Hierfür wird in der Datei worker.cpp die Funktion ftdi_set_gpio genutzt,

um die GPIO-Pins des FTDI-Chips in der run-Methode der Worker-Klasse an mehreren Stellen zu steuern.

Eine beispielhafte Verwendung der Methode ftdi_set_gpio wird in den folgenden Code-Auszügen demonstriert.

```
ftdi_set_gpio(0x00);
```

und

```
ftdi_set_gpio(0xff);
```

Diese Aufrufe beeinflussen die GPIO-Pins des FTDI-Chips und setzen alle Ausgänge auf 0 oder 1. Die genaue Funktionalität dieser Pins hängt von der spezifischen Konfiguration und dem Design des verwendeten FTDI-Chips ab. Durch Verbindung eines Oszilloskops mit den entsprechenden Pins des FTDI-Geräts AD4 und GND können die High- und Low-Signalübergänge auf dem Oszilloskop abgelesen werden.

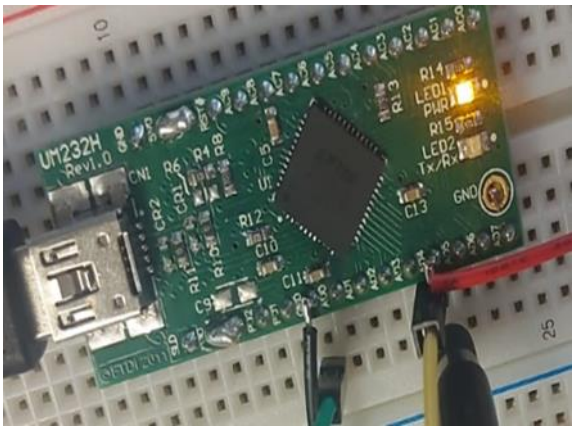

Abbildung 29: UM232H Impulse

Abbildung 30: Oszilloskop Screenshot, welcher den Übergang eines FTDI Ausgangssignals von Low nach High zeigt.

7.4 Code Beschreibung

Der Versuchsablauf für die Bestimmung des Einflusses von Temperatur und Versorgungsspannung des FPGAs auf die Antwort der PUF wird über eine Software gesteuert, welche mit Qt Creator entwickelt wurde. In der Datei worker.cpp werden die für die Versuchsdurchführung nötigen Methoden implementiert. Dazu gehören die Methoden setParameters, readSerialData, reprogramFPGA, reprogramAndSetVoltage, run, setMyFileName und stop, welche alle Mitglieder der worker Klasse sind. Die Aktualisierung der Parameter für die Datenerfassung erfolgt mittels der Methode Worker:: setParameters(). Im folgenden werden die Vorgänge in dieser Methode beschrieben:

Um zu gewährleisten, dass stets nur ein Thread auf die Daten zugreift, wird ein Mutex verwendet. Die Methode „QMutexLocker locker(&mutex)“ dient der Freigabe des Mutex. Die folgenden Variablen werden mit den der Methode übergebenen Werten initialisiert: minTemperature, temperatureStep, maxTemperature, minVoltage, voltageStep, maxVoltage, repeat, tempStabil und waitTime. Schließlich erfolgt die Ausgabe einer Debugmeldung, welche die aktualisierten Spannungsparameter (minVoltage, maxVoltage und voltageStep) beinhaltet.

```
void Worker::setParameters(float minTemperature, float temperatureStep,
                           float maxTemperature, double minVoltage,
                           double voltageStep, double maxVoltage, int repeat,
                           double tempStabil, double waitTime) {

  // Aktualisierung der Parameter des Workers
  QMutexLocker locker(&mutex);
  this->minTemperature = minTemperature;
  this->temperatureStep = temperatureStep;
  this->maxTemperature = maxTemperature;
  this->minVoltage = minVoltage;
  this->voltageStep = voltageStep;
  this->maxVoltage = maxVoltage;
  this->repeat = repeat;
  this->tempStabil = tempStabil;
  this->waitTime = waitTime;
  qDebug() << "minVoltage:" << minVoltage << "maxVoltage:" << maxVoltage
           << "voltageStep:" << voltageStep;
}
```

Quellcode 4: setParameters Methode

Wie in Quellcode 5 dargestellt, werden die Daten mit QSerialPort über die serielle Schnittstelle eingelesen und über QFile in die festgelegte Datei gespeichert. Dazu wird die Methode „readserial” verwendet. Die Methode „serial.waitForReadyRead” wird eingesetzt, um auf die Datenverfügbarkeit zu warten. Die Daten werden schließlich über QFile in die festgelegte Datei gespeichert.

```
bool Worker::readSerialData(QSerialPort &serial,int temperature,double volta
                                                int repetition) {

  for (int i = 0; i < 10 ; ++i) {
    int ready = serial.bytesAvailable();
    if (ready >= 1536)
      break;
    serial.waitForReadyRead(100);
    QThread::msleep(500);
   }

    if (serial.bytesAvailable() < 1536)
        return false;

  QByteArray m_read = serial.read(1536);
  QString m_reads = QString::fromLocal8Bit(m_read.toHex());
  if (!myFile.isOpen()) {
    myFile.setFileName("output_data.txt");
    myFile.open(QIODevice::WriteOnly | QIODevice::Text);
    QTextStream stream(&myFile);
    stream.setFieldAlignment(QTextStream::AlignRight);
    stream << qSetFieldWidth(10) << "temperature,"
           << qSetFieldWidth(10) << "voltage,"
           << qSetFieldWidth(15) << "repetition,"
           << qSetFieldWidth(15) << "Data," << '\n';
  }
  // die gelesenen Daten in die Ausgabedatei schreiben
  QTextStream stream(&myFile);
  stream.setFieldAlignment(QTextStream::AlignRight);
  stream << qSetFieldWidth(10) << (QString::number(temperature) + ",")
         << qSetFieldWidth(10) << (QString::number(voltage) + ",")
         << qSetFieldWidth(15) << (QString::number(repetition) + ",")
         << qSetFieldWidth(15) << ((m_reads) + ",") << '\n';

    return true;
}
```

Quellcode 5: readSerialData Methode

Die Methode reprogramFPGA () ist dafür verantwortlich, das FPGA neu zu programmieren und zu überprüfen, ob die Ausführung erfolgreich war. Der Pfad zur Bitstream-Datei, welche im Verzeichnis „/home/aya" gespeichert ist, lautet „/home/aya/PUF_BAT_00.cfg.bit". Der Name der Bitstream-Datei ist „PUF_BAT_00.cfg.bit". Die Erstellung dieser Datei erfolgt nach Ausführung des Befehls „Implement" in Windows während des VHDL-Designs und ihre Speicherung erfolgt anschließend im CC-Workspace.

Wie in Quellcode 6 dargestellt, erfolgt in der Methode „reprogramAndSetVoltage" zunächst das Setzen der Spannung sowie die Rekonfiguration des FPGAs. Im Anschluss wird die Spannung auf null gesetzt und anschließend auf die Zielspannung gebracht. Eine anschließende Wartezeit gewährleistet die Stabilisierung der Spannung.

```
void Worker::reprogramAndSetVoltage(double voltage) {

  if (set_voltage(0.0, ch1) != 0) {
    qDebug() << "Fehler beim Setzen der Spannung auf Null.";
    return;
  }



  if (set_voltage(voltage, ch1) != 0) {
    qDebug() << "Fehler beim Setzen der Spannung auf das Zielniveau.";
    return;
  }
  //Stabilisierungszeit
  QThread::msleep(1000);
  reprogramFPGA();
  QThread::msleep(1000);
}
```

Quellcode 6: reprogramAndSetVoltage

Die Methode „worker::run" koordiniert im Wesentlichen alle Schritte, die für die Erfassung und Verarbeitung der Daten erforderlich sind. Dazu zählt die schrittweise Veränderung der Temperatur innerhalb eines festgelegten Temperaturbereichs sowie die schrittweise Anlegung der Versorgungsspannung innerhalb eines festgelegten Spannungsbereichs. Für jedes Parameterpaar aus Temperatur und Versorgungsspannung wird die PUF-Antwort ausgelesen und in einer Datei gespeichert. Die PUF-Auslese wird durch das entsprechende Logiksignal mit Hilfe des FTDI-Moduls initiiert. Die Daten werden über QSerialPort über die serielle Schnittstelle eingelesen und über QFile in die festgelegte Datei gespeichert. Die Methode im Quellcode 7 „Worker::setMyFileName" wird verwendet, um den Dateinamen zu setzen. Am Ende dient die Methode „STOP" im Quellcode 8 zur Beendigung der Bearbeitung und zum Schließen der Datei.

```
void Worker::setMyFileName(const QString &newMyFileName) {
  myFileName = newMyFileName;
}
```

Quellcode 7: setMyFileName Methode

```
void Worker::stop() {
  running = false;
  myFile.close();
}
```

Quellcode 8: Stop Methode

7.5 Ergebnisse

7.5.1 Betrachtung auf Zuverlässigkeit

Zur Evaluierung der Zuverlässigkeit der generierten Schlüssel wird die Intra-Chip-Hamming-Distanz (HD) herangezogen. Dies bedeutet, dass gleiche Bitmuster für gleiche PUFs gelten. Die Intra-Hamming-Distanz sollte idealerweise den Wert 0 % und die Zuverlässigkeit den Wert 100 % aufweisen. Dies bedeutet, dass die PUF immer das gleiche Bitmuster erzeugen sollte. Zur Schätzung des Intra-Chip-HD wird eine n-Bit-Referenzantwort (R_i) extrahiert. Hierzu wird über alle Temperaturen und Spannung für jedes Datenwort des BRAMs die häufigste Bitkombination ermittelt. Die n-Bit-Antwort wird bei einer anderen Betriebsbedingung d.h. anderer Umgebungstemperatur oder anderer Versorgungsspannung mit einem Wert $\boldsymbol{R_i'}$ extrahiert. Anschließend werden m Proben von R_i' gesammelt. Der durchschnittliche Intra-Chip-HD wird für den Chip i wie folgt berechnet:

$$HD_{intra} = \frac{1}{m}\sum_{t=1}^{m} \frac{HD(R_i, R'i,t)}{n} 100\% \text{ [29]}$$

Wobei $R'_{i,t}$ die t-te Probe von Ri´ ist. $R'_{i,t}$ ist die PUF-Antwort für verschiedene Umgebungsbedingungen und verschiedene Versorgungsspannungen und HD_{intra} bezeichnet die durchschnittliche Anzahl der nicht reproduzierbaren PUF-Antwortbits.

$$\text{Zuverlässigkeit} = 100\% - HD_{intra} \text{ [29]}$$

Im Folgenden wird eine Beispielrechnung an Hand der Formeln für die Zuverlässigkeit präsentiert, wobei n die Anzahl der Bits in der Referenzantwort ist, R_i die Referenzantwort ist, $R_i^{`}$die unter anderen Betriebsbedingungen entnommene Antwort und m die Gesamtzahl der für R_i' gesammelten Proben ist. Wenn zum Beispiel 100 Proben von R_i' gesammelt werden, dann ist m=100. t ist eine Variable, die als Index für die Iteration verwendet wird. Sie reicht von 1 bis einschließlich m und steht für jede einzelne Probe innerhalb der Menge von m Proben. t nimmt also Werte von 1 bis m in Schritten von 1 an. So bezieht sich beispielsweise t = 1 auf die erste Probe, t = 2 auf die zweite Probe und so weiter bis t = m, was die letzte Probe darstellt. Die PUF-Reaktion $R'_{i,t}$ ist abhängig von den Umgebungsbedingungen sowie der Versorgungsspannung und variiert somit bei verschiedenen t-Werten.

In Abbildung 31 ist die Intra-Chip Hamming-Distanz der untersuchten SRAM-PUF als Funktion der Temperatur für verschiedene Versorgungsspannungen dargestellt. Wie der Abbildung zu entnehmen ist fällt die Hamming-Distanz mit zunehmender Temperatur. Die geringere Hamming-Distanz lässt auf eine höhere Stabilität und geringere Anfälligkeit der SRAM-Zellen für umweltbedingte Zustandsänderungen schließen, was eine zuverlässigere und konsistentere Funktion der PUF bei höheren Temperaturen erwarten lässt. Die Kurven deuten darauf hin, dass oberhalb der Raumtemperatur höhere Versorgungsspannungen die Stabilität der PUF-Antwort verbessern. Zusammenfassend lässt sich sagen, dass SRAM-PUF-Antworten bei höheren Temperaturen und höheren Versorgungsspannungen stabiler werden, was sich durch weniger Bitwechseln und geringere Hamming-Distanzen zeigt.

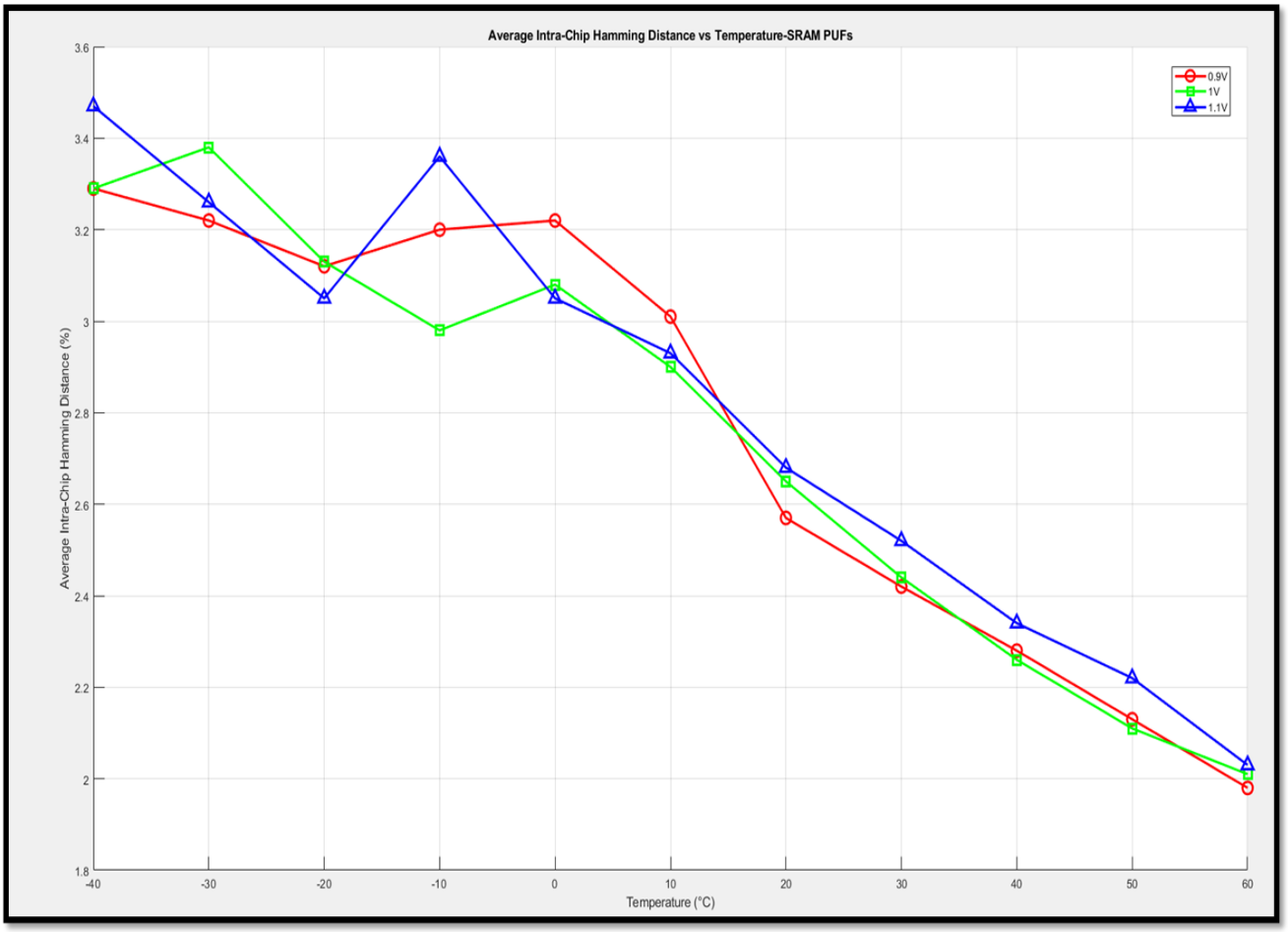

Abbildung 31: Intra-Chip Hamming-Distanz der untersuchten SRAM PUF als Funktion der Temperatur für verschiedene Versorgungsspannungen

7.5.2 Betrachtung auf Gleichmäßigkeit

Die Gleichmäßigkeit einer PUF quantifiziert den Grad der Konsistenz des Verhältnisses von Nullen und Einsen in den Antwortbits der PUF. Dabei ist die Bezeichnung 'Gleichmäßigkeit ' im Sinne der statistischen Signifikanz zu verstehen ist und erfasst den Anteil von Nullen und Einsen in den Antwortbits einer PUF. Die Gleichmäßigkeit ist definiert als:

$$u_n = \frac{1}{L}\sum_{l=0}^{L-1} R_{n,l^x} 100\%$$

$R_{(n,l)}$ ist das l-te Binärbit der Bitantwort mit der Bitlänge L. Der Idealwert der Gleichmäßigkeit liegt bei 50 %. [29]

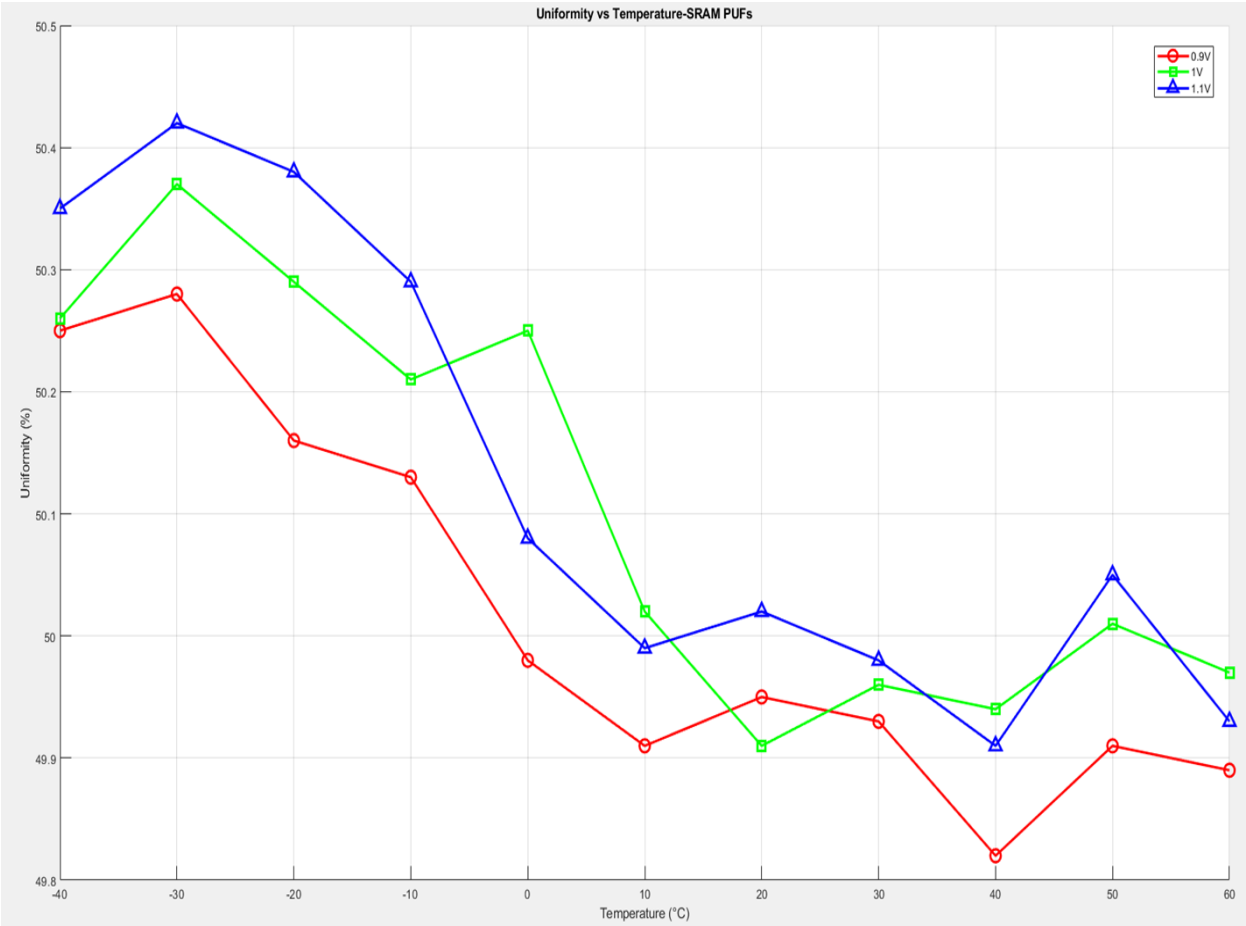

Abbildung 32: Einzigbarkeit vs Temperatur-SRAM PUFs

Aus Abbildung 32 wird ersichtlich, dass die Kurven für alle Spannungen eine leichte Abnahme der Gleichmäßigkeit mit steigender Temperatur aufweisen. Der Grad der Abnahme variiert jedoch mit der Spannung. Bei niedrigeren Spannungen (0,9 V) nimmt die Gleichmäßigkeit mit steigender Temperatur tendenziell schneller ab als bei höheren Spannungen (1 V und 1,1 V). Von den drei Spannungen scheint die Gleichmäßigkeit für 0,9 V am empfindlichsten auf Temperaturschwankungen zu reagieren.

7.5.3 Zufälligkeit

Zur Bewertung der Zufälligkeit generierter Schlüssel wird die minimale Entropie Hn herangezogen. Die im Allgemeinen zur Bewertung der Unsicherheit oder Zufälligkeit einer Bitfolge verwendet wird. Diese ist wie folgt definiert:

$$H_n = -\log_2 max(P_n, 1 - P_n)$$

P_n *ist dabei* die relative Häufigkeit des Auftretens von Einsen in den erzeugten Antwortbits der PUF.

$$P_n = \frac{1}{K \cdot T \cdot L} \sum_{k=1}^{k} \sum_{t=1}^{T} \sum_{l=1}^{L} b_{n,k,t,l}$$

Im Idealfall entspricht der Wert der minimalen Entropie H_n 100 %, während der Wert der Wahrscheinlichkeit P_n dem Wert 50 % entspricht. [30]

Die in der Formel verwendeten Größen haben dabei die folgende Bedeutung:

- K=Die Anzahl der verschiedenen IDs, die pro Gerät erzeugt werden.
- T=Die Anzahl der pro ID durchgeführten Tests.
- L=Die Länge einer ID.
- k= Der Index einer ID. Die k-te ID wird der Einfachheit halber mit k bezeichnet 1 ≤ k ≤ K.
- t= Der Index eines Tests. Der t-te Test wird der Einfachheit halber mit t bezeichnet.
- l= Die Bitposition einer ID. Das l-te Bit wird der Einfachheit halber mit l
- bn,k,l,t=Empirisch erzeugtes Antwortbit l der ID k im Test t. bn,k,t,l ∈{0, 1}. [30]

7.5.4 Betrachtung auf Einzigartigkeit

Zur quantitativen Bewertung der Einzigartigkeit der generierten Schlüssel wird die Inter-Hamming Distanz herangezogen, die wie folgt definiert ist:

$$D_{Pi,Pj}^{inter} = \frac{2}{k(k-1)} \sum_{j=1}^{k-1} \sum_{j=i+1}^{k} \frac{HD(Pi,Pj)}{n} . 100\%$$

Die Variablen i und j (i ≠ j) repräsentieren zwei Chips, die jeweils 2 Bit-Antworten liefern, nämlich Ri bzw. Rj für die Challenge C. HD entspricht Hamming-Distanz. K ist die Größe der Gerätepopulation bzw die Anzahl der Primitive, die miteinander verglichen werden, n ist die Gesamtzahl der Antwortbits einer PUF d.h. Bitlänge generierter Schlüssel. Im Idealfall ist die Inter-Hamming Distanz binomial verteilt und besitzt einen Erwartungswert von E(DinterPi;Pj) = 50%. [31]

Da in diesem Projekt nur eine SRAM Zelle betrachtet wurde, konnte die Inter-Hamming Distanz nicht ermittelt werden.

7.5.5 Betrachtung auf Bitfehler-Analyse

Die Bitfehleranalyse gibt den Prozentsatz der fehlerhaften Bits im Verhältnis zur Gesamtzahl der übertragenen oder verarbeiteten Bits an. Ein Bitfehler ist ein Bit, das zwischen den Binärwerten 0 und 1 wechselt, wenn es mehrmals hintereinander erzeugt wird. Der Bitfehler $\boldsymbol{BEn}$ ist definiert als die Summe aller HDs, die sich im Vergleich zwischen allen PUF Antworten R´n,w und seiner Referenzantwort Rref,n ergibt. L bezeichnet dabei die Gesamtzahl der Antwortbits in einer ID. Der ideale Wert für den BER liegt bei 0 %. Dies bedeutet, dass im Rahmen der Messung keine falschen Antworten eingehen. [32]

$$BE_n = \frac{1}{W} \sum_{w=1}^{W} \frac{HD(Rref,n,R'n,w)}{L} . 100\%$$

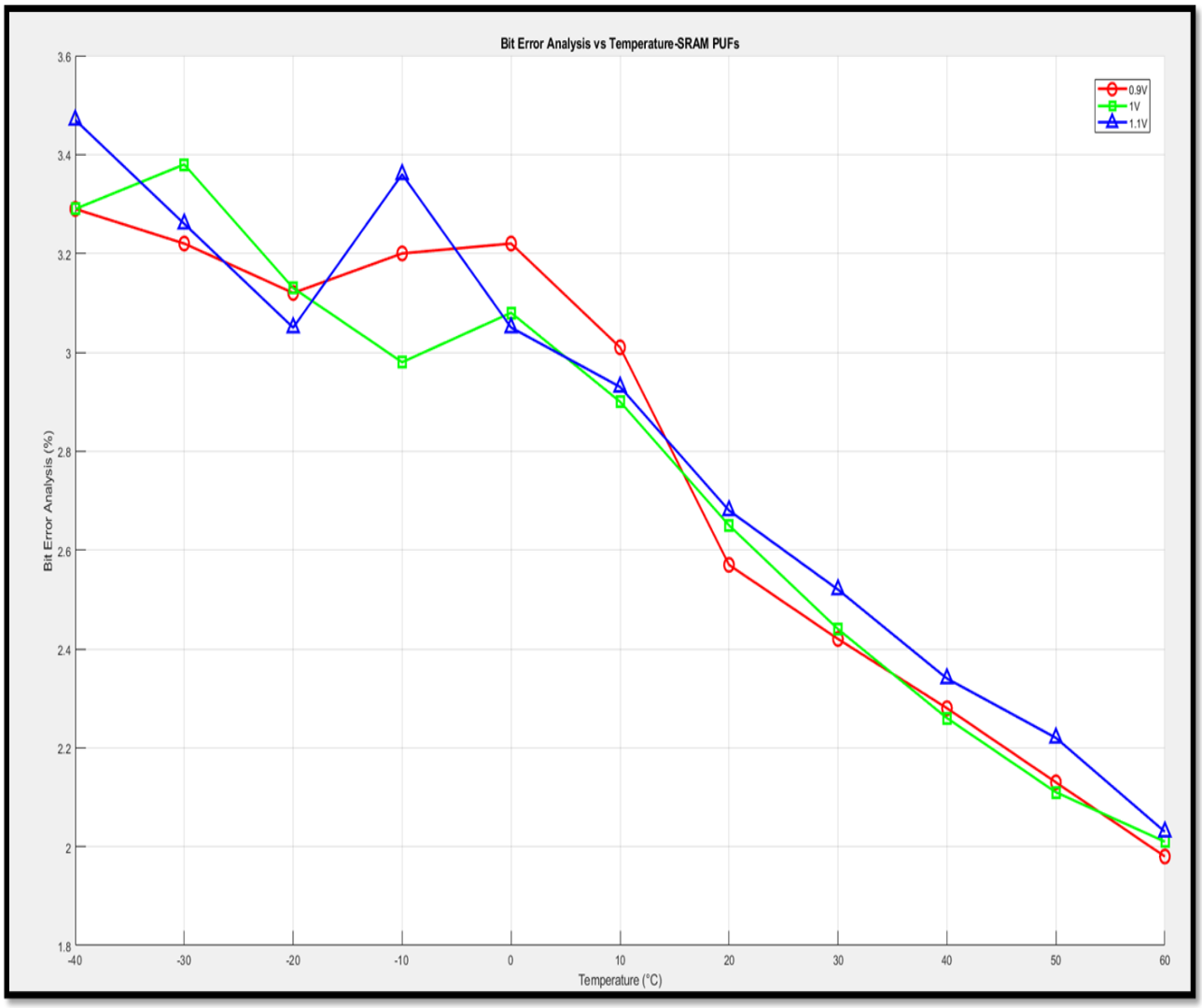

Abbildung 33: Bitfehleranalyse vs Temperatur SRAM PUFs

8.Fazit

Das Ziel dieses Projektes war es, herauszufinden, ob die in den GateMate FPGAs vorhandenen BRAMs als PUFs verwendet werden können. Hierfür wurden statistische Auswertungen unter Variation der Versorgungsspannung und der Betriebstemperatur durchgeführt. In der Einleitung wurde ein Überblick auf diese Arbeit und die Firma Cologne Chip gegeben. Im zweiten Kapitel wurde der PUF Begriff definiert und die Funktionsweise von SRAM basierten PUFs erläutert. Anschließend wurden in den Kapiteln 4 und 5 die verwendete Hardware und Software vorgestellt. Die Struktur des Schaltungsentwurfs und die Funktion der einzelnen Module wurden in Kapitel 6 vorgestellt. Abschließend wurde die Funktion des BRAMs in praktischen Versuchen unter Beweis gestellt. Die Messdaten wurden ausgewertet und es wurde überprüft, ob die Temperatur und die Versorgungsspannung einen Einfluss auf den BRAM haben.

9.Literaturverzeichnis

[1] Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions. Roel Maes, Ingrid Verbauwhede. Seite 2.

[2] Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions. Roel Maes, Ingrid Verbauwhede Seite 10, Seite 11.

[3] Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions. Roel Maes, Ingrid Verbauwhede Seite15, Seite 16.

[4] Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions. Roel Maes, Ingrid Verbauwhede Seite 15, Seite 16.

[5] Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. In: Cryptographic Hardware and Embedded Systems Workshop, LNCS, vol. 4727, pp. 63-80 (2007)

[6] Herschel, Sir William J.: The origin of nger-printing. Oxford University Press (1916)

[7] Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: Physical unclonable functions and publickey cry to for FPGA IP protection. In: Field Programmable Logic and Applications, 2007, pp. 189-195 (2007)

[8] Holcomb, D.E., Burleson, W.P., Fu, K.: Power-up sram state as an identifying ngerprint and source of true random numbers. IEEE Trans. Comput. 58(9), 1198-1210 (2009)

[9] „Pmod USBUART: USB to UART Interface ". Zugegriffen: 16.April.2024. [Online]. Verfügbar unter: https://digilent.com/shop/pmod-usbuart-usb-to-uart-interface/

[10] „Pmod BTN: 4 User Pushbuttons". Zugegriffen: 30.April.2024. [Online]. Verfügbar unter: https://digilent.com/shop/pmod-btn-4-user-pushbuttons/

[11] „Future Technology Devices International Ltd UM232H Single Channel USB Hi-Speed FT232H Development Module Datasheet", Seite 1,Seite . Zugegriffen: 9, April 2024. [Online]. Verfügbar unter: https://ftdichip.com/wp-content/uploads/2020/07/DS_UM232H.pdf

[12] „ R&S®HMC804x Power Supply 100 W and 1, 2 or 3 Channels", Seite2, Seite3. Zugegriffen: 9. April 2024. [Online]. Verfügbar unter: https://scdn.rohdeschwarz.com/ur/pws/dl_downloads/dl_common_library/dl_brochures_and_datasheets/pdf_1/HMC804x_dat_de_en_3607-0169-3x_v0200.pdf

[13] „R&S®HMC804x DC-Netzgeräteserie Ein Basisnetzgerät von R&S Essentials". Hauptmerkmale. Zugegriffen: 9. April 2024. [Online]. Verfügbar unter: https://www.rohde-schwarz.com/de/produkte/messtechnik/dc-netzgeraete/rs-hmc804x-dc-netzgeraeteserie_63493-61542.html

[14] „ThermoStream® Systems Precision Temperature Sources High speed thermal testing and conditioning". Seite 2, Seite 5. Zugegriffen: 9. April 2024. [Online]. Verfügbar unter: https://www.intestthermal.com/temptronic/thermostream

[15] Ubuntu – Wikipedia, 21. April 2022
https://de.wikipedia.org/wiki/Ubuntu

[16] Very High Speed Integrated Circuit Hardware Description Language, 2. Juni 2021
https://de.wikipedia.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language

[17] Qt documentation „QSerialPort Class“. Zugegriffen: 30.04.2024. [Online]. Verfügbar unter:
https://doc.qt.io/qt-6/qserialport.html

[18] 1. Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions. Roel Maes, Ingrid Verbauwhede. Seite 17.

[19] 1. Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions. Roel Maes, Ingrid Verbauwhede. Seite 17.

[20] 1. Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions. Roel Maes, Ingrid Verbauwhede. Seite 17.

[21] „Pmod USBUART: USB to UART Interface “. Zugegriffen: 16.April.2024. [Online]. Verfügbar unter:
https://digilent.com/shop/pmod-usbuart-usb-to-uart-interface/

[22] „Pmod USBUART: USB to UART Interface “. Zugegriffen: 16.April.2024. [Online]. Verfügbar unter:
https://digilent.com/shop/pmod-usbuart-usb-to-uart-interface/

[23] „Pmod BTN: 4 User Pushbuttons“. Zugegriffen: 30.April.2024. [Online]. Verfügbar unter:
https://digilent.com/shop/pmod-btn-4-user-pushbuttons/

[24] „Future Technology Devices International Ltd UM232H Single Channel USB Hi-Speed FT232H Development Module Datasheet“, Seite 2. Zugegriffen: 9, April 2024. [Online]. Verfügbar unter:
https://ftdichip.com/wp-content/uploads/2020/07/DS_UM232H.pdf

[25] „Future Technology Devices International Ltd UM232H Single Channel USB Hi-Speed FT232H Development Module Datasheet“, Seite 9. Zugegriffen: 9, April 2024. [Online]. Verfügbar unter:
https://ftdichip.com/wp-content/uploads/2020/07/DS_UM232H.pdf

[26] „ R&S®HMC804x Power Supply 100 W and 1, 2 or 3 Channels“. Seite2, Seite3. Zugegriffen: 9. April 2024. [Online]. Verfügbar unter:
https://scdn.rohdeschwarz.com/ur/pws/dl_downloads/dl_common_library/dl_brochures_and_datasheets/pdf_1/HMC804x_dat_de_en_3607-0169-3x_v0200.pdf

[27] „ThermoStream® Systems Precision Temperature Sources High speed thermal testing and conditioning“. Seite 2, Seite 1. Zugegriffen: 9. April 2024. [Online]. Verfügbar unter:
https://www.intestthermal.com/temptronic/thermostream

[28] „ThermoStream® Systems Precision Temperature Sources High speed thermal testing and conditioning“.Seite 2, Seite 5. Zugegriffen: 9. April 2024. [Online]. Verfügbar unter:
https://www.intestthermal.com/temptronic/thermostream

[29] Hori, Yohei & Yoshida, Takahiro & Katashita, Toshihiro & Satoh, Akashi. (2010). Quantitative and Statistical Performance Evaluation of Arbiter Physical Unclonable Functions on FPGAs. 298-303. 10.1109/ReConFig.2010.24
[29] Maiti, A., Gunreddy, V., Schaumont, P. (2013). A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions. In: Athanas, P., Pnevmatikatos, D., Sklavos, N. (eds) Embedded Systems Design with FPGAs. Springer, New York, NY. [Online]. Verfügbar unter: https://doi.org/10.1007/978-1-4614-1362-2_11
[30] Hori, Yohei & Yoshida, Takahiro & Katashita, Toshihiro & Satoh, Akashi. (2010). Quantitative and Statistical Performance Evaluation of Arbiter Physical Unclonable Functions on FPGAs. 298-303. 10.1109/ReConFig.2010.24
[31] Maiti, A., Gunreddy, V., Schaumont, P. (2013). A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions. In: Athanas, P., Pnevmatikatos, D., Sklavos, N. (eds) Embedded Systems Design with FPGAs. Springer, New York, NY. [Online]. Verfügbar unter: https://doi.org/10.1007/978-1-4614-1362-2_11
[32] Scholz, A.; Zimmermann, L.; Sikora, A.; Tahoori, M.B.; Aghassi-Hagmann, J. Embedded Analog Physical Unclonable Function System to Extract Reliable and Unique Security Keys. *Appl.Sci.* **2020**, *10*, 759. [Online]. Verfügbar unter: https://doi.org/10.3390/app10030759

10.Anhang

A.1 VHDL Codes

A.1.1 BRAM_BA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity BRAM_BA is
 generic (DATA_WIDTH : natural := 18;
         ADDR_WIDTH : natural := 9  );
  port (
      clk:        in  std_logic;
      data_in:    in  std_logic_vector (DATA_WIDTH-1 downto 0);
      address:    in  std_logic_vector (ADDR_WIDTH-1 downto 0);
      write_enable: in  std_logic;
      data_out:    out std_logic_vector (DATA_WIDTH-1 downto 0)
    );
end BRAM_BA;
architecture Behavioral of BRAM_BA is
 type ram is array(0 to 1024-1) of std_logic_vector(DATA_WIDTH-1 downto 0);
  signal mem : ram :=
  ( B"111111111111111111", B"111111111111111111", B"111111111111111111", B"111111111111111111",
B"111111111111111111", B"111111111111111111", B"111111111111111111", B"111111111111111111",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
```

```
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"1111111111111111", B"1111111111111111", B"1111111111111111", B"0000000000000001",
B"1111111111111111", B"1111111111111111", B"1111111111111111", B"1111111111111111",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"1111111111111111", B"1111111111111111", B"1111111111111111", B"0000000000000010",
B"1111111111111111", B"1111111111111111", B"1111111111111111", B"1111111111111111",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
```

```
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"111111111111111111", B"111111111111111111", B"111111111111111111", B"000000000000000111",
B"111111111111111111", B"111111111111111111", B"111111111111111111", B"111111111111111111",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"111111111111111111", B"111111111111111111", B"111111111111111111", B"000000000000001000",
B"111111111111111111", B"111111111111111111", B"111111111111111111", B"111111111111111111",
```

```
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"1111111111111111", B"1111111111111111", B"1111111111111111", B"0000000000000101",
B"1111111111111111", B"1111111111111111", B"1111111111111111", B"1111111111111111",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"1111111111111111", B"1111111111111111", B"1111111111111111", B"0000000000000110",
B"1111111111111111", B"1111111111111111", B"1111111111111111", B"1111111111111111",
```

```
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"1111111111111111", B"1111111111111111", B"1111111111111111", B"0000000000000001",
B"1111111111111111", B"1111111111111111", B"1111111111111111", B"1111111111111111",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
B"0000000000000000", B"0000000000000000", B"0000000000000000", B"0000000000000000",
```

```
B"111111111111111111", B"111111111111111111", B"111111111111111111", B"000000000000001010",
B"111111111111111111", B"111111111111111111", B"111111111111111111", B"111111111111111111",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"111111111111111111", B"111111111111111111", B"111111111111111111", B"000000000000001011",
B"111111111111111111", B"111111111111111111", B"111111111111111111", B"111111111111111111",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
```

```
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"111111111111111111", B"111111111111111111", B"111111111111111111", B"000000000000001100",
B"111111111111111111", B"111111111111111111", B"111111111111111111", B"111111111111111111",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000",
B"000000000000000000", B"000000000000000000", B"000000000000000000", B"000000000000000000" );
 begin
 process (clk)
 begin
  if rising_edge(clk) then
   if (write_enable = '1') then
    mem(to_integer(unsigned(address))) <= data_in;
   else
   data_out <= mem(to_integer(unsigned(address)));
    end if;
  end if;
 end process;

end Behavioral;
```

A.1.2 Counter_BA

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
entity Counter_BA is
 generic (
   ADDR_WIDTH : natural := 9 );
   Port (
      RSTn:          in std_logic;
      clk:           in std_logic;
      reset:         in std_logic; --high aktiv
      address:       out std_logic_vector (ADDR_WIDTH-1 downto 0);
      increment:     in std_logic  );
   end Counter_BA;
         architecture Behavioral of Counter_BA is
   signal counter_up: unsigned(8 downto 0);
      begin
   -- UP COUNTER
   process(clk,RSTn)
   begin
      if RSTn='0' then
         counter_up <= (others=>'0');
      elsif rising_edge(clk) then
         if(reset = '1') then
            counter_up <= (others=>'0');
         elsif (increment='1') then
            counter_up <= Counter_up + "1";
         end if;
      end if;
   end process;
   address <= std_logic_vector(counter_up);
   end Behavioral;
```

A.1.3 Counter_neu

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

entity counter_neu is
generic (
   top : natural := 250000
   );
     Port ( clk, push_button_in, RSTn: in std_logic;
          push_button_out: out std_logic
          );
end counter_neu;
architecture Behavioral of counter_neu is
signal counter_up: integer range 0 to top;
signal button_old: std_logic;
signal s_push_button_out:std_logic;
begin
   push_button_out<=s_push_button_out;
process(clk,RSTn)
   begin
      if RSTn='0' then
         counter_up <= 0;
         button_old <= '0';
         s_push_button_out <= '0';
      elsif (rising_edge(clk)) then
         if push_button_in = '0' and button_old = '1' then
            s_push_button_out <= '1';
            counter_up <= 0;
```

```
elsif counter_up  = 100 then
            s_push_button_out  <= '0';
            counter_up  <= 0;
         elsif s_push_button_out  = '1' then
            counter_up  <= counter_up  + 1;
         end if;
         button_old  <= push_button_in;
      end if;
   end process;
end Behavioral;
```

A.1.4 FSM_BA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity FSM_BA is
generic (

   ADDR_WIDTH : natural := 9
  );
   port (
clk:       in std_logic;
      fsm_rx:  in std_logic;
      RSTn:      in std_logic;
      ready:      in std_logic;
      address:    in  std_logic_vector  (ADDR_WIDTH-1 downto 0);
      push:       in std_logic;
      CRTL :     out std_logic_vector(1  downto  0);
      CR :       out std_logic_vector(2  downto 0);
      start:     out std_logic;
       increment:   out std_logic;
         write_enabler: out  std_logic;
      reset :out std_logic
  );
end FSM_BA;

architecture behavioral  of FSM_BA is
type
state_typeis(idle,idlepush,fbyte,fbytew,sbyte,sbytew,tbyte,tbytew,inc,incw,idelr,byter,bytewr,sbyter,sbytewr,tby
ter,updateb);


signal current_s, next_s: state_type;
```

```
begin

process (clk,RSTn)
   begin
      if RSTn='0' then
         current_s <= idle;
      elsif (rising_edge(clk))  then
            if(reset = '1') then
         current_s <= next_s;   --state change.
      end if;
end process;
--state machine process.
process (current_s, push, ready, address,fsm_rx)
begin
case current_s is
when idle =>
         if (push = '1') then
            next_s  <= idlepush;
         elsif (fsm_rx ='1') then
--             next_s  <= byter;
            next_s  <= idle;
         else
            next_s <= idle;
         end if;

       when idlepush =>
         if(push='0')  then
         next_s <= fbyte;
         else
         next_s <= idlepush;
         end if;
      when fbyte =>
            next_s<= fbytew;
           when fbytew =>
         if(ready ='1') then
            next_s <= sbyte ;
         else  next_s <= fbytew;
        end if;
```

```
when sbyte =>
            next_s <= sbytew;
  when sbytew =>
        if(ready ='1') then
            next_s <= tbyte;
        else  next_s <= sbytew;
      end if;
       when tbyte =>
            next_s <= tbytew;
      when tbytew =>
        if(ready ='1') then
            next_s <= inc;
        else next_s <= tbytew;
      end if;
      when inc =>
        if  (unsigned(address)  < "111111111")  then
            next_s <= fbyte;
        else next_s <= idle;
         end if;
      when  idelr =>
       if (fsm_rx='1')  then
       next_s <= byter;
       else next_s <= idelr;
       end if;
      when byter =>
       next_s <= bytewr;
        when bytewr =>
       if (fsm_rx='1')  then
       next_s <= sbyter;
       else  next_s <= bytewr;
       end if;
```

```
      when sbyter =>
      next_s <= sbytewr;
      when sbytewr =>
      if (fsm_rx='1') then
      next_s <= tbyter;
      else  next_s <= sbytewr;
      end if;
  when tbyter =>
      next_s <= updateb;
when updateb =>   if  (unsigned(address)  < "111111111")  then
             next_s <= idelr;
          else   next_s <= idle;
           end if;
        end case;
end process;
process(current_s)
begin
   start <= '0';
   increment<= '0';
   CRTL<= "00";
   CR<= "100";
   write_enabler<='0';
   resetr<= '0';
   case current_s is
      when idle =>
       start <= '0';
        increment<=  '0';
        CRTL<= "00";
when idlepush =>
        start <= '0';
        increment<=  '0';
        CRTL<= "00";
        reset<='1';
        when fbyte =>
          start <= '1';
CRTL <= "00";
          increment<='0';
```

```
    when fbytew =>
        CRTL <= "00";
         when sbyte =>
        start <= '1';
        CRTL <= "01";
      when sbytew =>
        CRTL <= "01";
          when tbyte =>
        start <= '1';
        CRTL <= "10";
           when tbytew =>
        CRTL <= "10";
       when inc =>
        increment<='1';
      when idelr =>
        CR<= "111";
     when byter =>
        CR<= "100";
     when bytewr =>
        CR<= "111";
     when sbyter =>
        CR<= "101";
      when sbytewr =>
        CR<= "111";
      when tbyter =>
        CR<= "110";
     when updateb =>
        write_enabler<='1';
        increment<='1';
        CR<= "111";
   end case;
end process;
end behavioral;
```

A.1.5 MUX_BA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX_BA is
generic (
   DATA_WIDTH : natural := 18;
   ADDR_WIDTH : natural := 9
 );
   port(
      TX_IN :    in std_logic_vector (DATA_WIDTH-1 downto 0);
      TX_OUT :   out std_logic_vector (7 downto 0);
      CRTL :     in std_logic_vector(1 downto 0)
       );
end MUX_BA;

architecture rtl of MUX_BA is
 -- declarative part: empty
   begin
      p_MUX_BA : process(CRTL,TX_IN)
   begin
      case CRTL is
         when "00"   => TX_OUT <= TX_IN(7 downto 0);
         when "01"   => TX_OUT <= TX_IN(15 downto 8);
         when "10"   => TX_OUT <= "000000" & TX_IN(17 downto 16);
         when others => TX_OUT <= "00000000";
   end case;
end process p_MUX_BA;

end rtl;
```

A.1.6 Register_RX

```
library ieee;
use ieee.std_logic_1164.all;

entity Register_RX is
   port (
      clk : in std_logic;
      rst : in std_logic;
      CRTL      : in std_logic_vector(2  downto 0);
      DATA_IN : in std_logic_vector(7  downto 0);
      DATA_OUT : out std_logic_vector(17  downto 0)
   );
end entity Register_RX;
architecture rtl of Register_RX is
   signal reg : std_logic_vector(17  downto 0);
begin
   process (clk, rst)
   begin
      if (rst = '1') then
         reg <= (others => '0');
      elsif (rising_edge(clk))  then
               case CRTL is
                  when "100"  => reg(7 downto 0) <= DATA_IN;
                  when "101"  => reg(15 downto 8) <= DATA_IN;
                  when "110"  =>  reg(17 downto 16)<= DATA_IN(1 down-to 0);
                  when "111"  => reg <= reg;
                  when others => reg <= (others => '0');
            end case;
      end if;
   end process;
 DATA_OUT <= reg;
end architecture rtl;
```

A.1.7 Serial_rx

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity serial_rx is
   generic (
      CLK_PER_BIT : natural := 1042;
      CTR_SIZE    : natural := 11
   );
   port (
      clk        : in  std_logic;
      rst        : in  std_logic;
      rx         : in  std_logic;
      new_data   : out std_logic;
      data       : out std_logic_vector(7 downto 0)
   );
end entity serial_rx;

architecture RTL of serial_rx is

type rx_state_type is (IDLE, WAIT_HALF, WAIT_FULL, WAIT_HIGH);

signal state_d, state_q        : rx_state_type ;
signal ctr_d, ctr_q            : unsigned(CTR_SIZE-1 downto 0);
signal bit_ctr_d, bit_ctr_q    : unsigned(2 downto 0);
signal data_d, data_q          : std_logic_vector(7 downto 0);
signal new_data_d, new_data_q  : std_logic;
signal rx_d, rx_q              : std_logic;

begin
```

```
new_data    <= new_data_q;
data        <= data_q;

rx_comb: process(rx, rx_q, state_q, ctr_q, bit_ctr_q, data_q)
begin
   rx_d        <= rx;
   state_d     <= state_q;
   ctr_d       <= ctr_q;
   bit_ctr_d   <= bit_ctr_q;
   data_d      <= data_q;
   new_data_d  <= '0';

   case state_q is
      when IDLE =>
         bit_ctr_d   <= "000";
         ctr_d       <= (others => '0');
         if rx_q = '0' then
state_d     <= WAIT_HALF;
         end if;
      when WAIT_HALF =>
         ctr_d       <= ctr_q + 1;
         if ctr_q = (CLK_PER_BIT/2) then
            ctr_d       <= (others => '0');
            state_d     <= WAIT_FULL;
         end if;
      when WAIT_FULL =>
         ctr_d       <= ctr_q + 1;
         if ctr_q = (CLK_PER_BIT-1) then
            data_d      <= rx_q & data_q(7 downto 1);
            bit_ctr_d   <= bit_ctr_q + 1;
            ctr_d       <= (others => '0');
            if bit_ctr_q = 7 then
```

```
state_d     <= WAIT_HIGH;
                new_data_d  <= '1';
            end if;
         end if;
      when WAIT_HIGH =>
         if rx_q = '1' then
            state_d     <= IDLE;
         end if;
      when others =>
         state_d     <= IDLE;
   end case;
end process rx_comb;

rx_seq: process(clk,  rst)
begin
   if rst = '1' then
      ctr_q       <= (others => '0');
      bit_ctr_q   <= "000";
      new_data_q  <= '0';
      state_q     <= IDLE;
      rx_q        <= '1';
      data_q      <= "00000000";
   elsif rising_edge(clk)  then
      ctr_q       <= ctr_d;
      bit_ctr_q   <= bit_ctr_d;
      new_data_q  <= new_data_d;
      state_q     <= state_d;
rx_q        <= rx_d;
      data_q      <= data_d;
   end if;
end process rx_seq;
end RTL;
```

A.1.8 serial_tx

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity serial_tx is
   generic (
      CLK_PER_BIT : natural := 1042;
      CTR_SIZE    : natural := 11
   );
   port (
      clk        : in  std_logic;
      rst        : in  std_logic;
      tx         : out std_logic;
      tx_block   : in  std_logic;
      busy       : out std_logic;
      new_data   : in  std_logic;
      data       : in  std_logic_vector(7 downto 0)
   );
end entity serial_tx;
architecture RTL of serial_tx is

type tx_state_type is (IDLE, START_BIT, DATA_BITS, STOP_BIT);

signal state_d, state_q     : tx_state_type;
signal ctr_d, ctr_q         : unsigned(CTR_SIZE-1 downto 0);
signal bit_ctr_d, bit_ctr_q : unsigned(2 downto 0);
signal data_d, data_q       : std_logic_vector(7 downto 0);
signal tx_d, tx_q           : std_logic;
signal busy_r               : std_logic;
signal block_d, block_q     : std_logic;
```

```
begin

tx      <= tx_q;
busy    <= busy_r;

tx_comb:  process(tx_block,  block_q,  ctr_q, bit_ctr_q,  data_q, state_q, da-ta, new_data)
begin
   block_d     <= tx_block;
   ctr_d       <= ctr_q;
   bit_ctr_d   <= bit_ctr_q;
   data_d      <= data_q;
   state_d     <= state_q;
   tx_d        <=  '0';
   busy_r <=   '0';

   case state_q is
when IDLE =>
         if block_q = '1' then
            busy_r      <= '1';
            tx_d        <= '1';
         else
            busy_r      <=  '0';
            tx_d        <=  '1';
            bit_ctr_d   <= "000";
            ctr_d       <=  (others => '0');
            if new_data  = '1' then
               data_d      <= data;
               state_d     <= START_BIT;
               --busy_r        <= '1';
            end if;
         end if;
```

```
      when START_BIT =>
         busy_r  <= '1';
         ctr_d   <= ctr_q + 1;
         tx_d    <= '0';
         if ctr_q = (CLK_PER_BIT-1) then
            ctr_d   <= (others => '0');
            state_d <= DATA_BITS;
         end if;
      when DATA_BITS =>
         busy_r  <= '1';
         tx_d    <= data_q(0);
         ctr_d   <= ctr_q + 1;
         if ctr_q = (CLK_PER_BIT-1) then
            data_d      <= "0" & data_q(7 downto 1);
            ctr_d       <= (others => '0');
            bit_ctr_d   <= bit_ctr_q + 1;
            if bit_ctr_q = 7 then
               state_d <= STOP_BIT;
            end if;
         end if;
when STOP_BIT =>
         busy_r  <= '1';
         tx_d    <= '1';
         ctr_d   <= ctr_q + 1;
         if ctr_q = (CLK_PER_BIT-1) then
            state_d <= IDLE;
         end if;
      when others =>
         busy_r  <= '1';
         tx_d    <= '1';
         state_d <= IDLE;
   end case;
end process tx_comb;
```

```
tx_seq: process(clk, rst)
begin
   if rst = '1' then
      state_q <= IDLE;
      tx_q    <= '1';

   elsif rising_edge(clk)  then
      state_q <= state_d;
      tx_q    <= tx_d;

      block_q     <= block_d;
      data_q      <= data_d;
      bit_ctr_q   <= bit_ctr_d;
      ctr_q       <= ctr_d;
   end if;
end process tx_seq;

end RTL;
```

A.1.9 PUF_BAT

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity PUF_BAT is
generic ( DATA_WIDTH : natural := 18;
          ADDR_WIDTH : natural := 9);

   Port ( clk: in std_logic;
     RSTP: in std_logic;
     push_button_in: in std_logic;
          rx: in std_logic;
 tx: out std_logic
           );
end PUF_BAT;

architecture Behavioral of PUF_BAT is
component counter_neu is
generic (
   top : natural := 250000);
```

```
Port (  clk, push_button_in,RSTn:  in std_logic;
          push_button_out:  out std_logic
          );
end component;

component  PUF_BA is
generic  (
          DATA_WIDTH : natural := 18;
          ADDR_WIDTH : natural  := 9
       );
    Port (  clk,rst: in std_logic;
         TX_IN :   in std_logic_vector  (DATA_WIDTH-1  downto 0);
      busy    : out std_logic;
      new_data  : in  std_logic;
      CRTL :    in std_logic_vector(1  downto 0);
          tx: out std_logic
          );
end component;

component  BRAM_BA is
 generic  (
          DATA_WIDTH : natural := 18;
          ADDR_WIDTH : natural  := 9
          );
 port  (
       clk:         in  std_logic;
       data_in:      in  std_logic_vector  (DATA_WIDTH-1  downto 0);
       address:      in  std_logic_vector  (ADDR_WIDTH-1 downto 0);
       write_enable:  in  std_logic;
       data_out:     out std_logic_vector  (DATA_WIDTH-1  downto 0));
end component;
```

```
component  FSM_BA is
 generic  (
    ADDR_WIDTH : natural  := 9
  );
    port  (
clk     :in std_logic;
    fsm_rx:in  std_logic;
    RSTn    :in std_logic;
    ready     :in std_logic;
    address     :in  std_logic_vector  (ADDR_WIDTH-1  downto  0);
    push    :in std_logic;
    CRTL      :out std_logic_vector(1  downto  0);
    CR      :out  std_logic_vector(2  downto 0);
    start     :out  std_logic;
    write_enabler:  out  std_logic;
    reset :out  std_logic;
    increment    :out  std_logic
    );
end component  ;
component  Counter_BA is
generic  (
    ADDR_WIDTH : natural  := 9
  );
Port  (
    RSTn            :in std_logic;
    clk            :in std_logic;
    reset           :in std_logic;
    address         :out  std_logic_vector  (ADDR_WIDTH-1  downto 0);
    increment       :in std_logic
      );
end component  ;
```

```
component  PUF_BA_REVERSE  is
   Port (  clk,rst: in std_logic;
        CRTL : in std_logic_vector  (2 downto  0);

       new_data  : out std_logic;
        rx: in std_logic;
         DATA_OUT : out std_logic_vector(17  downto  0)   );
end component;
signal push1: std_logic;
signal RSTn: std_logic;
signal DATA_D  : STD_LOGIC_VECTOR (17 downto 0);
signal CRTL1  : STD_LOGIC_VECTOR (1 downto  0);
signal CRTL2  : STD_LOGIC_VECTOR (2 downto  0);
signal fsm_rx_1: std_logic;
signal address1  : std_logic_vector  (ADDR_WIDTH-1 downto  0);
signal increment1: std_logic;
signal ready1: std_logic;
signal start1: std_logic;
signal write_enable1  : std_logic;
signal data_out1  :std_logic_vector  (DATA_WIDTH-1 downto  0);
signal busy1: std_logic;
signal resetr1: std_logic;
begin
RSTn<=not RST;
ready1<= not busy1
PUFBA:PUF_BA    generic map( DATA_WIDTH =>18,
                ADDR_WIDTH => 9)
         port map (     clk => clk,
                     rst =>  RST,
CRTL=>CRTL1,
                     new_data  => start1,
                     busy  => busy1,
```

```
TX_IN => data_out1,
                    tx => tx
  );

counterneu:counter_neu   generic map (   top => 250000
            )

         port map (     RSTn => RSTn,
          clk => clk,
                    push_button_in => push_button_in,
                    push_button_out => push1
                    );
BRAMBA:BRAM_BA generic map (  DATA_WIDTH => 18
                     )
       port map (  clk => clk,
                 write_enable => write_enable1,
                 data_out => data_out1,
                 data_in => DATA_D,
                 address => address1
                 );

 FSMBA:FSM_BA    generic map (
                       ADDR_WIDTH => 9  )
           port map (   clk => clk,
                    address => address1,
                    CRTL => CRTL1,
                    CR=>  CRTL2 ,
                   RSTn => RSTn,
                    ready => ready1,
                    push => push1,
                    start => start1,
                    fsm_rx => fsm_rx_1,
```

```
write_enabler=>write_enable1,
                    increment => increment1,
                    reset =>resetr1
                 );

 COUNTERBA:Counter_BA generic map (  ADDR_WIDTH => 9
                     )
           port map (   clk => clk,
                    reset => resetr1, --synchron—Aendern
address => address1,
                    increment => increment1,
                    RSTn => RSTn --Asynchron
                 );

PUF_BAREVERSE: PUF_BA_REVERSE  --generic map (
          -- DATA_WIDTH => 18,
          -- ADDR_WIDTH => 9

          --            )
 port map         (
               clk => clk,
               DATA_OUT => DATA_D,
               new_data => fsm_rx_1,
               CRTL => CRTL2,
               rst => RST,
rx => rx );
end Behavioral;
```

A.1.10 PUF_BA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.All;

entity PUF_BA is
generic (
   DATA_WIDTH : natural := 18;
   ADDR_WIDTH : natural := 9
 );
   Port (  clk,rst: in std_logic;
        TX_IN :   in std_logic_vector (DATA_WIDTH-1 downto 0);
        busy      : out std_logic;
        new_data   : in  std_logic;
        CRTL :       in std_logic_vector(1 downto 0);
          tx: out std_logic
          );
end PUF_BA;
architecture Behavioral of PUF_BA is
component MUX_BA is
generic (
   DATA_WIDTH : natural := 18 );
   port(
      TX_IN      :in std_logic_vector (DATA_WIDTH-1 downto 0);
      TX_OUT     :out std_logic_vector (7 downto 0);
      CRTL       :in std_logic_vector(1 downto 0));
end component ;
component serial_tx is
   generic (
      CLK_PER_BIT : natural := 1042;
      CTR_SIZE    : natural := 11 );
```

```
port (
      clk        : in  std_logic;
      rst        : in  std_logic;
      tx         : out std_logic;
      tx_block    : in  std_logic;
      busy        : out std_logic;
      new_data    : in  std_logic;
      data        : in  std_logic_vector(7  downto 0)
   );
end component;

signal data_in: std_logic_vector(17  downto 0);
signal tx1: std_logic;
signal tx_block1: std_logic;
signal data_1: std_logic_vector(7  downto 0);

 begin
tx<=tx1;
MUXBA:MUX_BA generic map (  DATA_WIDTH => 18
                        )
        port map( TX_IN => TX_IN,
                TX_OUT => data_1,
                CRTL => CRTL
                );


serialtx: serial_tx generic map (  CLK_PER_BIT  =>  1042,
                        CTR_SIZE   =>  11
```

```
                )
        port map (    clk => clk,
                rst => rst,
                tx => tx1,
                tx_block => '0',
                new_data => new_data,
                data => data_1,
                busy => busy
                );
end Behavioral;
```

A.1.11 PUF_BA_REVERSE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.All;

entity PUF_BA_REVERSE is
generic (
   DATA_WIDTH : natural := 18;
   ADDR_WIDTH : natural := 9

  );
   Port (  clk,rst: in std_logic;
        CRTL :  in std_logic_vector (2 downto 0);

       new_data : out std_logic;
        DATA_OUT : out std_logic_vector(17  downto 0);
        rx: in std_logic

          );
end PUF_BA_REVERSE;
architecture Behavioral of PUF_BA_REVERSE is
component  Register_RX is
     port  (
        clk     : in  std_logic;
                CRTL    : in std_logic_vector(2  downto 0);
        rst     : in  std_logic;
        DATA_IN    : in std_logic_vector(7  downto 0);
        DATA_OUT  : out std_logic_vector(17  downto 0)
        );
end component;
```

```
component  serial_rx is
generic  (
    CLK_PER_BIT : natural  := 1042;
    CTR_SIZE  : natural  := 11 );
  port  (
  clk     : in  std_logic;
    rst     : in  std_logic;
     rx      : in  std_logic;
    new_data   : out std_logic;
    data     : out std_logic_vector(7  downto 0)
  );
    end component;
signal DATA_R: STD_LOGIC_VECTOR  (7 downto 0);
 begin
Registerrx:  Register_RX
        port  map (      clk => clk,
                      rst => rst,
                       DATA_IN=>DATA_R,
                       CRTL => CRTL,
                       DATA_OUT=>DATA_OUT    );
Serialrx:serial_rx     generic  map (
                CLK_PER_BIT => 1042,
             CTR_SIZE  => 11
                       )
                          port  map (      clk => clk,
                       rst => rst,
                         rx=> rx,
                         data=>DATA_R,
                         new_data  => new_data  );
                         end;
```

A.1.12PUF_BAT.ccf

```
Pin_in "clk" Loc = "IO_SB_A8" | SCHMITT_TRIGGER=true;
Pin_in "push_button_in" Loc = "IO_NB_A0"; # BTN2 PMOD A
Pin_out "tx" Loc = "IO_NB_A5"; # tx
Pin_in "rx" Loc = "IO_NB_A6"; # rx
Pin_in "RSTP" Loc = "IO_EB_B0"; #SW3
```

A.2 Code (Qt Creator)

A.2.1 Header-Dateien

- **Hydra_ftdi.h**

```
#ifndef HYDRA_FTDI_H_
#define HYDRA_FTDI_H_

#include <ftd2xx.h>
#include <WinTypes.h>

int init_ftdi(const  char *target_serial);

void ftdi_set_gpio(int  word);

#endif
```

- **Lxi_con.h**

```
/*
 * lxi_con.h
 *
 *  Created on: Feb 16, 2022
 *      Author: awalsemann
 */

#ifndef LXI_CON_H_
#define LXI_CON_H_

#include <lxi.h>

#define DEFAULT_TIMEOUT 100

typedef struct {
        int handler;
        char ipv4[16];
        int timeout;
        int port;
        lxi_protocol_t protocol;

} lxi_connection_t;


int lxi_con_init(lxi_connection_t  * con);
int lxi_con_deinit(lxi_connection_t  * con);
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_hmc8043.h

```
/*
 * scpi_hmc8043.h
 *
 *  Created on: Feb 16, 2022
 *      Author: awalsemann
 */

#ifndef SCPI_HMC8043_H_
#define SCPI_HMC8043_H_

#include "lxi_con.h"

#define HMC8043_ERROR -1
#define HMC8043_INVALID_PARAMETER -2
#define HMC8043_LXI_ERROR -3
#define HMC8043_COM_ERROR -4
#define HMC8043_CHANNEL_SELECT_ERROR -5
#define HMC8043_OVP_ERROR -6


typedef struct {
   int channel;
   double voltage_limit;
   double current_limit;
   lxi_connection_t * lxi_connection;
} scpi_hmc8043_t;

int init_hmc8043(scpi_hmc8043_t* dev, double ovp_limit);

int hmc8043_set_current_setpoint(scpi_hmc8043_t* dev, double setpoint);
int hmc8043_set_voltage_setpoint(scpi_hmc8043_t* dev, double setpoint);

int hmc8043_get_current_setpoint(scpi_hmc8043_t* dev, double* value);
int hmc8043_get_voltage_setpoint(scpi_hmc8043_t* dev, double* value);

int hmc8043_measure_current(scpi_hmc8043_t* dev, double* value);
int hmc8043_measure_voltage(scpi_hmc8043_t* dev, double* value);

int hmc8043_get_output_state(scpi_hmc8043_t* dev, int* state);
int hmc8043_set_output_state(scpi_hmc8043_t* dev, int state, int enable_master);

int hmc8043_get_master_state(scpi_hmc8043_t* dev, int* state);
int hmc8043_set_master_state(scpi_hmc8043_t* dev, int state);


#endif /* SCPI_HMC8043_H_ */
```

- **Thermostream.h**

```
#ifndef THERMOSTREAM_H_
#define THERMOSTREAM_H_
#define TERMINAL_ERROR -1
#define TERMINAL_INVALID_PARAMETER -2
#define TERMINAL_SOCKET_ERROR -2
#define TERMINAL_INVALID_HOST -3
#define TERMINAL_INVALID_TYPE -4
#define THERMOSTRAM_SEND_ERROR -5
#define THERMOSTRAM_INVALID_MESSAGE -6
#define THERMOSTRAM_INVALID_PASSWORD -7
#define THERMOSTRAM_INVALID_PARAMETER -8
#define THERMOSTRAM_TEMPERATURE_EXCEEDED -9
#define THERMOSTRAM_MAX_TEMPERATURE 120.0
#define THERMOSTRAM_MIN_TEMPERATURE -80.0
#define THERMOSTRAM_ECHO_BUFFER_SIZE 512
#define THERMOSTRAM_PASSWORD_MAX_SIZE 64
typedef struct {
char ipv4[16];
int timeout;
int port;
int socket_handler;
char password[THERMOSTRAM_PASSWORD_MAX_SIZE];
} thermostream_connection_t;
int thermostream_init(thermostream_connection_t * con);
int thermostream_deinit(thermostream_connection_t * con);

int thermostream_read_temperature(thermostream_connection_t * con, float * air_value, float *
dut_value);
// Setpoint 0-2 (0=hot, 1=ambient, 2=cold)
int thermostream_set_setpoint(thermostream_connection_t * con, int setpoint, float value);
int thermostream_get_current_setpoint(thermostream_connection_t * con, float * value);
int thermostream_select_setpoint(thermostream_connection_t * con, int setpoint);
```

```
int thermostream_set_ramp(thermostream_connection_t * con, int setpoint, float value);
int thermostream_get_ramp(thermostream_connection_t * con, int setpoint, float * value);
int thermostream_set_flowtarget(thermostream_connection_t * con, float value);
int thermostream_get_flowtarget(thermostream_connection_t * con, float * value);
int thermostream_read_flow(thermostream_connection_t * con, float * value);
int thermostream_enable_trickle(thermostream_connection_t * con);
int thermostream_disable_trickle(thermostream_connection_t * con);
int thermostream_check_trickle(thermostream_connection_t * con, int * value);
int thermostream_enable_flow(thermostream_connection_t * con);
int thermostream_disable_flow(thermostream_connection_t * con);
int thermostream_check_flow(thermostream_connection_t * con, int * value);
int thermostream_raise_head(thermostream_connection_t * con);
int thermostream_lower_head(thermostream_connection_t * con);
int thermostream_check_head(thermostream_connection_t * con, int * value);
int thermostream_lock_head(thermostream_connection_t * con);
int thermostream_unlock_head(thermostream_connection_t * con);
int thermostream_enable_chiller(thermostream_connection_t * con);
int thermostream_disable_chiller(thermostream_connection_t * con);
int thermostream_check_chiller(thermostream_connection_t * con, int * value);
#endif /* THERMOSTREAM_H_ */
```

- **arguments.h**

```
#ifndef ARGUMENTS_H_
#define ARGUMENTS_H_
#define _VERSION_ "0.1"
int parse_args(int argc, char *argv[]);
#endif /* ARGUMENTS_H_ */
```

- **cargs.h**

```
#pragma once

/**
 * This is a simple alternative cross-platform implementation of getopt, which
 * is used to parse argument strings submitted to the executable (argc and argv
 * which are received in the main function).
 */

#ifndef CAG_LIBRARY_H
```

```
#define CAG_LIBRARY_H

#include <stdbool.h>
#include <stddef.h>
#include <stdio.h>

#if defined(_WIN32) || defined(__CYGWIN__)
#define CAG_EXPORT __declspec(dllexport)
#define CAG_IMPORT __declspec(dllimport)
#elif __GNUC__ >= 4
#define CAG_EXPORT __attribute__((visibility("default")))
#define CAG_IMPORT __attribute__((visibility("default")))
#else
#define CAG_EXPORT
#define CAG_IMPORT
#endif

#if defined(CAG_SHARED)
#if defined(CAG_EXPORTS)
#define CAG_PUBLIC CAG_EXPORT
#else
#define CAG_PUBLIC CAG_IMPORT
#endif
#else
#define CAG_PUBLIC
#endif

#ifdef __cplusplus
extern "C" {
#endif

/**
 * An option is used to describe a flag/argument option submitted when the
 * program is run.
 */
typedef struct cag_option
{
   const char identifier;
   const char *access_letters;
   const char *access_name;
   const char *value_name;
   const char *description;
} cag_option;

/**
 * A context is used to iterate over all options provided. It stores the parsing
 * state.
 */
typedef struct cag_option_context
{
   const struct cag_option *options;
   size_t option_count;
```

```
    int argc;
    char **argv;
    int index;
    int inner_index;
    bool forced_end;
    char identifier;
    char *value;
} cag_option_context;

/**
 * This is just a small macro which calculates the size of an array.
 */
#define CAG_ARRAY_SIZE(arr) (sizeof(arr) / sizeof((arr)[0]))

/**
 * @brief Prints all options to the terminal.
 *
 * This function prints all options to the terminal. This can be used to
 * generate the output for a "--help" option.
 *
 * @param options The options which will be printed.
 * @param option_count The option count which will be printed.
 * @param destination The destination where the output will be printed.
 */
CAG_PUBLIC void cag_option_print(const cag_option *options, size_t option_count,
                                 FILE *destination);

/**
 * @brief Prepare argument options context for parsing.
 *
 * This function prepares the context for iteration and initializes the context
 * with the supplied options and arguments. After the context has been prepared,
 * it can be used to fetch arguments from it.
 *
 * @param context The context which will be initialized.
 * @param options The registered options which are available for the program.
 * @param option_count The amount of options which are available for the
 * program.
 * @param argc The amount of arguments the user supplied in the main function.
 * @param argv A pointer to the arguments of the main function.
 */
CAG_PUBLIC void cag_option_prepare(cag_option_context *context,
                                   const cag_option *options, size_t option_count, int argc, char **argv);

/**
 * @brief Fetches an option from the argument list.
 *
 * This function fetches a single option from the argument list. The context
 * will be moved to that item. Information can be extracted from the context
 * after the item has been fetched.
 * The arguments will be re-ordered, which means that non-option arguments will
 * be moved to the end of the argument list. After all options have been
```

```
 * fetched, all non-option arguments will be positioned after the index of
 * the context.
 *
 * @param context The context from which we will fetch the option.
 * @return Returns true if there was another option or false if the end is
 * reached.
 */
CAG_PUBLIC bool cag_option_fetch(cag_option_context *context);

/**
 * @brief Gets the identifier of the option.
 *
 * This function gets the identifier of the option, which should be unique to
 * this option and can be used to determine what kind of option this is.
 *
 * @param context The context from which the option was fetched.
 * @return Returns the identifier of the option.
 */
CAG_PUBLIC char cag_option_get(const cag_option_context *context);

/**
 * @brief Gets the value from the option.
 *
 * This function gets the value from the option, if any. If the option does not
 * contain a value, this function will return NULL.
 *
 * @param context The context from which the option was fetched.
 * @return Returns a pointer to the value or NULL if there is no value.
 */
CAG_PUBLIC const char *cag_option_get_value(const cag_option_context *context);

/**
 * @brief Gets the current index of the context.
 *
 * This function gets the index within the argv arguments of the context. The
 * context always points to the next item which it will inspect. This is
 * particularly useful to inspect the original argument array, or to get
 * non-option arguments after option fetching has finished.
 *
 * @param context The context from which the option was fetched.
 * @return Returns the current index of the context.
 */
CAG_PUBLIC int cag_option_get_index(const cag_option_context *context);

#ifdef __cplusplus
} // extern "C"
#endif

#endif
```

- **mainwindow.h**

```
#ifndef MAINWINDOW_H
```

```
#define MAINWINDOW_H

#include <QMainWindow>
#include <fcntl.h>
#include <unistd.h>
#include <QTimer>
#include <QPushButton>
#include <iostream>
#include <QThread>
#include "worker.h"
#include <QTimer>

extern "C"
{
#include "../../../Hydra/inc/thermostream.h"
#include "../../../Hydra/inc/hydra_ftdi.h"
#include "../../../Hydra/inc/scpi_hmc8043.h"
#include "thermostream_control.h"
#include "supply_control.h"
#include "cargs.h"
#include "arguments.h"

};

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class Worker;

class MainWindow : public QMainWindow
{
   Q_OBJECT

public:
   MainWindow(QWidget *parent = nullptr);
   ~MainWindow();


private slots:

      void onrunterButtonClicked();
      void onStartButtonClicked();
      void onStopButtonClicked();
      void onhochButtonClicked();
      void updateParameters();


      //void on_edthoch_clicked();

private:
   Ui::MainWindow *ui;
```

```
    void initializeHMC8043();

    double minVoltage;
    double maxVoltage;
    double voltageStep;
    double minTemperature;
    double temperatureStep;
    double maxTemperature;
    double tempStabil;
    double waitTime;
    int repeat;

    QPushButton *startButton;
    QPushButton *stopButton;
    QPushButton *hochButton;
    QPushButton *runterButton;
    QThread *workerThread; // Déplacez cette déclaration avant QTimer*
    QTimer *timer;
    Worker *worker;

signals:
    void updateWorkerParameters(float minTemperature, float temperatureStep, float maxTempera-
ture, double minVoltage, double voltageStep, double maxVoltage, int repeat, double tempStabil,
double waitTime);

};

#endif // MAINWINDOW_H
```

- **supply_control.h**

```
/*
 * supply_control.h
 *
 *  Created on: May 20, 2022
 *      Author: awalsemann
 */

#ifndef SUPPLY_CONTROL_H_
#define SUPPLY_CONTROL_H_

#define IP_ADDR_SUPPLY "192.168.0.150"

#define OVP_LIMIT 5.5
#define MAX_VOLTAGE 5.0
#define MAX_CURRENT 1.0

enum supply_target{
    ch1,
    ch2,
    ch3,
    all
```

```
};

int init_supply_control(int init);
int deinit_supply_control(void);

int enable_dut_supply(enum supply_target target);
int disable_dut_supply(enum supply_target target);

int set_voltage(double voltage, enum supply_target target);
int set_current(double current, enum supply_target target);

int measure(enum supply_target target, double * voltage, double * current);

#endif /* SUPPLY_CONTROL_H_ */
```

- **Thermostream_control.h**

```
/*
 * thermostream_control.h
 *
 *  Created on: May 23, 2022
 *      Author: awalsemann
 */

#ifndef THERMOSTREAM_CONTROL_H_
#define THERMOSTREAM_CONTROL_H_

#include "../../../Hydra/inc/thermostream.h"

#define IP_ADDR_THERMOSTREAM "192.168.0.90"
#define PASSWORD_THERMOSTREAM "X-Stream"

#define THERMOSTREAM_AMB_TEMP 20.0
#define THERMOSTREAM_AMB_RAMP 0.5


int init_thermo_control(void);
int deinit_thermo_control(void);

int thermo_move_head(int position, int lock);
int thermo_trickle_control(int status);
int thermo_flow_control(float value, int status);
int thermo_read_flow(float * value);
int thermo_read_temps(float * air_value, float * dut_value);
int thermo_set_setpoint(float value, float ramp);
int thermo_set_chiller(int status);
int thermo_read_chiller(int * status);

#endif /* THERMOSTREAM_CONTROL_H_ */
```

- **Worker.h**

```
#ifndef WORKER_H
#define WORKER_H

#include <QThread>
#include <QMutex>
#include <QtSerialPort>
#include <QFile>

class Worker : public QThread
{
   Q_OBJECT

public:
   Worker(QObject *parent = nullptr);

   void stop();
   void setMyFile(const QFile &newMyFile);
   void setMyFileName(const QString &newMyFileName);
   void reprogramFPGA();
   void reprogramAndSetVoltage(double voltage);

public slots:
   void setParameters(float minTemperature, float temperatureStep, float maxTemperature, double
minVoltage, double voltageStep, double maxVoltage, int repeat, double tempStabil, double
waitTime);
   bool readSerialData(QSerialPort &serial, int temperature,double voltage, int repetition );
protected:
   void run() override;

private:
   bool running;
   double minVoltage;
   double voltageStep;
   double maxVoltage;
   double minTemperature;
   double temperatureStep;
   double maxTemperature;
   double tempStabil;
   double waitTime;
   int repeat;

   QMutex mutex;

   QFile myFile;
   QString myFileName;
};

#endif // WORKER_H
```

A.2.2 Quelldateien

- Hydra_ftdi.c

```
#include <stdio.h>
#include <assert.h>
#include "../inc/hydra_ftdi.h"

FT_HANDLE ftHandle;

FT_STATUS status;
#define FTCHECK(x, msg) status = x; if (status != FT_OK) {printf("Error retval = %d: %s\n", status,
msg);return -1;}
//#define FTCHECK(x, msg) x

int init_ftdi(const char *target_serial) {
  DWORD numDevs;
  DWORD Flags;
  DWORD ID;
  DWORD Type;
  DWORD LocId;
  char SerialNumber[16];
  char Description[64];

  // We manually set the PID to 0x9999 for devices that are to be used as GPIO bridge
  FT_SetVIDPID(0x0403, 0x9999);

  FTCHECK(FT_CreateDeviceInfoList(&numDevs), "Create device list")
  printf("Found %d FTDI devices\n", numDevs);

  int target_id = -1;
  for (int i = 0; i < numDevs; ++i) {
    FT_HANDLE ftHandleTemp;
    FTCHECK(FT_GetDeviceInfoDetail(i, &Flags, &Type, &ID, &LocId, SerialNumber, Description,
&ftHandleTemp), "")
    printf("Dev %d:\n", i);
    printf("  Flags=0x%x\n", Flags);
    printf("  Type=0x%x\n", Type);
    printf("  ID=0x%x\n", ID);
    printf("  LocId=0x%x\n", LocId);
    printf("  SerialNumber=%s\n", SerialNumber);
    printf("  Description=%s\n", Description);
    printf("  ftHandle=0x%x\n", ftHandleTemp);

    if (!strcmp(SerialNumber, target_serial)) {
      assert(target_id == -1);
      target_id = i;
    }
  }

  if (target_id == -1) {
    printf("Failed to find device with serial number %s\n", target_serial);
    return -2;
  }

  FTCHECK(FT_Open(target_id, &ftHandle), "open")
```

```
  FTCHECK(FT_ResetDevice(ftHandle),  "reset")

  FTCHECK(FT_SetBitMode(ftHandle,  0, 0), "set bit mode 1")
  FTCHECK(FT_SetBitMode(ftHandle,  0xff, 0x02), "set bit mode 2") // MPSSE all pins output

  return 0;
}

void ftdi_set_gpio(int word) {
  char command[3];
  command[0] = 0x80;
  command[1] = word;
  command[2] = 0xff;

  DWORD dwSent;
  FT_Write(ftHandle,  command, 3, &dwSent);
}
```

- **Lxi_con_c**

```
/*
 * lxi_con.c
 *
 *  Created on: Feb 16, 2022
 *      Author: awalsemann
 */
#include <assert.h>
#include <stdlib.h>
#include <lxi.h>
#include "../inc/lxi_con.h"



int lxi_con_init(lxi_connection_t  * con){
		assert(con != NULL);
		assert(con->port >= 0);
		assert(con->timeout >= 0);

		int res = lxi_connect(con->ipv4, con->port, "inst0", con->timeout, con->protocol);
		if(res != LXI_ERROR){
			con->handler = res;
			return 0;
		}else{
			con->handler = -1;
			return -1;
		}
}

int lxi_con_deinit(lxi_connection_t  * con){
		assert(con != NULL);

		if(lxi_disconnect(con->handler)  != LXI_OK){
```

```
		if(lxi_disconnect(con->handler)  != LXI_OK){
			return -1;
		}else{
			con->handler = -1;
			return 0;
		}
	}else{
		con->handler = -1;
		return 0;
	}
}

int lxi_con_send(lxi_connection_t  * con, char * msg, int len){
	assert(con  != NULL);
	assert(con->handler  >= 0);
	assert(msg  != NULL);
	assert(len > 0);

	int ret = lxi_send(con->handler,  msg, len, con->timeout);
	if( ret == LXI_ERROR || ret < len || ret > len+1){
		return -1;
	}else{
		return 0;
	}
}

int lxi_con_receive(lxi_connection_t  * con, char * msg, int  * len, int max_len){
	assert(con  != NULL);
	assert(con->handler  >= 0);
	assert(msg  != NULL);
	assert(len  != NULL);
	assert(max_len  > 0);

	int ret = lxi_receive(con->handler,  msg, max_len, con->timeout);
	if(ret == LXI_ERROR){
		*msg = 0;
		*len = 0;
		return -1;
	}else{
		*len = ret;
		return 0;
	}

}
```

- **scpi_hmc8043.c**

```
/*
 * scpi_hmc8043.c
 *
 *  Created on: Feb 16, 2022
 *      Author: awalsemann
 */
```

```
#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <locale.h>

#include "scpi_hmc8043.h"
#include "lxi_con.h"


int select_channel(scpi_hmc8043_t*  dev){
   assert(dev != NULL);
   assert(dev->lxi_connection != NULL);
   assert(dev->lxi_connection->handler >= 0);
   assert(dev->channel >= 1);
   assert(dev->channel <= 3);

   char msg[25];
   int length;
   int ret;
   length = snprintf(msg, 25, "INSTrument:NSELect %i", dev->channel);
   ret = lxi_con_send(dev->lxi_connection, msg, length);
   if(ret != 0)
      return HMC8043_ERROR;

   return 0;
}

int init_hmc8043(scpi_hmc8043_t*  dev, double ovp_limit){
   char *locale = setlocale(LC_ALL, "C");
   printf("locale : %s\n", locale);

   assert(dev != NULL);
   assert(dev->lxi_connection != NULL);
   assert(dev->lxi_connection->handler >= 0);
   assert(dev->channel >= 1);
   assert(dev->channel <= 3);
   assert(ovp_limit > 0);

   char msg[60];
   int length;
   int ret;

   //select channel
   if(select_channel(dev) != 0)
      return HMC8043_CHANNEL_SELECT_ERROR;
   length = snprintf(msg, 60,"INSTrument:NSELect?");
   ret = lxi_con_send(dev->lxi_connection, msg, length);
   if(ret != 0)
      return HMC8043_LXI_ERROR;
   lxi_con_receive(dev->lxi_connection, msg, &length, 60);
   if(ret != 0)
```

```
   return HMC8043_LXI_ERROR;
if(length != 2)
   return HMC8043_COM_ERROR;
char str_ch[2];
snprintf(str_ch, 2, "%i", dev->channel);
msg[length-1] = 0;
if(strcmp (msg, str_ch) != 0)
   return HMC8043_CHANNEL_SELECT_ERROR;

//set ovp mode
length = snprintf(msg, 60,"SOURce:VOLTage:PROTection:MODE PROTected");
ret = lxi_con_send(dev->lxi_connection, msg, length);
if(ret != 0)
   return HMC8043_LXI_ERROR;
length = snprintf(msg, 60,"SOURce:VOLTage:PROTection:MODE?");
ret = lxi_con_send(dev->lxi_connection, msg, length);
if(ret != 0)
   return HMC8043_LXI_ERROR;
lxi_con_receive(dev->lxi_connection, msg, &length, 60);
if(ret != 0)
   return HMC8043_LXI_ERROR;
if(length != 5)
   return HMC8043_COM_ERROR;
msg[length-1] = 0;
if(strcmp (msg, "PROT") != 0)
   return HMC8043_OVP_ERROR;

//set ovp value
length = snprintf(msg, 60, "SOURce:VOLTage:PROTection:LEVel %2.3f", ovp_limit);
ret = lxi_con_send(dev->lxi_connection, msg, length);
if(ret != 0)
   return HMC8043_LXI_ERROR;
length = snprintf(msg, 60,"SOURce:VOLTage:PROTection:LEVel?");
ret = lxi_con_send(dev->lxi_connection, msg, length);
if(ret != 0)
   return HMC8043_LXI_ERROR;
lxi_con_receive(dev->lxi_connection, msg, &length, 60);
if(ret != 0)
   return HMC8043_LXI_ERROR;
if(length == 60)
   return HMC8043_COM_ERROR;
msg[length-1] = 0;
double x = atof(msg);
if(x != ovp_limit)
   return HMC8043_OVP_ERROR;

//enable ovp
length = snprintf(msg, 60, "SOURce:VOLTage:PROTection:STATe 1");
ret = lxi_con_send(dev->lxi_connection, msg, length);
if(ret != 0)
   return -1;
length = snprintf(msg, 60, "SOURce:VOLTage:PROTection:STATe?");
```

```
    ret = lxi_con_send(dev->lxi_connection,  msg, length);
    if(ret != 0)
        return -1;
    lxi_con_receive(dev->lxi_connection,  msg, &length, 60);
    if(ret != 0)
        return HMC8043_LXI_ERROR;
    if(length != 2)
        return HMC8043_COM_ERROR;
    msg[length-1] = 0;
    if(strcmp (msg, "1") != 0)
        return HMC8043_OVP_ERROR;

    return 0;
}

int hmc8043_set_current_setpoint(scpi_hmc8043_t*  dev, double setpoint){
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(dev->channel >= 1);
    assert(dev->channel <= 3);

    if(setpoint < 0 || setpoint > dev->current_limit)
        return HMC8043_INVALID_PARAMETER;

    if(select_channel(dev) != 0)
        return HMC8043_CHANNEL_SELECT_ERROR;

    char msg[35];
    int length = snprintf(msg,  35, "SOURce:CURRent:IMMediate %1.4f", setpoint);
    int ret = lxi_con_send(dev->lxi_connection,  msg, length);
    if(ret != 0)
        return HMC8043_LXI_ERROR;

    return 0;
}
int hmc8043_set_voltage_setpoint(scpi_hmc8043_t*  dev, double setpoint){
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(dev->channel >= 1);
    assert(dev->channel <= 3);
printf("hallo\n");
    if(setpoint < 0 || setpoint > dev->voltage_limit)
        return HMC8043_INVALID_PARAMETER;

    if(select_channel(dev) != 0)
        return HMC8043_CHANNEL_SELECT_ERROR;

    char msg[35];
    int length = snprintf(msg,  35, "SOURce:VOLTage:IMMediate %2.3f", setpoint);
    printf("%s\n",  msg);
```

```
    int ret = lxi_con_send(dev->lxi_connection,  msg, length);
    if(ret != 0)
        return HMC8043_LXI_ERROR;
    if(length == 60)
        return HMC8043_COM_ERROR;

    return 0;
}

int hmc8043_get_current_setpoint(scpi_hmc8043_t*  dev, double * value){
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(dev->channel >= 1);
    assert(dev->channel <= 3);
    assert(value != NULL);

    if(select_channel(dev) != 0)
        return HMC8043_CHANNEL_SELECT_ERROR;

    char msg[30];
    int length = snprintf(msg,  30, "SOURce:CURRent?");
    int ret = lxi_con_send(dev->lxi_connection,  msg, length);
    if(ret != 0)
        return HMC8043_LXI_ERROR;

    lxi_con_receive(dev->lxi_connection,  msg, &length, 30);
    if(ret != 0)
        return HMC8043_LXI_ERROR;
    if(length == 0 || length == 30)
        return HMC8043_COM_ERROR;
    double x = atof(msg);
    *value = x;

    return 0;
}
int hmc8043_get_voltage_setpoint(scpi_hmc8043_t*  dev, double * value){
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(dev->channel >= 1);
    assert(dev->channel <= 3);
    assert(value != NULL);

    if(select_channel(dev) != 0)
        return HMC8043_CHANNEL_SELECT_ERROR;

    char msg[30];
    int length = snprintf(msg,  30, "SOURce:VOLTage?");
    int ret = lxi_con_send(dev->lxi_connection,  msg, length);
    if(ret != 0)
        return HMC8043_LXI_ERROR;
```

```
    lxi_con_receive(dev->lxi_connection, msg, &length, 30);
    if(ret != 0)
        return HMC8043_LXI_ERROR;
    if(length == 0 || length == 30)
        return HMC8043_COM_ERROR;
    double x = atof(msg);
    *value = x;

    return 0;
}

int hmc8043_measure_current(scpi_hmc8043_t* dev, double * value){
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(dev->channel >= 1);
    assert(dev->channel <= 3);
    assert(value != NULL);

    if(select_channel(dev) != 0)
        return HMC8043_CHANNEL_SELECT_ERROR;

    char msg[30];
    int length = snprintf(msg, 30, "MEASure:CURRent?");
    int ret = lxi_con_send(dev->lxi_connection, msg, length);
    if(ret != 0)
        return HMC8043_LXI_ERROR;

    lxi_con_receive(dev->lxi_connection, msg, &length, 30);
    if(ret != 0)
        return HMC8043_LXI_ERROR;
    if(length == 0 || length == 30)
        return HMC8043_COM_ERROR;
    double x = atof(msg);
    *value = x;

    return 0;
}

int hmc8043_measure_voltage(scpi_hmc8043_t* dev, double * value){
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(dev->channel >= 1);
    assert(dev->channel <= 3);
    assert(value != NULL);

    if(select_channel(dev) != 0)
        return HMC8043_CHANNEL_SELECT_ERROR;

    char msg[30];
```

```
    int length = snprintf(msg,  30, "MEASure:VOLTage?");
    int ret = lxi_con_send(dev->lxi_connection,  msg, length);
    if(ret != 0)
        return HMC8043_LXI_ERROR;

    lxi_con_receive(dev->lxi_connection,  msg, &length, 30);
    if(ret != 0)
        return HMC8043_LXI_ERROR;
    if(length == 0 || length == 30)
        return HMC8043_COM_ERROR;
    double x = atof(msg);
    *value = x;

    return 0;
}

int hmc8043_get_output_state(scpi_hmc8043_t*  dev, int* state){
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(dev->channel >= 1);
    assert(dev->channel <= 3);
    assert(state != NULL);

    if(select_channel(dev) != 0)
        return HMC8043_CHANNEL_SELECT_ERROR;

    char msg[20];
    int length = snprintf(msg,  20, "OUTPut:CHANnel?");
    int ret = lxi_con_send(dev->lxi_connection,  msg, length);
    if(ret != 0)
        return HMC8043_LXI_ERROR;

    lxi_con_receive(dev->lxi_connection,  msg, &length, 20);
    if(ret != 0)
        return HMC8043_LXI_ERROR;
    if(length != 2)
        return HMC8043_COM_ERROR;
    msg[length-1] = 0;
    int x = atoi(msg);
    *state = x;

    return 0;
}
int hmc8043_set_output_state(scpi_hmc8043_t*  dev, int state, int enable_master){
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(dev->channel >= 1);
    assert(dev->channel <= 3);
    assert(state >= 0);
    assert(state <= 1);
```

```
    assert(enable_master >= 0);
    assert(enable_master <= 1);

    if(select_channel(dev) != 0)
        return HMC8043_CHANNEL_SELECT_ERROR;

    char msg[20];
    int length;
    if(enable_master == 1){
        length = snprintf(msg, 20, "OUTPut:State %i", state);
    }else{
        length = snprintf(msg, 20, "OUTPut:CHANnel %i", state);
    }
    int ret = lxi_con_send(dev->lxi_connection, msg, length);
    if(ret != 0)
        return HMC8043_LXI_ERROR;

    return 0;
}

int hmc8043_get_master_state(scpi_hmc8043_t* dev, int* state){
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(state != NULL);

    char msg[20];
    int length = snprintf(msg, 20, "OUTPut:MASTer?");
    int ret = lxi_con_send(dev->lxi_connection, msg, length);
    if(ret != 0)
        return HMC8043_LXI_ERROR;

    lxi_con_receive(dev->lxi_connection, msg, &length, 20);
    if(ret != 0)
        return HMC8043_LXI_ERROR;
    if(length != 2)
        return HMC8043_COM_ERROR;
    msg[length-1] = 0;
    double x = atoi(msg);
    *state = x;

    return 0;
}

int hmc8043_set_master_state(scpi_hmc8043_t* dev, int state){
    assert(dev != NULL);
    assert(dev->lxi_connection != NULL);
    assert(dev->lxi_connection->handler >= 0);
    assert(state >= 0);
    assert(state <= 1);

    char msg[20];
```

```
   int length = snprintf(msg, 20, "OUTPut:MASTer %i", state);
   int ret = lxi_con_send(dev->lxi_connection, msg, length);
   if(ret != 0)
      return HMC8043_LXI_ERROR;

   return 0;
}
```

- **thermostream.c**

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "../inc/thermostream.h"


int thermostream_recv_message(int fd, char * send_msg, int send_msg_len, char * recv_msg, int *
recv_msg_len, int max_recv_len){
assert(fd >= 0);
assert(send_msg != NULL);
assert(send_msg_len > 0);
assert(recv_msg != NULL);
assert(max_recv_len > 0);

//send message
if( send(fd, send_msg , send_msg_len , 0) < send_msg_len)
return TERMINAL_SOCKET_ERROR;
//check if echo is received correctly
int ret = recv(fd, recv_msg, send_msg_len, 0);
if (ret < 0)
return TERMINAL_SOCKET_ERROR;
if (ret > 0) {
recv_msg[ret] = 0;
int ret_str = strcmp(recv_msg, send_msg);
if(ret_str != 0)
return THERMOSTRAM_SEND_ERROR;
}

//read response from thermostream
ret = recv(fd, recv_msg, max_recv_len, 0);
if (ret < 0)
return TERMINAL_SOCKET_ERROR;
if (ret == 0)
return THERMOSTRAM_INVALID_MESSAGE;
if(recv_msg_len != NULL)
```

```
*recv_msg_len = ret;
recv_msg[ret] = 0;

return 0;
}

int thermostream_send_message(int  fd, char * send_msg, int send_msg_len){
assert(fd >= 0);
assert(send_msg  != NULL);
assert(send_msg_len  > 0);
assert(send_msg_len  < THERMOSTRAM_ECHO_BUFFER_SIZE);

char buffer[THERMOSTRAM_ECHO_BUFFER_SIZE];
//send message
if( send(fd, send_msg , send_msg_len , 0) < send_msg_len)
return TERMINAL_SOCKET_ERROR;
//check if echo is received correctly
int ret = recv(fd, buffer, sizeof(buffer), 0);
if (ret < 0)
return TERMINAL_SOCKET_ERROR;
if (ret > 0) {
buffer[ret] = 0;
int ret_str = strcmp(buffer, send_msg);
if(ret_str != 0)
return THERMOSTRAM_SEND_ERROR;
}

return 0;
}

int thermostream_init(thermostream_connection_t  * con){
assert(con != NULL);

int sock;
// create socket
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)
return TERMINAL_SOCKET_ERROR;

   struct sockaddr_in addr;
memset(&addr, 0, sizeof(addr));
   addr.sin_addr.s_addr  = inet_addr(con->ipv4);
addr.sin_family = AF_INET;
addr.sin_port = htons( con->port );

//connect
if (connect(sock, (struct sockaddr *)&addr, sizeof(addr)) == -1) {
close(sock);
return TERMINAL_SOCKET_ERROR;
}

//set timeout
struct timeval tv;
```

```
tv.tv_sec = con->timeout/1000;
tv.tv_usec = (con->timeout-tv.tv_sec*1000)*1000;
setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof tv);

//set socket handler in struct
con->socket_handler = sock;

char buffer[512];
int ret;
//check login message
ret = recv(con->socket_handler, buffer, sizeof(buffer), 0);
if (ret < 0)
return TERMINAL_SOCKET_ERROR;
if (ret > 0) {
char* ret_str = strstr(buffer, "Welcome to the X-Stream Server");
if (ret_str == NULL)
return THERMOSTRAM_INVALID_MESSAGE;
}

//send password login
int len = snprintf (buffer, 512, "LGIN %s\r\n", con->password);
int recv_len;
ret = thermostream_recv_message(con->socket_handler, buffer, len, buffer, &recv_len,
sizeof(buffer));
if (ret < 0)
return ret;

char *ret_str = strstr(buffer, "Password incorrect.");
if (ret_str != NULL)
return THERMOSTRAM_INVALID_PASSWORD;
else{
char * ret_str = strstr(buffer, "Password correct. Controller now is \"Network\"");
if (ret_str == NULL)
return THERMOSTRAM_INVALID_MESSAGE;
}

return 0;
}


int thermostream_deinit(thermostream_connection_t * con){
assert(con != NULL);
assert(con->socket_handler >= 0);

char * logout_msg = "LO\r\n";
thermostream_send_message(con->socket_handler, logout_msg, strlen(logout_msg));

close(con->socket_handler);

return 0;
}
```

```
int thermostream_read_temperature(thermostream_connection_t * con, float * air_value, float *
dut_value){
assert(con != NULL);
assert(con->socket_handler >= 0);
assert(air_value != NULL);
assert(dut_value != NULL);

char recv_buffer[64];
int ret;
int recv_len;

//send air temperature string to thermostream
char * air_temp_msg = "TMPA?\r\n";
ret = thermostream_recv_message(con->socket_handler, air_temp_msg, strlen(air_temp_msg),
recv_buffer, &recv_len, sizeof(recv_buffer));
if(ret < 0)
return ret;
//check air temperature response
float tmp_val = strtof(recv_buffer, NULL);
if(tmp_val < -100.0 || tmp_val > 300)
return THERMOSTRAM_TEMPERATURE_EXCEEDED;
*air_value = tmp_val;

//send dut temperature string to thermostream
char * dut_temp_msg = "TMPD?\r\n";
ret = thermostream_recv_message(con->socket_handler, dut_temp_msg, strlen(dut_temp_msg),
recv_buffer, &recv_len, sizeof(recv_buffer));
if(ret < 0)
return ret;
//check temperature response
tmp_val = strtof(recv_buffer, NULL);
if(tmp_val < -100.0 || tmp_val > 300)
return THERMOSTRAM_TEMPERATURE_EXCEEDED;
*dut_value = tmp_val;

return 0;
}


int thermostream_set_setpoint(thermostream_connection_t * con, int setpoint, float value){
assert(con != NULL);
assert(con->socket_handler >= 0);

if(setpoint < 0 || setpoint > 2)
return THERMOSTRAM_INVALID_PARAMETER;

if(value < THERMOSTRAM_MIN_TEMPERATURE || value > THERMOSTRAM_MAX_TEMPERATURE)
return THERMOSTRAM_TEMPERATURE_EXCEEDED;
//####### Bug in Thermostream? Edit mode sets active setpoint to 15°C?! #######
//####### Workaround
//read current setpoint
char * fix_get_setn = "SETN?\r\n";
```

```
char fix_recv_buffer[64];
int fix_ret = thermostream_recv_message(con->socket_handler,  fix_get_setn, strlen(fix_get_setn),
fix_recv_buffer, NULL, sizeof(fix_recv_buffer));
if(fix_ret < 0)
return fix_ret;
int fix_setn = strtol(fix_recv_buffer,  NULL, 10);
if(fix_setn < 0)
return THERMOSTRAM_INVALID_MESSAGE;
//####### Workaround

char * edit_start_msg = "EDIT 1\r\n";
int ret = thermostream_send_message(con->socket_handler,  edit_start_msg,
strlen(edit_start_msg));
if(ret < 0)
return ret;
//select desired setpoint
char select_setn[16];
int len =  snprintf (select_setn, 16, "SETN %i\r\n", setpoint);
ret = thermostream_send_message(con->socket_handler,  select_setn, len);
if(ret < 0)
return ret;
//edit desired setpoint
char set_setp[20];
len =  snprintf (set_setp, 20, "SETP %3.2f\r\n", value);
ret = thermostream_send_message(con->socket_handler,  set_setp, len);
if(ret < 0)
return ret;

//####### Bug in Thermostream? Edit mode sets active setpoint to 15°C?! #######
//####### Workaround
//write back current setpoint
char fix_set_setn[20];
int fix_len =  snprintf (fix_set_setn, 20, "SETN %i\r\n", fix_setn);
fix_ret = thermostream_send_message(con->socket_handler,  fix_set_setn, fix_len);
if(fix_ret < 0)
return fix_ret;
//####### Workaround

//leave setpoint edit mode
char * edit_stop_msg = "EDIT 0\r\n";
ret = thermostream_send_message(con->socket_handler,  edit_stop_msg, strlen(edit_stop_msg));
if(ret < 0)
return ret;

return 0;
}

int thermostream_get_current_setpoint(thermostream_connection_t  * con, float * value){
assert(con != NULL);
assert(con->socket_handler >= 0);
assert(value != NULL);
```

```
//read current setpoint
char * get_setp = "SETP?\r\n";
char recv_buffer[64];
int ret = thermostream_recv_message(con->socket_handler, get_setp, strlen(get_setp), recv_buffer,
NULL, sizeof(recv_buffer));
if(ret < 0)
return ret;
float tmp_val = strtof(recv_buffer, NULL);
if(tmp_val < -100.0 || tmp_val > 300)
return THERMOSTRAM_TEMPERATURE_EXCEEDED;
*value = tmp_val;

return 0;
}

int thermostream_select_setpoint(thermostream_connection_t * con, int setpoint){
assert(con != NULL);
assert(con->socket_handler >= 0);

if(setpoint < 0 || setpoint > 2)
return THERMOSTRAM_INVALID_PARAMETER;

//select desired setpoint
char select_setn[16];
int len = snprintf (select_setn, 16, "SETN %i\r\n", setpoint);
int ret = thermostream_send_message(con->socket_handler, select_setn, len);

return ret;
}

int thermostream_set_ramp(thermostream_connection_t * con, int setpoint, float value){
assert(con != NULL);
assert(con->socket_handler >= 0);

if(setpoint < 0 || setpoint > 2)
return THERMOSTRAM_INVALID_PARAMETER;

if(value < 0 || value > 9999)
return THERMOSTRAM_INVALID_PARAMETER;
//####### Bug in Thermostream? Edit mode sets active setpoint to 15°C?! #######
//####### Workaround
//read current setpoint
char * fix_get_setn = "SETN?\r\n";
char fix_recv_buffer[64];
int fix_ret = thermostream_recv_message(con->socket_handler, fix_get_setn, strlen(fix_get_setn),
fix_recv_buffer, NULL, sizeof(fix_recv_buffer));
if(fix_ret < 0)
return fix_ret;
int fix_setn = strtol(fix_recv_buffer, NULL, 10);
if(fix_setn < 0)
return THERMOSTRAM_INVALID_MESSAGE;
//####### Workaround
```

```
//enter setpoint edit mode
char * edit_start_msg = "EDIT 1\r\n";
int ret = thermostream_send_message(con->socket_handler,  edit_start_msg,
strlen(edit_start_msg));
if(ret < 0)
return ret;
//select desired setpoint
char select_setn[16];
int len =  snprintf (select_setn, 16, "SETN %i\r\n", setpoint);
ret = thermostream_send_message(con->socket_handler,  select_setn, len);
if(ret < 0)
return ret;
//edit desired setpoint
char set_setp[20];
len =  snprintf (set_setp, 20, "RAMP %4.1f\r\n", value);
ret = thermostream_send_message(con->socket_handler,  set_setp, len);
if(ret < 0)
return ret;

//####### Bug in Thermostream? Edit mode sets active setpoint to 15°C?! #######
//####### Workaround
//write back current setpoint
char fix_set_setn[20];
int fix_len =  snprintf (fix_set_setn, 20, "SETN %i\r\n", fix_setn);
fix_ret = thermostream_send_message(con->socket_handler,  fix_set_setn, fix_len);
if(fix_ret < 0)
return fix_ret;
//####### Workaround

//leave setpoint edit mode
char * edit_stop_msg = "EDIT 0\r\n";
ret = thermostream_send_message(con->socket_handler,  edit_stop_msg, strlen(edit_stop_msg));
if(ret < 0)
return ret;

return 0;
}
int thermostream_get_ramp(thermostream_connection_t  * con, int setpoint, float * value){
assert(con != NULL);
assert(con->socket_handler >= 0);
assert(value != NULL);

if(setpoint < 0 || setpoint > 2)
return THERMOSTRAM_INVALID_PARAMETER;
//####### Bug in Thermostream? Edit mode sets active setpoint to 15°C?! #######
//####### Workaround
//read current setpoint
char * fix_get_setn = "SETN?\r\n";
char fix_recv_buffer[64];
int fix_ret = thermostream_recv_message(con->socket_handler,  fix_get_setn, strlen(fix_get_setn),
fix_recv_buffer, NULL, sizeof(fix_recv_buffer));
```

```
if(fix_ret < 0)
return fix_ret;
int fix_setn = strtol(fix_recv_buffer, NULL, 10);
if(fix_setn < 0)
return THERMOSTRAM_INVALID_MESSAGE;
//####### Workaround

//enter edit mode
char * edit_start_msg = "EDIT 1\r\n";
int ret = thermostream_send_message(con->socket_handler, edit_start_msg,
strlen(edit_start_msg));
if(ret < 0)
return ret;
//select desired setpoint
char select_setn[16];
int len = snprintf (select_setn, 16, "SETN %i\r\n", setpoint);
ret = thermostream_send_message(con->socket_handler, select_setn, len);
if(ret < 0)
return ret;
//read current setpoint ramp
char * get_setp = "RAMP?\r\n";
char recv_buffer[64];
ret = thermostream_recv_message(con->socket_handler, get_setp, strlen(get_setp), recv_buffer,
NULL, sizeof(recv_buffer));
if(ret < 0)
return ret;
float tmp_val = strtof(recv_buffer, NULL);
if(tmp_val < 0 || tmp_val > 9999)
return THERMOSTRAM_INVALID_MESSAGE;
*value = tmp_val;

//####### Bug in Thermostream? Edit mode sets active setpoint to 15°C?! #######
//####### Workaround
//write back current setpoint
char fix_set_setn[20];
int fix_len = snprintf (fix_set_setn, 20, "SETN %i\r\n", fix_setn);
fix_ret = thermostream_send_message(con->socket_handler, fix_set_setn, fix_len);
if(fix_ret < 0)
return fix_ret;
//####### Workaround

//leave setpoint edit mode
char * edit_stop_msg = "EDIT 0\r\n";
ret = thermostream_send_message(con->socket_handler, edit_stop_msg, strlen(edit_stop_msg));
if(ret < 0)
return ret;

return 0;
}

int thermostream_set_flowtarget(thermostream_connection_t * con, float value){
assert(con != NULL);
```

```
assert(con->socket_handler >= 0);

if(value < 4 || value > 18)
return THERMOSTRAM_INVALID_PARAMETER;

//set current flowrate target
char set_flse[20];
int len = snprintf (set_flse, 20, "FLSE %2.1f\r\n", value);
int ret = thermostream_send_message(con->socket_handler, set_flse, len);

return ret;
}
int thermostream_get_flowtarget(thermostream_connection_t * con, float * value){
assert(con != NULL);
assert(con->socket_handler >= 0);
assert(value != NULL);

//read current flowrate target
char * get_flse = "FLSE?\r\n";
char recv_buffer[64];
int ret = thermostream_recv_message(con->socket_handler, get_flse, strlen(get_flse), recv_buffer,
NULL, sizeof(recv_buffer));
if(ret < 0)
return ret;
float tmp_val = strtof(recv_buffer, NULL);
if(tmp_val < 0 || tmp_val > 25)
return THERMOSTRAM_INVALID_MESSAGE;
*value = tmp_val;

return 0;
}
int thermostream_read_flow(thermostream_connection_t * con, float * value){
assert(con != NULL);
assert(con->socket_handler >= 0);
assert(value != NULL);

//read current flowrate
char * get_flwr = "FLWR?\r\n";
char recv_buffer[64];
int ret = thermostream_recv_message(con->socket_handler, get_flwr, strlen(get_flwr), recv_buffer,
NULL, sizeof(recv_buffer));
if(ret < 0)
return ret;
float tmp_val = strtof(recv_buffer, NULL);
if(tmp_val < 0 || tmp_val > 25)
return THERMOSTRAM_INVALID_MESSAGE;
*value = tmp_val;

return 0;
}
```

```
int thermostream_enable_trickle(thermostream_connection_t  * con){
assert(con != NULL);
assert(con->socket_handler >= 0);

//send enable trickle flow message
char * trkl_msg = "TRKL 1\r\n";
int ret = thermostream_send_message(con->socket_handler,  trkl_msg, strlen(trkl_msg));

return ret;
}
int thermostream_disable_trickle(thermostream_connection_t  * con){
assert(con != NULL);
assert(con->socket_handler >= 0);

//send disable trickle flow message
char * trkl_msg = "TRKL 0\r\n";
int ret = thermostream_send_message(con->socket_handler,  trkl_msg, strlen(trkl_msg));

return ret;
}
int thermostream_check_trickle(thermostream_connection_t  * con, int * value){
assert(con != NULL);
assert(con->socket_handler >= 0);
assert(value != NULL);

//read trickle flow status
char * get_trkl = "TRKL?\r\n";
char recv_buffer[64];
int ret = thermostream_recv_message(con->socket_handler,  get_trkl, strlen(get_trkl),  recv_buffer,
NULL, sizeof(recv_buffer));
if(ret < 0)
return ret;
int tmp_val = strtol(recv_buffer,  NULL,10);
if(tmp_val < 0 || tmp_val > 1)
return THERMOSTRAM_INVALID_MESSAGE;
*value = tmp_val;

return 0;
}

int thermostream_enable_flow(thermostream_connection_t  * con){
assert(con != NULL);
assert(con->socket_handler >= 0);

//send start main flow message
char * flow_msg = "FLOW 1\r\n";
int ret = thermostream_send_message(con->socket_handler,  flow_msg, strlen(flow_msg));

return ret;
}
int thermostream_disable_flow(thermostream_connection_t  * con){
assert(con != NULL);
```

```
assert(con->socket_handler >= 0);

//send disable main flow message
char * flow_msg = "FLOW 0\r\n";
int ret = thermostream_send_message(con->socket_handler, flow_msg, strlen(flow_msg));

return ret;
}
int thermostream_check_flow(thermostream_connection_t * con, int * value){
assert(con != NULL);
assert(con->socket_handler >= 0);
assert(value != NULL);

//read main flow status
char * get_flow = "FLOW?\r\n";
char recv_buffer[64];
int ret = thermostream_recv_message(con->socket_handler, get_flow, strlen(get_flow), recv_buffer,
NULL, sizeof(recv_buffer));
if(ret < 0)
return ret;
int tmp_val = strtol(recv_buffer, NULL,10);
if(tmp_val < 0 || tmp_val > 1)
return THERMOSTRAM_INVALID_MESSAGE;
*value = tmp_val;

return 0;
}

int thermostream_raise_head(thermostream_connection_t * con){
assert(con != NULL);
assert(con->socket_handler >= 0);

//send raise head message
char * head_msg = "HEAD 0\r\n";
int ret = thermostream_send_message(con->socket_handler, head_msg, strlen(head_msg));

return ret;
}
int thermostream_lower_head(thermostream_connection_t * con){
assert(con != NULL);
assert(con->socket_handler >= 0);

//send lower head message
char * head_msg = "HEAD 1\r\n";
int ret = thermostream_send_message(con->socket_handler, head_msg, strlen(head_msg));
if(ret < 0)
return ret;
//wait, then directly turn flow back off
struct timespec ts = {0};
ts.tv_sec = 1;
nanosleep(&ts, NULL);
//send disable flow
```

```
ret = thermostream_disable_flow(con);

return ret;
}

int thermostream_check_head(thermostream_connection_t  * con, int * value){
assert(con != NULL);
assert(con->socket_handler >= 0);
assert(value != NULL);

//read main flow status
char * get_head = "HEAD?\r\n";
char recv_buffer[64];
int ret = thermostream_recv_message(con->socket_handler,  get_head, strlen(get_head),
recv_buffer, NULL, sizeof(recv_buffer));
if(ret < 0)
return ret;
int tmp_val = strtol(recv_buffer,  NULL,10);
if(tmp_val < 0 || tmp_val > 1)
return THERMOSTRAM_INVALID_MESSAGE;
*value = tmp_val;

return 0;

}

int thermostream_lock_head(thermostream_connection_t  * con){
assert(con != NULL);
assert(con->socket_handler >= 0);

//send lock head message
char * head_msg = "HDLK 1\r\n";
int ret = thermostream_send_message(con->socket_handler,  head_msg, strlen(head_msg));

return ret;
}
int thermostream_unlock_head(thermostream_connection_t  * con){
assert(con != NULL);
assert(con->socket_handler >= 0);

//send unlock head message
char * head_msg = "HDLK 0\r\n";
int ret = thermostream_send_message(con->socket_handler,  head_msg, strlen(head_msg));

return ret;
}


int thermostream_enable_chiller(thermostream_connection_t  * con){
assert(con != NULL);
assert(con->socket_handler >= 0);
```

```
//send enable chiller message
char * head_msg = "COOL 1\r\n";
int ret = thermostream_send_message(con->socket_handler,  head_msg, strlen(head_msg));

return ret;
}

int thermostream_disable_chiller(thermostream_connection_t  * con){
assert(con != NULL);
assert(con->socket_handler >= 0);

//send disable chiller message
char * head_msg = "COOL 0\r\n";
int ret = thermostream_send_message(con->socket_handler,  head_msg, strlen(head_msg));

return ret;
}

int thermostream_check_chiller(thermostream_connection_t  * con, int * value){
assert(con != NULL);
assert(con->socket_handler >= 0);
assert(value != NULL);

//read chiller status
char * get_head = "COOL?\r\n";
char recv_buffer[64];
int ret = thermostream_recv_message(con->socket_handler,  get_head, strlen(get_head),
recv_buffer, NULL, sizeof(recv_buffer));
if(ret < 0)
return ret;
int tmp_val = strtol(recv_buffer,  NULL,10);
if(tmp_val < 0 || tmp_val > 1)
return THERMOSTRAM_INVALID_MESSAGE;
*value = tmp_val;

return 0;
}
```

- **arguments.c**

```
/*
 * arguments.c
 *
 *  Created on: May 20, 2022
 *      Author: awalsemann
 */

#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <time.h>
#include <unistd.h>
```

```
#include "cargs.h"
#include "arguments.h"
#include "supply_control.h"


static struct cag_option supply_options[] = {
   {.identifier = 'k',
    .access_letters = "k",
    .access_name = "keep-config",
    .value_name = NULL,
    .description = "Keep current equipment configuration, do not re-init"},

   {.identifier = 'e',
    .access_letters = "e",
    .access_name = "enable",
    .value_name = NULL,
    .description = "Enable power supply or channel"},

   {.identifier = 'd',
    .access_letters = "d",
    .access_name = "disable",
    .value_name = NULL,
    .description = "Disable power supply or channel"},

   {.identifier = 'c',
    .access_letters = "c",
    .access_name = "current",
    .value_name = "x.xxx",
    .description = "Output current limit [A]"},

   {.identifier = 'v',
    .access_letters = "v",
    .access_name = "voltage",
    .value_name = "xx.xxx",
    .description = "Output voltage level [V]"},

   {.identifier = 'm',
    .access_letters = "m",
    .access_name = "measure",
    .value_name = NULL,
    .description = "Measure output current and voltage"},

   {.identifier = 'n',
    .access_letters = "n",
    .access_name = "channel",
    .value_name = "1..3,all",
    .description = "Select output channel 1..3 or all"},

   {.identifier = 'h',
    .access_letters = "h",
    .access_name = "help",
    .description = "Shows the command help"}};
```

```
struct supply_config{
   bool enable_flag;
   bool disable_flag;
   bool keep_flag;
   bool channel_flag;
   enum supply_target target;
   bool voltage_flag;
   bool current_flag;
   float voltage_value;
   float current_value;
   bool measure_flag;
};

int execute_supply(struct supply_config * config){
   if(init_supply_control(!config->keep_flag)){
      printf("\nSupply init failed! exiting.\n");
      return -1;
   }

   if(config->enable_flag){
      if(enable_dut_supply(config->target)){
         printf("\nSupply enable failed! exiting.\n");
         return -1;
      }
   }

   if(config->disable_flag){
      if(disable_dut_supply(config->target)){
         printf("\nSupply disabled failed! exiting.\n");
         return -1;
      }
   }

   if(config->voltage_flag){
      if(set_voltage(config->voltage_value, config->target)){
         printf("\nSupply voltage setpoint change failed! exiting.\n");
         return -1;
      }
   }

   if(config->current_flag){
      if(set_current(config->current_value, config->target)){
         printf("\nSupply current setpoint change failed! exiting.\n");
         return -1;
      }
   }

   if(config->measure_flag){
      double m_voltage[3];
      double m_current[3];
```

```
        if(measure(config->target, m_voltage, m_current)){
            printf("\nSupply measurement failed! exiting.\n");
            return -1;
        }
        if(config->target == ch1){
            printf("Channel: 1\tVoltage: %2.3fV\tCurrent: %1.5fA\n", m_voltage[0], m_current[0]);
        }
        if(config->target == ch2){
            printf("Channel: 2\tVoltage: %2.3fV\tCurrent: %1.5fA\n", m_voltage[0], m_current[0]);
        }
        if(config->target == ch3){
            printf("Channel: 3\tVoltage: %2.3fV\tCurrent: %1.5fA\n", m_voltage[0], m_current[0]);
        }
        if(config->target == all){
            printf("Channel: 1\tVoltage: %2.3fV\tCurrent: %1.5fA\n", m_voltage[0], m_current[0]);
            printf("Channel: 2\tVoltage: %2.3fV\tCurrent: %1.5fA\n", m_voltage[1], m_current[1]);
            printf("Channel: 3\tVoltage: %2.3fV\tCurrent: %1.5fA\n", m_voltage[2], m_current[2]);
        }
    }

    if(deinit_supply_control()){
        printf("Closing LXI connections failed!\n");
        return -1;
    }

    return 0;
}

int check_supply(struct supply_config * config){
    if(config->channel_flag == false){
        printf("Missing channel selction! Use -n option\n");
        return -1;
    }
    if(config->enable_flag == true && config->disable_flag == true){
        printf("Invalid combination of supply enable and disable commands!\n");
        return -1;
    }
    if(config->voltage_value < 0.0 || config->current_value < 0.0 ){
        printf("Invalid voltage / current defined!\n");
        return -1;
    }
    return 0;
}

int parse_supply_target(const char * string, enum supply_target * target){
    if (strcmp(string, "all") == 0) {
        *target=all;
        return 0;
    }
    else if (strcmp(string, "1") == 0) {
        *target=ch1;
        return 0;
```

```
    }
    else if (strcmp(string, "2") == 0) {
        *target=ch2;
        return 0;
    }
    else if (strcmp(string, "3") == 0) {
        *target=ch3;
        return 0;
    }
    return -1;
}

int parse_supply(cag_option_context * context){
    char identifier;
    const char *value;
    struct supply_config config = {.enable_flag=false, .disable_flag=false, .keep_flag=false, .chan-
nel_flag=false, .target=ch1, .voltage_flag=false, .current_flag=false, .voltage_value=0.5, .cur-
rent_value=0.05, .measure_flag=false};

    while (cag_option_fetch(context)) {
        identifier = cag_option_get(context);
        switch (identifier) {
        case 'e':
            config.enable_flag = true;
            break;

        case 'd':
            config.disable_flag = true;
            break;

        case 'c':
            value = cag_option_get_value(context);
            if(value != NULL){
                config.current_flag=true;
                config.current_value = strtof(value, NULL);
                if(config.current_value == 0.0){
                    printf("Invalid current value!\n");
                    return EXIT_FAILURE;
                }
            }else{
                printf("Missing current value!\n");
                return EXIT_FAILURE;
            }
            break;

        case 'v':
            value = cag_option_get_value(context);
            if(value != NULL){
                config.voltage_flag=true;
                config.voltage_value = strtof(value, NULL);
                if(config.voltage_value == 0.0){
                    printf("Invalid voltage value!\n");
```

```
            return EXIT_FAILURE;
         }
      }else{
         printf("Missing  voltage value!\n");
         return EXIT_FAILURE;
      }
      break;

   case 'm':
      config.measure_flag = true;
      break;

   case 'n':
      config.channel_flag = true;
      value = cag_option_get_value(context);
      if(value != NULL){
         if(parse_supply_target(value,  &config.target)){
            printf("Invalid  channel selection!\n");
            return EXIT_FAILURE;
         }
      }else{
         printf("Missing  SRAM voltage value!\n");
         return EXIT_FAILURE;
      }
      break;

   case 'k':
      config.keep_flag = true;
      break;


   case 'h':
      printf("\nUsage: supply_control  [optional arguments]...\n");
      cag_option_print(supply_options,  CAG_ARRAY_SIZE(supply_options),  stdout);
      return EXIT_SUCCESS;
      break;

   default:
      printf("Invalid  parameter!\n");
      printf("Usage: supply_control  [optional arguments]...\n");
      cag_option_print(supply_options,  CAG_ARRAY_SIZE(supply_options),  stdout);
      return EXIT_FAILURE;
      break;
   }
}

if(check_supply(&config)){
   return EXIT_FAILURE;
}

if(execute_supply(&config)){
   return EXIT_FAILURE;
```

```
    }

    return  EXIT_SUCCESS;
}



int parse_args(int  argc, char *argv[]){
    cag_option_context context;

    cag_option_prepare(&context,  supply_options,  CAG_ARRAY_SIZE(supply_options),  argc, argv);
    parse_supply(&context);

    return  EXIT_SUCCESS;
}
```

- **cargs.c**

```
#include <assert.h>
#include <memory.h>
#include <stdio.h>
#include <string.h>
#include "cargs.h"

#define CAG_OPTION_PRINT_DISTANCE 4
#define CAG_OPTION_PRINT_MIN_INDENTION 20

static void cag_option_print_value(const  cag_option *option,
                                   size_t *accessor_length, FILE *destination)
{
    if (option->value_name  != NULL) {
        *accessor_length += fprintf(destination,  "=%s", option->value_name);
    }
}

static void cag_option_print_letters(const  cag_option *option, bool *first,
                                     size_t *accessor_length,  FILE *destination)
{
    const char *access_letter;
    access_letter = option->access_letters;
    if (access_letter != NULL) {
        while (*access_letter) {
            if (*first) {
                *accessor_length += fprintf(destination,  "-%c", *access_letter);
                *first = false;
            } else {
                *accessor_length += fprintf(destination,  ", -%c", *access_letter);
            }
            ++access_letter;
        }
    }
}
```

```
static void cag_option_print_name(const cag_option *option, bool *first,
                                  size_t *accessor_length, FILE *destination)
{
   if (option->access_name != NULL) {
      if (*first) {
         *accessor_length += fprintf(destination, "--%s", option->access_name);
      } else {
         *accessor_length += fprintf(destination, ", --%s", option->access_name);
      }
   }
}

static size_t cag_option_get_print_indention(const cag_option *options,
                                             size_t option_count)
{
   size_t option_index, indention, result;
   const cag_option *option;

   result = CAG_OPTION_PRINT_MIN_INDENTION;

   for (option_index = 0; option_index < option_count; ++option_index) {
      indention = CAG_OPTION_PRINT_DISTANCE;
      option = &options[option_index];
      if (option->access_letters != NULL && *option->access_letters) {
         indention += strlen(option->access_letters) * 4 - 2;
         if (option->access_name != NULL) {
            indention += strlen(option->access_name) + 4;
         }
      } else if (option->access_name != NULL) {
         indention += strlen(option->access_name) + 2;
      }

      if (option->value_name != NULL) {
         indention += strlen(option->value_name) + 1;
      }

      if (indention > result) {
         result = indention;
      }
   }

   return result;
}

void cag_option_print(const cag_option *options, size_t option_count,
                      FILE *destination)
{
   size_t option_index, indention, i, accessor_length;
   const cag_option *option;
   bool first;

   indention = cag_option_get_print_indention(options, option_count);
```

```
  for (option_index = 0; option_index < option_count; ++option_index) {
    option = &options[option_index];
    accessor_length = 0;
    first = true;

    fputs("  ", destination);

    cag_option_print_letters(option, &first, &accessor_length, destination);
    cag_option_print_name(option, &first, &accessor_length, destination);
    cag_option_print_value(option, &accessor_length, destination);

    for (i = accessor_length; i < indention; ++i) {
      fputs(" ", destination);
    }

    fputs(" ", destination);
    fputs(option->description, destination);

    fprintf(destination, "\n");
  }
}

void cag_option_prepare(cag_option_context *context, const cag_option *options,
                        size_t option_count, int argc, char **argv)
{
  // This just initialized the values to the beginning of all the arguments.
  context->options = options;
  context->option_count = option_count;
  context->argc = argc;
  context->argv = argv;
  context->index = 1;
  context->inner_index = 0;
  context->forced_end = false;
}

static const cag_option *cag_option_find_by_name(cag_option_context *context,
                                                 char *name, size_t name_size)
{
  const cag_option *option;
  size_t i;

  // We loop over all the available options and stop as soon as we have found
  // one. We don't use any hash map table, since there won't be that many
  // arguments anyway.
  for (i = 0; i < context->option_count; ++i) {
    option = &context->options[i];

    // The option might not have an item name, we can just skip those.
    if (option->access_name == NULL) {
      continue;
    }
```

```
    // Try to compare the name of the access name. We can use the name_size or
    // this comparison, since we are guaranteed to have null-terminated access
    // names.
    if (strncmp(option->access_name, name, name_size) == 0) {
      return option;
    }
  }

  return NULL;
}

static const cag_option *cag_option_find_by_letter(cag_option_context *context,
                                                   char letter)
{
  const cag_option *option;
  size_t i;

  // We loop over all the available options and stop as soon as we have found
  // one. We don't use any look up table, since there won't be that many
  // arguments anyway.
  for (i = 0; i < context->option_count; ++i) {
    option = &context->options[i];

    // If this option doesn't have any access letters we will skip them.
    if (option->access_letters == NULL) {
      continue;
    }

    // Verify whether this option has the access letter in it's access letter
    // string. If it does, then this is our option.
    if (strchr(option->access_letters, letter) != NULL) {
      return option;
    }
  }

  return NULL;
}

static void cag_option_parse_value(cag_option_context *context,
                                   const cag_option *option, char **c)
{
  // And now let's check whether this option is supposed to have a value, which
  // is the case if there is a value name set. The value can be either submitted
  // with a '=' sign or a space, which means we would have to jump over to the
  // next argv index. This is somewhat ugly, but we do it to behave the same as
  // the other option parsers.
  if (option->value_name != NULL) {
    if (**c == '=') {
      context->value = ++(*c);
    } else {
      // If the next index is larger or equal to the argument count, then the
```

```
        // parameter for this option is missing. The user will know about this,
        // since the value pointer of the context will be NULL because we don't
        // set it here in that case.
        if (context->argc > context->index + 1) {
          // We consider this argv to be the value, no matter what the contents
          // are.
          ++context->index;
          *c = context->argv[context->index];
          context->value = *c;
        }
      }

      // Move c to the end of the value, to not confuse the caller about our
      // position.
      while (**c) {
        ++(*c);
      }
    }
}

static void cag_option_parse_access_name(cag_option_context *context, char **c)
{
    const cag_option *option;
    char *n;

    // Now we need to extract the access name, which is any symbol up to a '=' or
    // a '\0'.
    n = *c;
    while (**c && **c != '=') {
      ++*c;
    }

    // Now this will obviously always be true, but we are paranoid. Sometimes. It
    // doesn't hurt to check.
    assert(*c >= n);

    // Figure out which option this name belongs to. This might return NULL if the
    // name is not registered, which means the user supplied an unknown option. In
    // that case we return true to indicate that we finished with this option. We
    // have to skip the value parsing since we don't know whether the user thinks
    // this option has one or not. Since we don't set any identifier specifically,
    // it will remain '?' within the context.
    option = cag_option_find_by_name(context, n, (size_t)(*c - n));
    if (option == NULL) {
      // Since this option is invalid, we will move on to the next index. There is
      // nothing we can do about this.
      ++context->index;
      return;
    }

    // We found an option and now we can specify the identifier within the
    // context.
```

```
    context->identifier = option->identifier;

    // And now we try to parse the value. This function will also check whether
    // this option is actually supposed to have a value.
    cag_option_parse_value(context, option, c);

    // And finally we move on to the next index.
    ++context->index;
}

static void cag_option_parse_access_letter(cag_option_context *context,
                                           char **c)
{
    const cag_option *option;
    char *n = *c;
    char *v;

    // Figure out which option this letter belongs to. This might return NULL if
    // the letter is not registered, which means the user supplied an unknown
    // option. In that case we return true to indicate that we finished with this
    // option. We have to skip the value parsing since we don't know whether the
    // user thinks this option has one or not. Since we don't set any identifier
    // specifically, it will remain '?' within the context.
    option = cag_option_find_by_letter(context, n[context->inner_index]);
    if (option == NULL) {
        ++context->index;
        context->inner_index = 0;
        return;
    }

    // We found an option and now we can specify the identifier within the
    // context.
    context->identifier = option->identifier;

    // And now we try to parse the value. This function will also check whether
    // this option is actually supposed to have a value.
    v = &n[++context->inner_index];
    cag_option_parse_value(context, option, &v);

    // Check whether we reached the end of this option argument.
    if (*v == '\0') {
        ++context->index;
        context->inner_index = 0;
    }
}

static void cag_option_shift(cag_option_context *context, int start, int option,
                             int end)
{
    char *tmp;
    int a_index, shift_index, shift_count, left_index, right_index;
```

```
  shift_count = option - start;

  // There is no shift is required if the start and the option have the same
  // index.
  if (shift_count == 0) {
    return;
  }

  // Lets loop through the option strings first, which we will move towards the
  // beginning.
  for (a_index = option; a_index < end; ++a_index) {
    // First remember the current option value, because we will have to save
    // that later at the beginning.
    tmp = context->argv[a_index];

    // Let's loop over all option values and shift them one towards the end.
    // This will override the option value we just stored temporarily.
    for (shift_index = 0; shift_index < shift_count; ++shift_index) {
      left_index = a_index - shift_index;
      right_index = a_index - shift_index - 1;
      context->argv[left_index] = context->argv[right_index];
    }

    // Now restore the saved option value at the beginning.
    context->argv[a_index - shift_count] = tmp;
  }

  // The new index will be before all non-option values, in such a way that they
  // all will be moved again in the next fetch call.
  context->index = end - shift_count;
}

static bool cag_option_is_argument_string(const char *c)
{
  return *c == '-' && *(c + 1) != '\0';
}

static int cag_option_find_next(cag_option_context *context)
{
  int next_index, next_option_index;
  char *c;

  // Prepare to search the next option at the next index.
  next_index = context->index;
  next_option_index = next_index;

  // Grab a pointer to the string and verify that it is not the end. If it is
  // the end, we have to return false to indicate that we finished.
  c = context->argv[next_option_index];
  if (context->forced_end || c == NULL) {
    return -1;
  }
```

```
   // Check whether it is a '-'. We need to find the next option - and an option
   // always starts with a '-'. If there is a string "-\0", we don't consider it
   // as an option neither.
   while (!cag_option_is_argument_string(c))  {
      c = context->argv[++next_option_index];
      if (c == NULL) {
         // We reached the end and did not find any argument anymore. Let's tell
         // our caller that we reached the end.
         return -1;
      }
   }

   // Indicate that we found an option which can be processed. The index of the
   // next option will be returned.
   return next_option_index;
}

bool cag_option_fetch(cag_option_context *context)
{
   char *c;
   int old_index, new_index;

   // Reset our identifier to a question mark, which indicates an "unknown"
   // option. The value is set to NULL, to make sure we are not carrying the
   // parameter from the previous option to this one.
   context->identifier = '?';
   context->value = NULL;

   // Check whether there are any options left to parse and remember the old
   // index as well as the new index. In the end we will move the option junk to
   // the beginning, so that non option arguments can be read.
   old_index = context->index;
   new_index = cag_option_find_next(context);
   if (new_index >= 0) {
      context->index = new_index;
   } else {
      return false;
   }

   // Grab a pointer to the beginning of the option. At this point, the next
   // character must be a '-', since if it was not the prepare function would
   // have returned false. We will skip that symbol and proceed.
   c = context->argv[context->index];
   assert(*c == '-');
   ++c;

   // Check whether this is a long option, starting with a double "--".
   if (*c == '-') {
      ++c;

      // This might be a double "--" which indicates the end of options. If this
```

```
      // is the case, we will not move to the next index. That ensures that
      // another call to the fetch function will not skip the "--".
      if (*c == '\0') {
         context->forced_end = true;
      } else {
         // We parse now the access name. All information about it will be written
         // to the context.
         cag_option_parse_access_name(context, &c);
      }
   } else {
      // This is no long option, so we can just parse an access letter.
      cag_option_parse_access_letter(context, &c);
   }

   // Move the items so that the options come first followed by non-option
   // arguments.
   cag_option_shift(context, old_index, new_index, context->index);

   return context->forced_end == false;
}

char cag_option_get(const cag_option_context *context)
{
   // We just return the identifier here.
   return context->identifier;
}

const char *cag_option_get_value(const cag_option_context *context)
{
   // We just return the internal value pointer of the context.
   return context->value;
}

int cag_option_get_index(const cag_option_context *context)
{
   // Either we point to a value item,
   return context->index;
}
```

- **main.cpp**

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
   QApplication a(argc, argv);
   MainWindow w;
   w.show();
   return a.exec();
}
```

- **mainwindow.cpp**

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "worker.h"
#include <iostream>

extern "C"
{
#include "../../../Hydra/inc/hydra_ftdi.h"
}

MainWindow::MainWindow(QWidget *parent)
   : QMainWindow(parent), ui(new Ui::MainWindow), workerThread(nullptr), timer(nullptr),
worker(nullptr)
{
   ui->setupUi(this);

   initializeHMC8043();
   if(init_thermo_control()!=0){
       std::cerr <<"Erreur lors de l'initialisation du controle du thermostream." << std::endl;
       return;

   }
   startButton = ui->edtstartButton;
   stopButton = ui->edtstopButton;

   //ui->e
  hochButton = ui->btnHoch;
  runterButton = ui->btnrunter;

      // Verbindung von Signalen

      connect(startButton, &QPushButton::clicked, this, &MainWindow::onStartButtonClicked);
      connect(stopButton, &QPushButton::clicked, this, &MainWindow::onStopButtonClicked);
      connect(hochButton, &QPushButton::clicked, this, &MainWindow::onhochButtonClicked);
      connect(runterButton, &QPushButton::clicked, this, &MainWindow::onrunterButtonClicked);

   // Créer l'objet Worker et le thread

   worker = new Worker;
   workerThread = new QThread;

   // Connecter le thread et l'objet Worker

   worker->moveToThread(workerThread);
   //worker->run();
   //Erstellen eines QTimers

   // Créer et initialiser le QTimer

   timer = new QTimer(this);
   timer->setSingleShot(false);   // Répéter le timer
   timer->setInterval(10000);     // 10 secondes d'intervalle
```

```
    worker->setMyFileName(ui->lineEdit->text());//aya

    workerThread->start();
}



MainWindow::~MainWindow()
{
    delete ui;
    delete worker;
    //delete workerThread;
    delete timer;
}

void MainWindow::onhochButtonClicked()
{
    if (thermo_move_head(0,  1) != 0) {
        printf("\nERROR: beim Bewegen des Kopfes nach oben.");
        return;
    }
}

void MainWindow::onrunterButtonClicked()
{
    if (thermo_move_head(1,  1) != 0) {
        printf("\nERROR: beim Bewegen des Kopfes nach unten.");
        return;
    }
}

void MainWindow::initializeHMC8043()
{
    if (init_supply_control(1)  != 0) {
        printf("\nERROR: LXI connection to supply failed...");
        return;
    }
}


void MainWindow::onStartButtonClicked()
{
    int ftdi_ret = init_ftdi("FT7UO3WU");
    fflush(stdout);

    if (ftdi_ret  != 0) {
        qDebug() << "Konnte FTDI GPIO nicht öffnen";
        return;
    }

    qDebug() << "Programm läuft:";
```

```
    // Create a new worker and start the thread

    worker->start();

    // Start the timer to trigger the updateVoltage slot every 10 seconds

    timer->start(10000);

    updateParameters();
}

void MainWindow::onStopButtonClicked()
{

    // Stop the worker thread

    if (worker) {
        worker->stop();
        //workerThread->quit();
        worker->wait();
        delete worker;
        worker = nullptr;
    }


    if (deinit_supply_control()  != 0) {
        printf("\nERROR: Deinit of LXI connection to HMC failed.");
        return;
    }

    // Stop the timer
    timer->stop();

}

void MainWindow::updateParameters()
{

    // Émettre le signal de mise à jour des paramètres

    worker->setParameters(
    ui->edtminTemperature->text().toDouble(),
    ui->edttemperatureStep->text().toDouble(),
    ui->edtmaxTemperature->text().toDouble(),
    ui->edtminVoltage->text().toDouble(),
    ui->edtvoltageStep->text().toDouble(),
    ui->edtmaxVoltage->text().toDouble(),
    ui->edtrepeat->text().toDouble(),
    ui->edttempStabil->text().toDouble(),
    ui->edtwait_time->text().toDouble()
    );
}
```

- **supply_control.c**

```
/*
 * supply_control.h
 *
 *  Created on: May 20, 2022
 *      Author: awalsemann
 */

#ifndef SUPPLY_CONTROL_H_
#define SUPPLY_CONTROL_H_

#define IP_ADDR_SUPPLY "192.168.0.150"

#define OVP_LIMIT 5.5
#define MAX_VOLTAGE 5.0
#define MAX_CURRENT 1.0

enum supply_target{
   ch1,
   ch2,
   ch3,
   all
};

int init_supply_control(int init);
int deinit_supply_control(void);

int enable_dut_supply(enum supply_target target);
int disable_dut_supply(enum supply_target target);

int set_voltage(double voltage, enum supply_target target);
int set_current(double current, enum supply_target target);

int measure(enum supply_target target, double * voltage, double * current);

#endif /* SUPPLY_CONTROL_H_ */
```

- **thermostream_control.c**

```
/*
 * thermostream_control.c
 *
 *  Created on: May 23, 2022
 *      Author: awalsemann
 */

#include <stdio.h>
#include <string.h>
#include <time.h>
#include "../../../Hydra/inc/thermostream.h"
```

```
#include "thermostream_control.h"

thermostream_connection_t  tstream;

int init_thermo_control(void){

   strcpy(tstream.ipv4,"192.168.0.90");
   tstream.port = 40957;
   strcpy(tstream.password,PASSWORD_THERMOSTREAM);

   if(thermostream_init(&tstream)){
      printf("\nERROR: Connection to Thermostream failed.");
      return -1;
   }

   if(thermostream_set_setpoint(&tstream, 1, THERMOSTREAM_AMB_TEMP)){
      printf("\nERROR: Setting of default Thermostream AMB Setpoint failed.");
      return -1;
   }
   if(thermostream_set_ramp(&tstream, 1, THERMOSTREAM_AMB_RAMP)){
      printf("\nERROR: etting of default Thermostream AMB Ramp failed.");
      return -1;
   }

   return 0;
}


int deinit_thermo_control(void){
   return thermostream_deinit(&tstream);
}


int thermo_move_head(int position, int lock){
   if(thermostream_unlock_head(&tstream)){
      printf("\nERROR: Unlocking of Thermostream Head failed.");
      return -1;
   }
   if(position == 1){
      if(thermostream_lower_head(&tstream)){
         printf("\nERROR: Lowering of Thermostream Head failed.");
         return -1;
      }
   }else{
      if(thermostream_raise_head(&tstream)){
         printf("\nERROR: Raising of Thermostream Head failed.");
         return -1;
      }
   }
   struct timespec ts = {0};
   ts.tv_sec = 1;
   nanosleep(&ts, NULL);
```

```
    if(lock == 1){
        if(thermostream_lock_head(&tstream)){
            printf("\nERROR: Locking of Thermostream Head failed.");
            return -1;
        }
    }

    return 0;
}

int thermo_trickle_control(int status){
    if(status == 1){
        if(thermostream_enable_trickle(&tstream)){
            printf("\nERROR: Thermostream Trickle Flow enable failed.");
            return -1;
        }
    }else{
        if(thermostream_disable_trickle(&tstream)){
            printf("\nERROR: Thermostream Trickle Flow disable failed.");
            return -1;
        }
    }

    return 0;
}


int thermo_flow_control(float value, int status){
    if(thermostream_set_flowtarget(&tstream, value)){
        printf("\nERROR: Thermostream Flowrate setpoint change failed.");
        return -1;
    }
    if(status == 1){
        if(thermostream_enable_flow(&tstream)){
            printf("\nERROR: Thermostream Main Flow enable failed.");
            return -1;
        }
    }else{
        if(thermostream_disable_flow(&tstream)){
            printf("\nERROR: Thermostream Main Flow disable failed.");
            return -1;
        }
    }

    return 0;
}


int thermo_read_flow(float * value){
    if(thermostream_read_flow(&tstream, value)){
        printf("\nERROR: Thermostream current Flowrate read failed.");
        return -1;
```

```
    }
    return 0;
}

int thermo_read_temps(float  * air_value, float * dut_value){
    if(thermostream_read_temperature(&tstream,  air_value, dut_value)){
        printf("\nERROR: Thermostream current Temperature read failed.");
        return -1;
    }
    return 0;
}

int thermo_set_setpoint(float   value, float ramp){
    if(value < 25.0){
        if(thermostream_set_ramp(&tstream,  2, ramp)){
            printf("\nERROR: Setting of Thermostream COLD Ramp failed.");
            return -1;
        }
        if(thermostream_set_setpoint(&tstream,  2, value)){
            printf("\nERROR: Setting of default Thermostream COLD Setpoint failed.");
            return -1;
        }
        int flow;
        if(thermostream_check_flow(&tstream,  &flow)){
            printf("\nERROR: Check of Thermostream flow status failed.");
            return -1;
        }
        if(thermostream_select_setpoint(&tstream,  2)){
            printf("\nERROR: Selecting of Thermostream COLD Setpoint failed.");
            return -1;
        }
        if(flow == 0){
            if(thermostream_disable_flow(&tstream)){
                printf("\nERROR: Thermostream Main Flow disable failed.");
                return -1;
            }
        }
    }else{
        if(thermostream_set_ramp(&tstream,  0, ramp)){
            printf("\nERROR: Setting of Thermostream HOT Ramp failed.");
            return -1;
        }
        if(thermostream_set_setpoint(&tstream,  0, value)){
            printf("\nERROR: Setting of default Thermostream HOT Setpoint failed.");
            return -1;
        }
        int flow;
        if(thermostream_check_flow(&tstream,  &flow)){
            printf("\nERROR: Check of Thermostream flow status failed.");
            return -1;
        }
        if(thermostream_select_setpoint(&tstream,  0)){
```

```
        printf("\nERROR: Selecting of Thermostream HOT Setpoint failed.");
        return -1;
      }
      if(flow == 0){
        if(thermostream_disable_flow(&tstream)){
          printf("\nERROR: Thermostream Main Flow disable failed.");
          return -1;
        }
      }
    }
    return 0;
}


int thermo_set_chiller(int  status){
    if(status == 0){
      if(thermostream_disable_chiller(&tstream)){
        printf("\nERROR: Enable of Thermostream Chiller failed.");
        return -1;
      }
    }else if(status == 1){
      if(thermostream_enable_chiller(&tstream)){
        printf("\nERROR: Disable of Thermostream Chiller failed.");
        return -1;
      }
    }else{
      printf("\nERROR: Thermostream Chiller invalid setting.");
      return -1;
    }

    return 0;
}

int thermo_read_chiller(int  * status){
    if(thermostream_check_chiller(&tstream,  status)){
      printf("\nERROR: Reading of Thermostream Chiller status failed.");
      return -1;
    }
    return 0;
}
```

- **worker.cpp**

```
#include "worker.h"
#include "mainwindow.h"

extern "C" {
#include "../../../Hydra/inc/hydra_ftdi.h"
#include "../../../Hydra/inc/scpi_hmc8043.h"
#include "../../../Hydra/inc/thermostream.h"
#include "arguments.h"
#include "cargs.h"
```

```
#include "supply_control.h"
};

#include <QDebug>
#include <QFile>
#include <QTextStream>
#include <QThread>

Worker::Worker(QObject *parent) : QThread(parent), running(false) {}

void Worker::setParameters(float minTemperature, float temperatureStep,
                    float maxTemperature, double minVoltage,
                    double voltageStep, double maxVoltage, int repeat,
                    double tempStabil, double waitTime) {

 // Aktualisierung der Parameter des Workers
 QMutexLocker locker(&mutex);
 this->minTemperature = minTemperature;
 this->temperatureStep = temperatureStep;
 this->maxTemperature = maxTemperature;
 this->minVoltage = minVoltage;
 this->voltageStep = voltageStep;
 this->maxVoltage = maxVoltage;
 this->repeat = repeat;
 this->tempStabil = tempStabil;
 this->waitTime = waitTime;
 qDebug() << "minVoltage:" << minVoltage << "maxVoltage:" << maxVoltage
        << "voltageStep:" << voltageStep;
}

bool Worker::readSerialData(QSerialPort &serial,int temperature,double voltage,
                     int repetition) {

 for (int i = 0; i < 10 ; ++i) {
  int ready = serial.bytesAvailable();
  if (ready >= 1536)
   break;
  serial.waitForReadyRead(100);
  QThread::msleep(500);
 }

  if (serial.bytesAvailable() < 1536)
     return false;

 QByteArray m_read = serial.read(1536);
 QString m_reads = QString::fromLocal8Bit(m_read.toHex());
 if (!myFile.isOpen()) {
  myFile.setFileName("output_data.txt");
  myFile.open(QIODevice::WriteOnly | QIODevice::Text);
  QTextStream stream(&myFile);
  stream.setFieldAlignment(QTextStream::AlignRight);
  stream << qSetFieldWidth(10) << "temperature,"
```

```
        << qSetFieldWidth(10)  << "voltage,"
        << qSetFieldWidth(15)  << "repetition,"
        << qSetFieldWidth(15)  << "Data," << '\n';
  }
  // die gelesenen Daten in die Ausgabedatei schreiben
  QTextStream stream(&myFile);
  stream.setFieldAlignment(QTextStream::AlignRight);
  stream << qSetFieldWidth(10)  << (QString::number(temperature)  + ",")
       << qSetFieldWidth(10)  << (QString::number(voltage)  + ",")
       << qSetFieldWidth(15)  << (QString::number(repetition)  + ",")
       << qSetFieldWidth(15)  << ((m_reads) + ",") << '\n';

    return true;
}

void Worker::reprogramFPGA()  {
  QProcess process;
  QString program = "openFPGALoader";  // Beispiel: Pfad zum ausführbaren Befehl
  QStringList arguments;

  arguments << "-c"
        << "gatemate_evb_jtag"
        << "/home/aya/PUF_BAT_00.cfg.bit";

  process.start(program, arguments);
  if (!process.waitForStarted())  {
    qDebug() << "Failed to start FPGA reprogramming process.";
    return;
  }

  if (!process.waitForFinished())  {
    qDebug() << "FPGA reprogramming process finished with an error:"
          << process.errorString();
    return;
  }

  qDebug() << "FPGA reprogramming process finished successfully.";
}

void Worker::reprogramAndSetVoltage(double  voltage) {

  if (set_voltage(0.0, ch1) != 0) {
    qDebug() << "Fehler beim Setzen der Spannung auf Null.";
    return;
  }

  //QThread::msleep(2000);

  if (set_voltage(voltage, ch1) != 0) {
    qDebug() << "Fehler beim Setzen der Spannung auf das Zielniveau.";
    return;
  }
```

```
  // temps de stabilisation
  QThread::msleep(1000);
  reprogramFPGA();
  QThread::msleep(1000);
}

void Worker::run()
{
   int errcnt=0;
    if (thermo_flow_control(4.0, 1) != 0)
    {
        printf("\nERROR: Enabling Thermostream Main Flow failed.");
        return;
    }
    // activer l"alimentation de l appareil sous test DUT
  if (enable_dut_supply(ch1)  != 0)
  {
    printf("\nERROR:enabling  DUT supply");
  }

  // Contrôle du débit


  QSerialPort serial;

  serial.setBaudRate(9600);

  serial.setPortName("/dev/serial/by-id/usb-FTDI_FT232R_USB_UART_A503VSLI-if00-port0");
  if (!serial.open(QIODevice::ReadWrite))  {
    qDebug() << "ERROR: opening serial port" << serial.errorString();
  }

  for(int temperature = minTemperature;  temperature <= maxTemperature;  temperature  += temper-
atureStep)
  {
      if (thermo_set_setpoint(temperature,  20.0) != 0)
      {
          printf("\nERROR: Setting of default Thermostream COLD Setpoint failed.");
          return;
       }
       if(temperature==minTemperature)
       {
           QThread::sleep(tempStabil);
       }
       else
       {
         QThread::sleep(waitTime);
       }
        // Temperaturen ablesen
        float airTemperature, dutTemperature;
        if (thermo_read_temps(&airTemperature,  &dutTemperature)  != 0)
        {
```

```
        printf("\nERROR: Thermostream current Temperature read failed.");
        return;
    }
    float flowRate;
    if (thermo_read_flow(&flowRate) != 0)
    {
        printf("\nERROR: Thermostream current Flowrate read failed.");
        return;
    }
    std::cout << "Flowrate: " << flowRate << std::endl;
    for (double voltage = minVoltage; voltage <= maxVoltage; voltage += voltageStep)
    {
        // Voltage ablesen
        double measuredVoltage, measuredCurrent;
        if (measure(ch1, &measuredVoltage, &measuredCurrent) != 0)
        {
          printf("\nERROR: measuring.");
          return;
        }
        for (int repetition = 0; repetition < repeat; )
        {

         qDebug() << "Starte wiederholung " << repetition << " von " << repeat;
         ftdi_set_gpio(0x00);
         reprogramAndSetVoltage(voltage);
         ftdi_set_gpio(0xff);
         // Perform the necessary actions with the measured values
         qDebug() <<"Ziel temperature:" << temperature<<"Ziel Voltage:" << voltage << "repetition:"
<< repetition;
         // declare un tableau de caractere de taille 1536 NICHT löschen
         //char testdata[1536];
         //memset(testdata, 0, 1536);
         // serial.write(testdata, 1536);
         // serial.flush();
         ftdi_set_gpio(0x00);
         qDebug() << "Starte empfang";
         if (readSerialData(serial,temperature ,voltage, repetition))
         {
            ++repetition;
             qDebug() << "serial data empfangen";
             errcnt=0;
         }
         else
         {
            errcnt++;
            qDebug() << "Fehlerhafter Empfang Nr.: " << errcnt;
         }
        }
    }

  }
 // Stop der thermo_Controlle
```

```
  if (deinit_thermo_control()  != 0) {
      printf("\nERROR: stopping the thermo control.");
      return;
  }
 // stop the HMC
 if (disable_dut_supply(all)  != 0) {
   printf("\nERROR: Disable of HMC DUT supply channel failed.");
   return;
 }
 serial.close();

}

void Worker::setMyFileName(const QString &newMyFileName) {
 myFileName = newMyFileName;
}

void Worker::stop() {
 running = false;
 myFile.close();
}
```