ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Integrierte Systeme
Integrated Systems Laboratory

Department of Information Technology and
Electrical Engineering

Spring Semester 2025

Design and Validation of a Framework for Emerging Nanoscale Memory Characterization on an FPGA Platform

Semester Project

Jeff Ren
renj@student.ethz.ch.com

May 2025

Supervisors: Till Zellweger, till.zellweger@iis.ee.ethz.ch, Dr. Nikhil Garg, nigarg@ethz.ch
PD. Dr. Emboras Alexandros, emborasa@iis.ee.ethz.ch

Professor: Prof. Dr. Laura Bégon-Lours, lbegon@ethz.ch
Prof. Dr. Mathieu Luisier, mluisier@iis.ee.ethz.ch

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten schriftlichen Arbeit. Eine der folgenden zwei Optionen ist **in Absprache mit der verantwortlichen Betreuungsperson** verbindlich auszuwählen:

☐ Ich erkläre hiermit, dass ich die vorliegende Arbeit eigenverantwortlich verfasst habe, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge der Betreuungsperson. Es wurden keine Technologien der generativen künstlichen Intelligenz[1] verwendet.

☒ Ich erkläre hiermit, dass ich die vorliegende Arbeit eigenverantwortlich verfasst habe. Dabei habe ich nur die erlaubten Hilfsmittel verwendet, darunter sprachliche und inhaltliche Korrekturvorschläge der Betreuungsperson sowie Technologien der generativen künstlichen Intelligenz. Deren Einsatz und Kennzeichnung ist mit der Betreuungsperson abgesprochen.

Titel der Arbeit:

Design and Validation of a Framework for Emerging Nanoscale Memory Characterization on an FPGA Platform

Verfasst von:
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Ren	Jeff

Ich bestätige mit meiner Unterschrift:
- Ich habe mich an die Regeln des «Zitierleitfadens» gehalten.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu und vollständig dokumentiert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Eigenständigkeit überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
Zürich, 10.06.25	

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie grundsätzlich gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.

[1] Für weitere Informationen konsultieren Sie bitte die Webseiten der ETH Zürich, bspw. https://ethz.ch/de/die-eth-zuerich/lehre/ai-in-education.html und https://library.ethz.ch/forschen-und-publizieren/Wissenschaftliches-Schreiben-an-der-ETH-Zuerich.html (Änderungen vorbehalten).

Acknowledgements

First, I would like to express my gratitude for my supervisors Till Zellweger and Dr. Nikhil Garg for helping to familiarize myself with the laboratory equipment and generally guiding me through this entire project with a lot of patience, involving countless hours of meetings and lab sessions. Alexandre Baigol Sisó also provided valuable guidance and generously offered his own chips for measurements. In addition, I appreciate all the valuable insights from the ArC Instruments team who assisted with the usage of the ArC TWO board. I also thank Prof. Dr. Laura Bégon-Lours for valuable input and providing the ArC TWO board for the whole semester.

Furthermore, I want to thank PD. Dr. Emboras Alexandros and Prof. Dr. Mathieu Luisier for giving me the opportunity to work on this project. Lastly, I want to thank all the members that I have met from the Brain Inspired Device and Circuits Group and the Neuromorphic Electronics using Oxides Group for providing a friendly working environment where I felt very welcome throughout the semester.

Abstract

As artificial intelligence advances, machine learning workloads are consuming an unprecedented amount of data, hence they demand new computing architectures beyond traditional von Neumann systems. In-memory computing architectures using memristive crossbar arrays for Vector-Matrix Multiplication (VMM) are shown to be a promising solution, as VMM is the key operation in Artificial Neural Networks (ANN) which can be significantly accelerated by in-memory computing with analog memristive arrays. However, with the diversity of emerging memristive device technologies, significant challenges for scalable and reliable device characterization arise. Single-device characterization requires broad voltage and current ranges, long-term stability testing, and precise control over dynamic switching behavior. Array-level measurements require scalability for parallel channels, making manual probing impractical.

In this project, a flexible and extensible framework with testing modules was developed for the ArC TWO board, a FPGA-based 64-channel testing platform, enabling efficient and cost-effective memristive device and crossbar characterization. Based on the provided Python wrapper API, various protocols for single-device characterization such as IV characteristics, long-term potentiation and depression (LTP/LTD), retention, endurance, and pulse characterization were implemented. The pulsing capabilities were validated with analysis of overshoot, speed limitations, and buffer-induced execution stalls. The system was demonstrated using HfO_2/ZrO_2 based ferroelectric Tunnel Junctions (FTJs) and TiN/CMO-HfO_x based Valence Change Memory (VCM) devices. A crossbar control module was tested on a dummy 2x2 array, establishing the foundation for scalable, automated benchmarking of neuromorphic hardware.

Contents

List of Acronyms

1T1R . . . . . One Transistor One Resistor

ANN . . . . . Artificial Neural Network
API . . . . . Application Programming Interface

BEOL . . . . . Back-End-Of-Line
BNC . . . . . Bayonet Neill-Concelman

CMO-HfOx . . . Cobalt Manganese Oxide - Hafnium Oxide
CMOS . . . . . Complementary Metal-Oxide-Semiconductor

DAC . . . . . Digital-to-Analog Converter
DUT . . . . . Device Under Test

FET . . . . . Field-Effect Transistor
FIFO . . . . . First-In, First-Out
FPGA . . . . . Field-Programmable Gate Array
FTJ . . . . . Ferroelectric Tunnel Junction

GPIO . . . . . General Purpose Input/Output
GUI . . . . . Graphical User Interface

HDF5 . . . . . Hierarchical Data Format version 5
HfO_2 . . . . . Hafnium Dioxide
HRS . . . . . High Resistance State

IV . . . . . Current-Voltage

LRS . . . . . Low Resistance State

LTD . . . . . Long-Term Depression
LTP . . . . . Long-Term Potentiation

PCB . . . . . Printed Circuit Board
PIM . . . . . Processing-in-Memory

RLC . . . . . Resistor-Inductor-Capacitor

SMA . . . . . SubMiniature version A
SMU . . . . . Source Measure Unit
STDP . . . . . Spike-Timing-Dependent Plasticity

TIA . . . . . Transimpedance Amplifier
TiN . . . . . Titanium Nitride
TOML . . . . . Tom's Obvious, Minimal Language

VCM . . . . . Valence Change Memory
VMM . . . . . Vector-Matrix Multiplication

WOx . . . . . Tungsten Oxide

ZrO_2 . . . . . Zirconium Dioxide

Chapter 1

Introduction

In the era of artificial intelligence, machine learning workloads continue to scale up, resulting in more memory-heavy computations. Today's traditional computing systems are based on the von Neumann architecture. Memory and processing units are separated where the main limitation is the von Neumann bottleneck – the separation between memory and processing units - that leads to high latency and power consumption due to constant data transfers [1].

This is where neuromorphic computing comes into play. In this field, researchers take inspiration from the human brain, mimicking its structure by placing computation and memory in the same physical location [2]. This paradigm is also known as Processing-in-Memory (PIM) and eliminates the memory bus, resulting in energy-efficient processing without data movement [3].

A promising technology for implementing PIM and enabling neuromorphic computing are memristors. When arranged in in an crossbar array setting, memristors naturally perform Vector-Matrix-Multiplications (VMM) - a key operation in neural network inference - by leveraging fundamental physical laws. Specifically, through Ohm's law, a matrix weight is controlled by the current through each memristor, while Kirchhoff's current law enables the summation of currents at each node, representing the output vector. Hence, the entire VMM can be executed in effectively $O(1)$ time complexity, as all multiplications and additions occur simultaneously in analog across the array. As a result, crossbar arrays offer massive parallelism and energy efficiency, bypassing the von Neumann bottleneck in conventional digital architectures.[4, 5].

This has driven intense global research into memristive devices using many different technologies [6]. In the research groups Brain-Inspired Devices and Circuits and Neuromorphic Electronics with Oxides, they have developed and manufactured single devices and crossbars using Valence Change Memory (VCM) and Ferroelectric Tunnel Junction (FTJ) technologies. The setup in the lab allowed for testing for single devices with signal

generators, oscilloscopes and manual probe stations. This is sufficient for single-device measurements, but becomes impractical for larger crossbars.

To address this issue, we used the Arc TWO board by ArC Instruments, an FPGA-based general purpose testing system tailored for memristor and crossbar research. It features a 64-channel source meter unit (SMU) array for analog measurements. These include trans-impedance amplifiers (TIA) and independent pulse generators covering pulse widths down to 40ns. For digital control, it offers two banks of 32 I/O pins [7]. Combining this with an available test socket daughter board, the setup allows for testing of up to 32x32 wire bonded crossbar arrays, either passive (NT1R) or active using selector transistors (1T1R), controlled by the digital pins. Traditional testing platforms for memristor and crossbar arrays often force trade-offs between throughput, measurement precision, and cost. High-end SMU-based systems (e.g. Keysight B1500A) offer exceptional accuracy but are inherently sequential and costly, limiting their scalability to large arrays. On the other end, custom ASIC testers deliver exceptional throughput but at the expense of accessibility, reconfigurability, and development cost. The Arc TWO platform uniquely fills this gap. It delivers sufficient measurement accuracy for resistive switching analysis, while maintaining the flexibility and affordability needed for rapid prototyping and exploratory research. This balance makes it particularly well-suited for emerging memory characterization and neuromorphic hardware testing, where throughput, adaptability and cost are equally critical.

The goal of this project was to implement a framework and testing modules on the Arc TWO board to help laboratories with device characterization and set the groundwork for future crossbar measurements. In particular, single device characterization methods such as IV characteristics and pulse based protocols for multi-level programming, retention and endurance protocols were implemented and validated.

Chapter 2

Background and Related Work

This section provides a foundational understanding of memristive devices, their operational principles, and the various techniques used for their electrical characterization. It also touches upon challenges in array integration, specifically focusing on sneak path issues in crossbar architectures.

2.0.1. Memristive Switching Devices

A memristor is a two-terminal electrical device whose resistance can be tuned by applied voltage signals. This change in resistance is non-volatile, based on the history of charge flow through the device, allowing it to retain its programmed state even without power. Its ability to be programmed to multiple, stable resistance states makes it a promising technology for non-volatile memory and neuromorphic computing.[4][5] There are many memristive technologies actively being researched. In this project we specifically focus on Ferroelectric Tunnel Junctions (FTJ) and Valence Change Memory (VCM) devices.

Filamentery VCM devices

Filamentary Valence Change Memory (VCM) devices operate by modulating resistance through the formation and rupture of a conductive filament within an oxide. An initial forming step then involves applying a positive voltage such that an electric field drives an oxygen exchange reaction. Oxygen ions migrate away and leave behind positively charged oxygen vacancies in the oxide shown in Fig. 2.1b). These vacancies then aggregate to form a conductive filament, bridging the electrodes (Fig. 2.1c)). Once formed, the device resistance can be tuned by further applying electric fields which leads to movement of these oxygen vacancies within the filament.

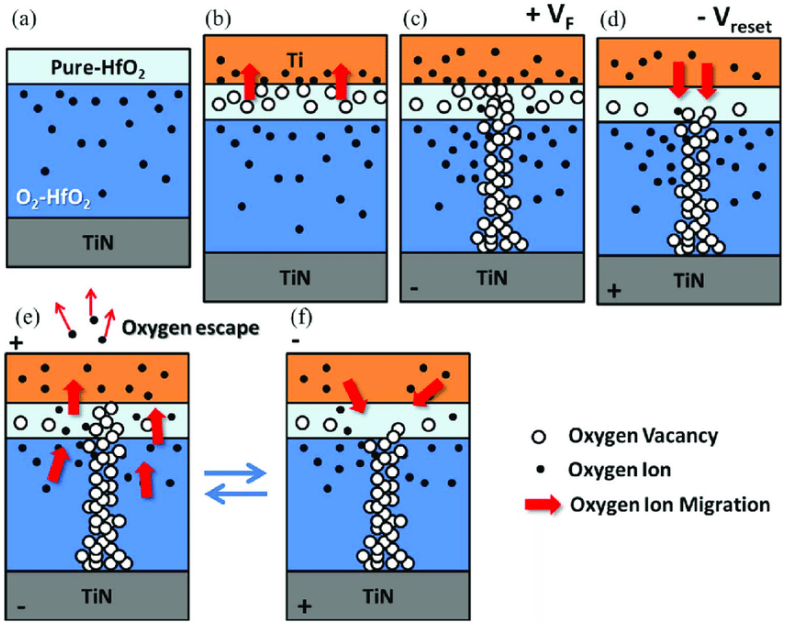

Figure 2.1.: VCM working principle. Adapted with permission from [8], fig. 3, p. 549

This ability to finely control the filament conductivity by adjusting the oxygen vacancy concentration (Figure 2.1e),f)), allows VCM devices to achieve various stable intermediate resistance states, achieving multi-level resistance programming, meaning, we can store multiple bits of information on a single device. In architectures like crossbar arrays, these tunable resistors can directly represent synaptic weights, enabling efficient in-memory computation for artificial neural networks. The VCM devices used here are based on a TiN/CMO-HfO_x/TiN structure [9] shown in figure 2.2 adapted from Falcone et al. in Ref. [9].

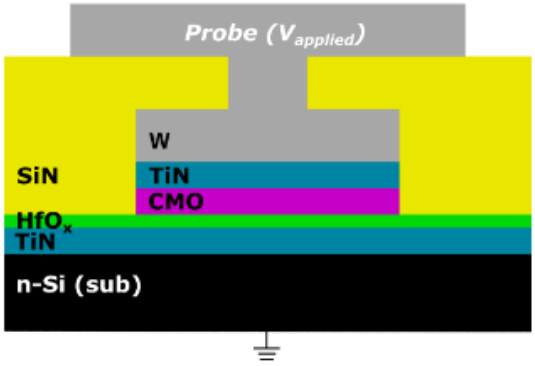

Figure 2.2.: Cross Section of VCM device

Ferroelectric devices

Ferroelectric resistive devices used in this project are two terminal Ferroelectric Tunnel Junctions (FTJs) which modulate resistance using the material's polarization states. They utilize electric dipoles in ferroelectric crystal structures, where the barycenter of cations is offset relative to anions, creating a permanent electrical dipole. An example with Barium Titanide is shown in 2.3

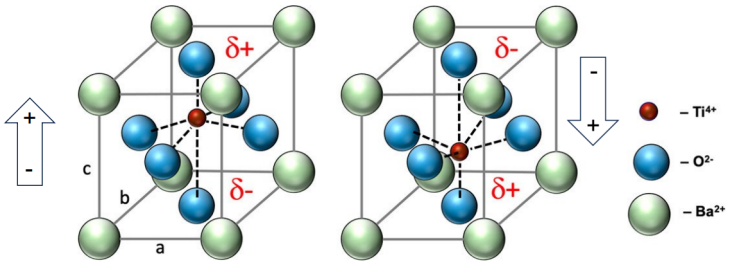

Figure 2.3.: Dipol in ferrolectrics with Barium ions adapted from [10]

When applying a voltage across the two terminals, the resulting electric field switches the orientation of the ferroelectric polarization P (a_1 and b_1 in Figure 2.4). Due to asymmetry between the two electrodes, this polarization change leads to unequal charge screenings at the interfaces, modifying its electrostatic profile $\phi(x)$ (shown in a_2 and b_2). This results in different shapes and heights of the tunneling barrier $E_B(x)$ (a_3 and b_3). The higher barrier in b_3, corresponds to the high resistive state, while the lower barrier in a_3 corresponds to the low resistance state.

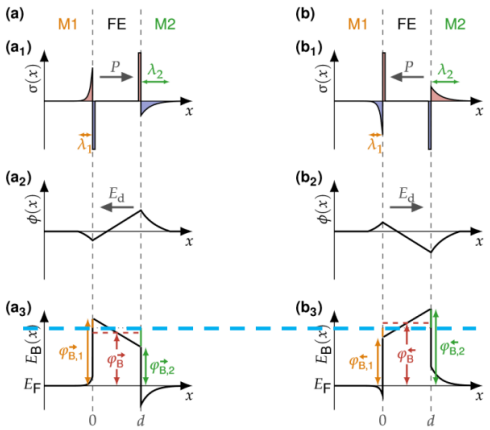

Figure 2.4.: Band diagrams of FTJ

The devices fabricated by the NEO group are based on HfO_2/ZrO_2 nanolaminates (Figure 2.5, adapted from Bégon-Lours in Ref. [11]). Although both electrodes are made of TiN—ensuring CMOS BEOL compatibility—the WO_x interlayer (purple layer on the right), placed on only one side of the ferroelectric, introduces the necessary asymmetry for resistive switching. As Bégon-Lours et al.[11] described, the resulting asymmetry in the energy band diagram arises not from the electrode materials, but from the direction of the ferroelectric polarization relative to the WO_x layer: when the polarization points toward WO_x, the tunneling barrier is affected differently than when it points away.

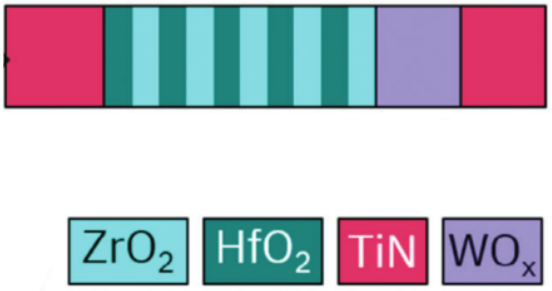

Figure 2.5.: Cross section of ferroelectric resistive device

2.0.2. Electrical Characterization Methods for Memristive Devices

To evaluate the performance and application potential of memristive and ferroelectric devices, a set of standard electrical characterization techniques is employed. These methods investigate critical device properties such as switching behavior, stability, endurance and analog tuning capability. Based on the paper by Stathopoulos et al.[12] on characterization methologies for memristive devices, the following methods were deployed:

IV Characteristics

The current-voltage (IV) characteristic curve is the fundamental method for analyzing the resistive switching behavior of memristor devices. By sweeping through voltages and measuring the corresponding current, one can identify the physical properties such as SET and RESET voltage, hysteresis behavior and the nature of the switching.

LTP/LTD

Long-Term Potentiation (LTP) and Long-Term Depression (LTD) protocols are used to simulate synaptic plasticity by gradually adjusting the device conductance using a series of short pulses. This is done by first increasing the conductance (potentiation) and then

decreasing it (depression). These protocols evaluate the device capabilities for analog or multilevel switching, essential for neuromorphic computation applications.

Retention

Retention tests assess the non-volatility of the resistance states in memristors. After programming the device into an initial resistance state, it is frequently monitored over a long period of time. This long-term stability and reliability is essential for memory applications.

Endurance

Endurance tests measure how well a device can withstand repeated switching. This is done by repeating many consecutive SET and RESET pulses and track the resistance state after each pulse. This protocol is essential for evaluating device degradation or drift in switching parameters, hence understanding fatigue behavior and lifetime.

2.0.3. Crossbar Arrays and Sneak Paths Currents

Memristive devices are often integrated into crossbar array configuration to achieve high-density memory or neuromorphic computing systems. In this section we introduce the structure of crossbar arrays, the primary challenge in form of sneak path currents and describe common mitigation strategies for both active and passive configurations.

Crossbar Array Architecture

A crossbar array consists of two sets of metal lines perpendicular to each other - word and bit lines. At each intersection point, a memristor is placed and acts as a programmable resistor. This structure can then be utilized as memory or synaptic elements in various applications. Figure 2.6 shows an example 2x2 crossbar.

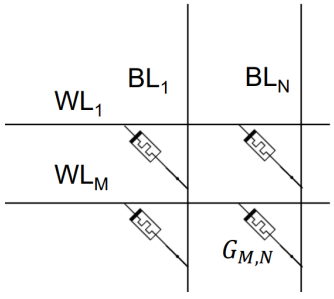

Figure 2.6.: 2x2 crossbar illustration reproduced13 from [10]

Sneak Path Currents

When memristive devices are arranged in a crossbar fashion, undesired leakage currents occur, causing errors in programming the devices and generally dissipate heat. Figure 2.7 shows the same example 2x2 crossbar with an illustration of how sneak path currents are established. Particularly, the worst case is shown where the selected cell is in a high resistance state (HRS) and the unselected cells are in a low resistance state (LRS). Due to Ohm's law and Kirchhoff's Current Law, the majority of the current is diverted through the adjacent LRS cells. This results in incomplete or incorrect switching of the selected device.

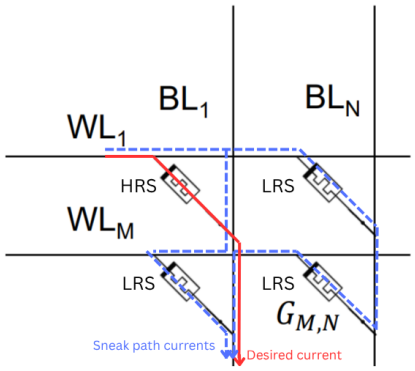

Figure 2.7.: Sneak path currents in crossbars adapted from[10]

One approach to mitigate this issue is to use transistors in series to the memristors shown in Figure 2.8, often called 1 Transistor 1 Resistor (1T1R) or active crossbar configuration. In this way, only the devices to be programmed can be selected, completely eliminating sneak paths. However, this requires a more complex architecture when using 3 terminal nodes.

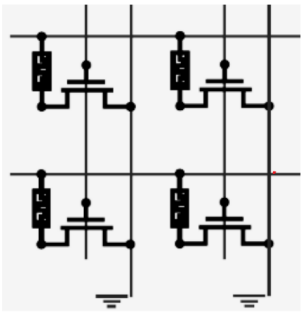

Figure 2.8.: 1T1R configuration adapted from [10]

When dealing with passive array configurations, we need a different approach in order to handle unwanted programming of untargeted cells. Adayemo et al. [13] propose three different writing schemes: $V/2$, $V/3$ and floating. The general idea for all of these schemes, illustrated in Figure 2.9, is to bias the word and bit lines in a way, such that the targeted cell (red cell) sees the full voltage while the half-selected (green and orange) cells and unselected (blue) cells see a low enough voltage drop in order to stay unaffected and hence do not suffer from unwanted programming.

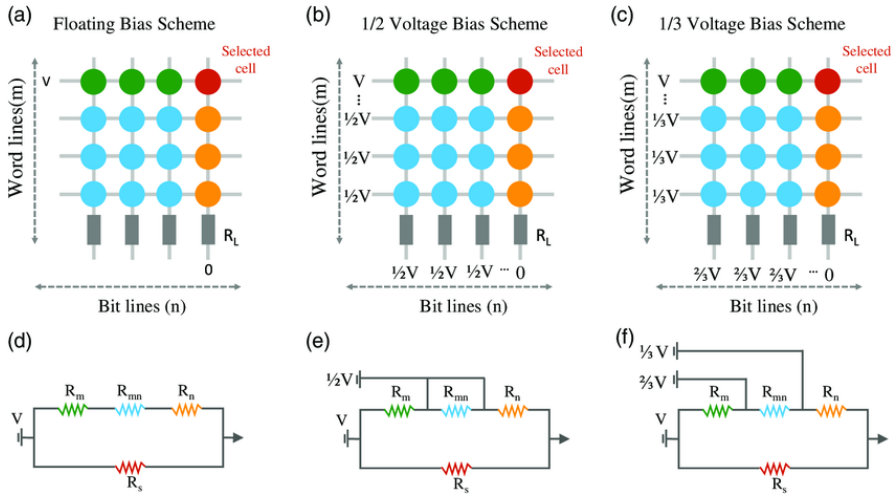

Figure 2.9.: Biasing schemes adapted from Li et al. (2021) [14]. CC BY 4.0 (creativecommons.org/licenses/by/4.0/)

Chapter 3

Methods and Implementation

3.0.1. ArC TWO platform

The ArC TWO board is a compact and generic electrical characterization platform developed by ArC Intruments for testing a wide range of electronic such as single devices, microchips, or any complex circuits. It serves as the central instrument used throughout this project for electrical measurements.

System Overview

Figure 3.1 shows the ArC TWO main board and alongside its available daughterboards which can be attached to the mezzanine connector for analog and digital I/O. The system is built around a Xilinx XC7A200T FPGA, which orchestrates all measurements, interfacing with a host PC over USB 3.0. [15]

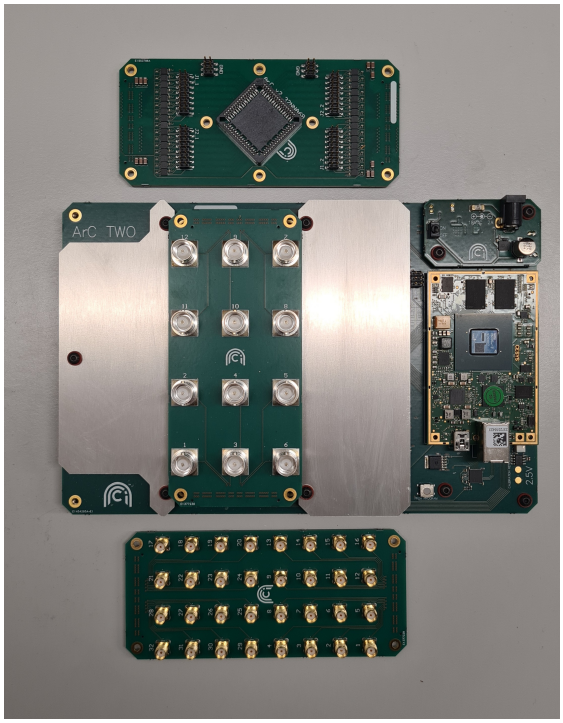

Figure 3.1.: ArC TWO and daughterboards

Hardware Features

The ArC TWO's modular architecture and high-channel count enable both single-device and large-array measurements. Key hardware features are [16]:

- 64 fully independent analog channels Source Meter Unit (SMU) channels and pulse generators and access to unified current source
- 32 digital outputs with arbitrary high/low levels at ±13.5V
- 32 digital I/Os with arbitrary high level at 1.8-5.5V
- 4 arbitrary supplies at ±13.5 V and ±100 mA
- Modular daughterboards with different connection types:
  - passive 12 channel BNC breakout
  - passive 32 channel SMA breakout
  - active 64-channel PLCC68 daugtherboard for wire bonded chips

Active in this context means that there is additional on-board logic for switching between the socket (PLCC68) and the headerbanks, seen on the left and right side of the socket in Figure 3.1.

The lack of tunable compliance current capabilities was a major concern, as it prevents safe testing of real devices - particularly considering the risk of damaging a fully wire-bonded chip which requires significant time and effort to fabricate. As mentioned in the in the listing of hardware features, all channels have a fixed compliance current of 100mA but this is too high for a lot of memristive devices.

Software Features

The ArC TWO API in pyarc2 provides Python bindings for low-level hardware control via the rust based libarc2 library underneath. The main role of pyarc2 is to expose essential hardware operations. These includes:

- Channel grounding, floating and open/close configuration
  (e.g ground_all, float_all, config_channel)
- Voltage and current measurement routines
  (e.g read_one, read_slice, read_all, vread_channels)
- Pulse generation control and timing with high-speed drivers or slow conventional biasing
  (e.g pulse_one, pulse_slice, pulse_all)
- Advanced operations for ramp and read-train measurements
- Data retrieval and conversion
  (e.g arc.pick_one, get_iter)
- Digital logic configuration for GPIOs and switches.
- Auxiliary DAC configuration for specialized voltage supplies and references

The design of this library is intentionally held low-level and generic. There is no application-specific logic, sequencing or automation. It is the user's responsibility to implement safe and consistent measurement protocols and perform data analysis. Although a graphical user interface (GUI) is available, it was not utilized for this project due to its limited flexibility compared to scripting in Jupyter notebooks, facilitating custom protocol development and data analysis.

Digital Interface

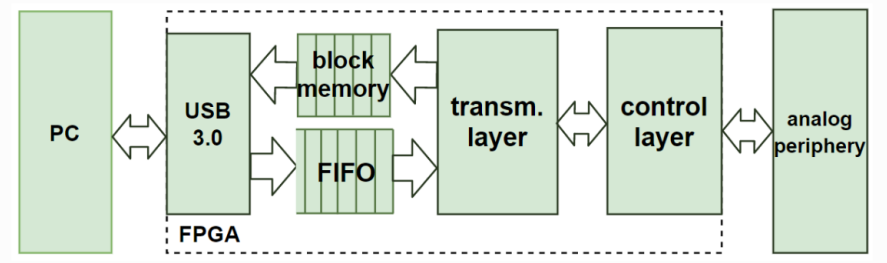

Figure 3.2.: Hierarchy of the digital interface from ArC Instruments [17]

Figure 3.2 shows the digital interface hierarchy of the ArC TWO board. At the top level, the Python API configures the hardware actions. These method calls are first stored in the ArC TWO's internal First-in-First-Out (FIFO command buffer). Then the transmission layer generates PCB level instructions and the control layer executes them. But these commands are not executed immediately. Instead, they are queued in the internal buffer until the user calls execute(), which flushes the FIFO buffer and triggers the hardware execution. This queuing system improves performance by reducing USB communication overhead. The following code provides an example of how a set of arbitrary commands can be executed.

```
1    # ground channels 0,1 and 2
2    arc.connect_to_gnd([0, 1, 2])
3    # set channel 0 to 1.5V and channel 1 to 1V, other channels
        remain in current state
4    arc.config_channels([(0, 1.5), (1, 1.0)], base=None)
5    # apply pulse between channel 0 (low) and 1 (high)
6    arc.pulse_one(0, 1, 2.5, 50000)
7    # execute queued commands in FIFO order
8    arc.execute()
```

On the other hand, the block memory (on top of the FIFO command buffer in the figure 3.2) is used to store the measurement results. When using deferred reading operations, the data remains in the block memory until the PC explicitly reads it out. If the block memory fills up due to uncollected results, further hardware operations may stall.

3.0.2. Measurement setup

To evaluate individual device characteristics, a probe station within the Brain-Inspired Devices and Circuits Group laboratory was used for all measurements. The system was configured to allow seamless toggling between the conventional measurement instruments and the ArC TWO system using a switching matrix.

Single Device Measurements

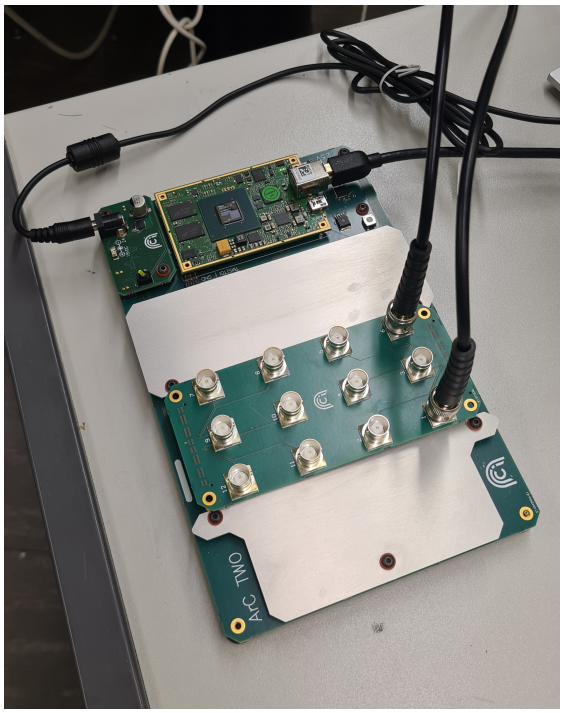

Figure 3.3.: ArC single device connection using BNC

Figure 3.3 shows the ArC TWO board interfacing with the measurement setup through the BNC breakout board. For two terminal devices this board was sufficient for the low number of connections and rather low-frequency operation with the shortest pulse possible being 40ns. BNC connectors on the other hand are typically rated up to 2-4 GHz, which equals to sub-nanosecond pulses.

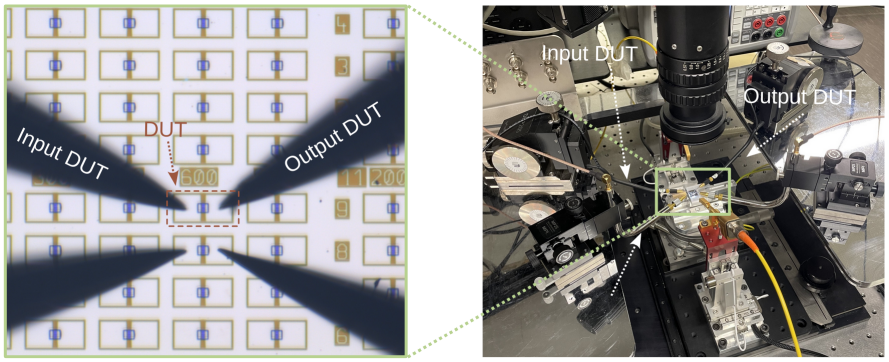

Figure 3.4.: lab setup using probes

Figure 3.4 provides a detailed view of the probe station. The left side shows an optical microscope image of the DUT with input and output probes creating direct electrical contacts. These probes are then directly connected to the switching matrix. The right side of Figure 3.4 shows the probe station itself including microscope and probing arms. Utilizing the switching matrix, we could toggle the electrical connections of the probes between the laboratories separate Source-Measure Units (SMUs) and waveform generator and the ArC TWO system. This configuration allowed for device verification and establishment of reference measurements, all while not having to reposition or rewire the DUT.

Crossbar Array Measurements

When doing measurements on a crossbar array - mentioned earlier in the hardware section - either the passive BNC and SMA boards can be used for lower scale arrays in combination with the probe station (depending on how many probes are available) or the socket board for directly placing a wire bonded chip on the ArC system. The 64 independent channels are configurable as inputs or outputs. This is particularly useful for crossbar array measurements, where the total number of channels needed is the sum of its wordlines (rows) and bitlines (columns). With this setup the platform can measure any rectangular crossbar array using 64 combined In- and Ouput channels. For instance it can handle a 32x32 square array using 32 channels as inputs and 32 channels as outputs.

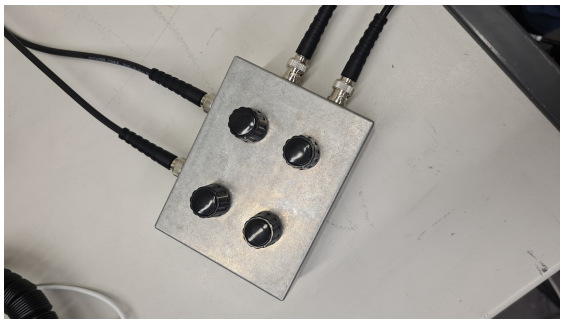

Figure 3.5.: 2x2 Dummy Crossbar

Due to the limited duration of this semester project, it was not possible to do crossbar measurements on fabricated devices. Instead, the validation of measurement code and addressing schemes for crossbar arrays was performed on a dummy 2x2 crossbar (Figure 3.5) with BNC connections. It consist of four potentiometers as tunable resistors which can be employed to emulate the behavior of a crossbar array.

3.0.3. Code Structure and Workflow

The measurement framework is organized into two parallel branches: pulse-based measurements and IV sweeps. The flow chart in Figure 3.6 illustrates both branches each producing standardized results for analysis but are fundamentally different in how measurements are defined and executed.

Pulse Measurement Workflow

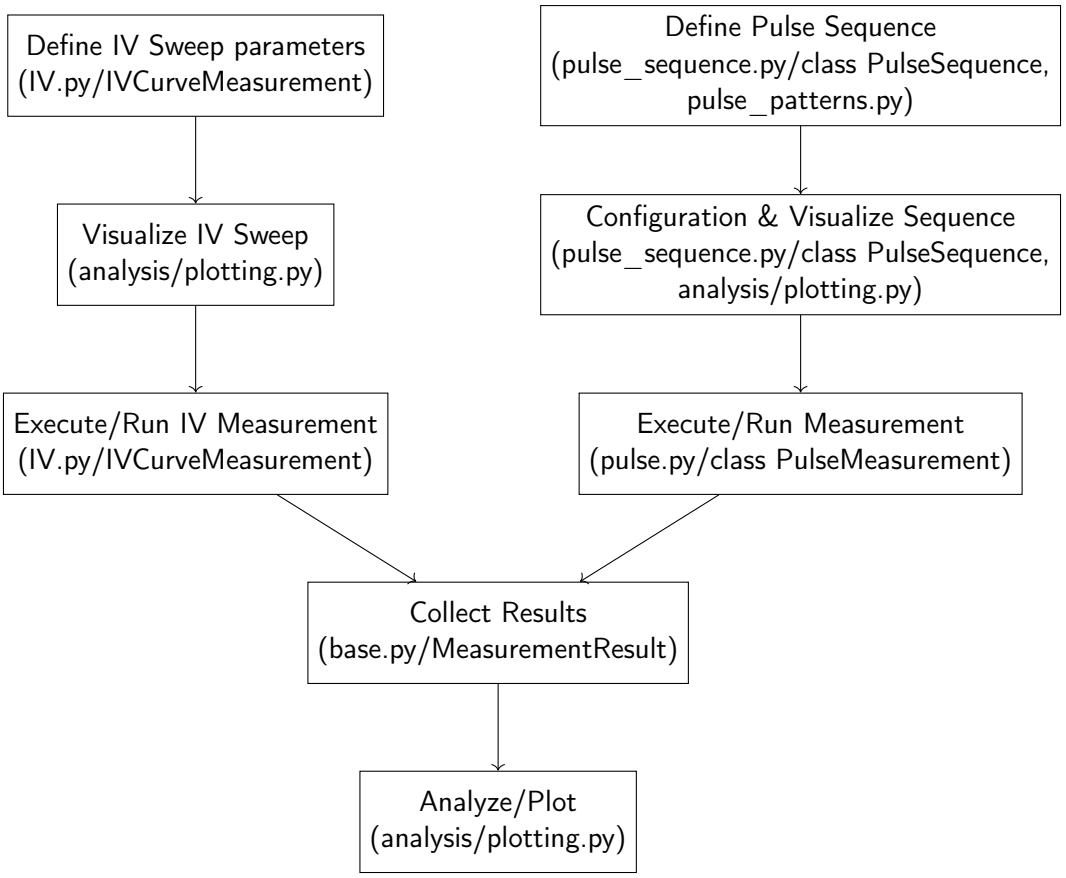

Figure 3.6.: worklow of IV and pulse measurements

At the heart of the pulse measurement is the PulseSequence class (pulse_sequence.py), which provides a flexible and extensible way to define and generate the desired pulse sequence. This class is designed to support any arbitrarily timed pulse trains.

To promote reusability, the actual construction of pulse sequences is handled by the dedicated generator functions in pulse_pattern.py. This file contains a library of sequence generators for LTP/LTD, retention and endurance protocols. For new measurement

protocols, generator functions should be added to pulse_pattern.py, ensuring all sequence logic remaining modular and maintainable.

A pulse sequence can be defined either by using the generator functions or directly populating four parallel parameter lists: voltages, types, pulse_widths and delays. In these lists, each index corresponds to a single pulse, fully describing its behavior in the sequence. These lists are then passed to PulseSequence.build_relative_pulse_sequence for configuration, which combines them into a list of dictionaries, one dictionary per pulse, each containing all the relevant parameters and a unique identifier.

Before execution, the pulse sequence should be visualized for verification of timing, order and structure of the pulse sequence, avoiding configuration errors. This is done by using the visualize_sequence function found in pulse.py which is based on the generic plot_pulse_sequence function in plotting.py.

In the execution phase, the PulseMeasurement class receives the list of pulse dictionaries, translates them into hardware instructions and executes them on the DUT.

After execution, all raw measurement data and detailed metadata describing the pulse sequence are collected in a standardized MeasurementResult container.

For a detailed overview of the repository organization, see Appendix A.

IV Measurement Workflow

The IV measurement branch follows a similar structure. However, it was implemented independently using voltage staircase sweeps in contrast to the pulsing module. The IVCurveMeasurement class in IV.py manages the configuration and execution for these sweeps, including a dedicated visualization and measurement plotting function. Instead of defining a sequence of pulses, the user specifies the parameters such as voltage range, number of steps and sweep type.

Data Handling

Regardless of measurement type, all raw measurement data and metadata are stored in a MeasurementResult class container. Using the save and load functions in data_io.py, results can be stored as or loaded from HDF5 files for further analysis and reproducibility. The HDF5 format is also used in the existing measurement setup for its flexibility offering a hierarchical structure and capacity for storing very large, multidimensional datasets.

The HDF5 file is organized into two groups: /data and /metadata. The /data group contains all raw measurement arrays for easy access and analysis of the results. The /metadata group stores all experiment parameters and pulse sequence details to ensure full reproducibility.

Originally, only basic parameters were stored (/data). As the project evolved, especially when relative sequencing approach was introduced, the metadata was expanded to include dictionaries for each pulse (write_metadata and read_metadata). This allows any arbitrary pulse measurement to be fully described and reconstructed by matching the raw measurement values to pulse ids.

For a detailed overview of the data hierarchy, see Appendix A.

3.0.4. Example Usage

In order to use the ArC TWO, one must first initialize the board with the correct firmware.

```
arc = connect_arc_two(firmware_path='/home/abaigol/.local/share/
    arc2control/firmware/efm03_20240918.bin') % example path
```

```
Using ArC Two device: 0
Using firmware: /home/abaigol/.local/share/arc2control/firmware/efm03_20240918.bin
Successfully connected to ArC Two board.
```

Once the ArC Two is successfully initialized, we can proceed with using the different modules available. We work with Jupyiter notebooks because the cell-to-cell environment streamlines our workflow, allowing us to instantiate the ArC Two device once and then execute the desired measurement protocols in separate cells. This is also common practice in the current measurement setup at the Brain-Inspired Devices and Circuits Group laboratory for the same reasons. All example code is directly from the entry-point notebook main.ipynb with each snippet representing a separate cell.

IV Measurement

As mentioned earlier, the IVCurveMeasurement class provides the interface for this type of measurement. The user instantiates the IV measurement and specifies the device pins, maximum voltage, number of steps and the sweep type. There are currently two options being provided: full, hysteresis. These configuration options are based on numpy linspace for generating equally spaced voltage ramp segments. For instance, the "full" sweep performs a cycle from 0 $\rightarrow$ max $\rightarrow$ min $\rightarrow$ 0, while "hysteresis" does not start at 0: min $\rightarrow$ max $\rightarrow$ min. For any other sweep type, the user is encouraged to create its own configuration in the IVCurveMeasurement class.

```
iv_handler = IVCurveMeasurement(arc)

row_pin = 18              # High pin
col_pin = 13              # Low pin
max_voltage = 0.5         # Maximum voltage magnitude (V)
steps = 100               # Number of voltage steps
sweep_type = "full"       # Options: "hysteresis", "full",
                          # "positive, "negative"
```

Before executing, one should visualize the sweep first using the dedicated visualize_iv_sweep function. It takes the defined sweep configuration and plots it for verification purposes.

```
fig, ax, voltages = plotting.visualize_iv_sweep(
    max_voltage=max_voltage,
    steps=steps,
    sweep_type=sweep_type,
    title=f"{sweep_type.title()} IV Sweep for Device at pins {
       row_pin}-{col_pin}"
)
```

Now we can execute the measurement, plot it and save the figure as a PNG and the data as a HDF5 file, as described earlier. The user is encouraged to create their own filename structure. By default for IV, data and plot filenames include the sweep type and a full date and time stamp.

```
result = iv_handler.sweep(
    row_pin=row_pin,
    column_pin=col_pin,
    sweep_type=sweep_type,
    voltage_range=(-max_voltage, max_voltage),
    steps=steps
)
iv_handler.plot_iv_curves(result, save_plot=True)
iv_handler.save_results(result, filename, format="h5")
```

Arbitrary Pulse Sequencing

The core strength of this framework is its arbitrary pulsing sequencing capabilities, which allow users to construct any desired pulse protocol by combining individual pulses as building blocks. An arbitrary pulse sequence can be configured by populating a list with the required parameters of each pulse, being the applied voltage, pulse type, pulse width and the delay after each pulse, illustrated in Figure 3.7. This flexibility is extremely powerful, as it enables the construction of any desired protocol, from simple single pulse measurement to possibly complex neuromorphic learning rules.

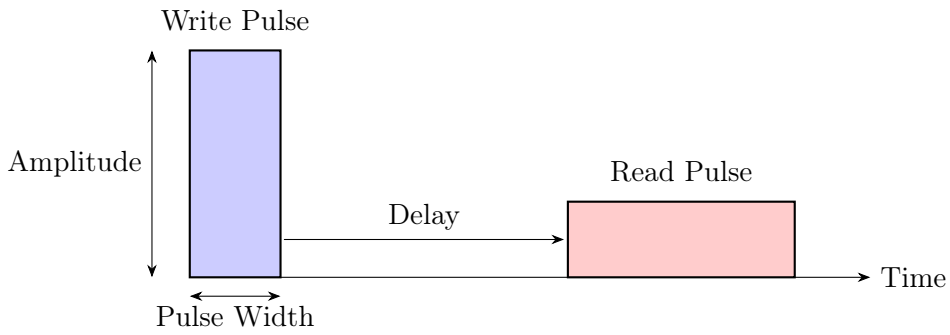

Figure 3.7.: Arbitrary pulse parameters

For example, to define a custom sequence:

```
pulse_handler = PulseMeasurement(arc)

row_pin = 13
col_pin = 18
print(f"Using pins: Row={row_pin}, Column={col_pin}")

voltages = [1,0.3, 0.6, -1.2, 0.1] #in V
types = ['write','read', 'write', 'write', 'read']
pulse_widths = [500_000,None, 50_000, 100_000, None] #in ns
delays = [100000,30000,0,500000,250] #in ns


seq = PulseSequence.build_relative_pulse_sequence(voltages, types,
    pulse_widths, delays)

pulse_handler.visualize_pulse_sequence(seq)
```

Here, each list entry corresponds to a single pulse. The only constraint is the read pulse width, which is fixed by the hardware hence overwritten no matter what value is passed for a read pulse. This is due to setup and integration time to guarantee precise and reliable read-outs. PulseSequence.build_relative_pulse_sequence translates the separate lists into pulse dictionaries which are then visualized (Figure 3.8 using visualize_pulse_sequence.

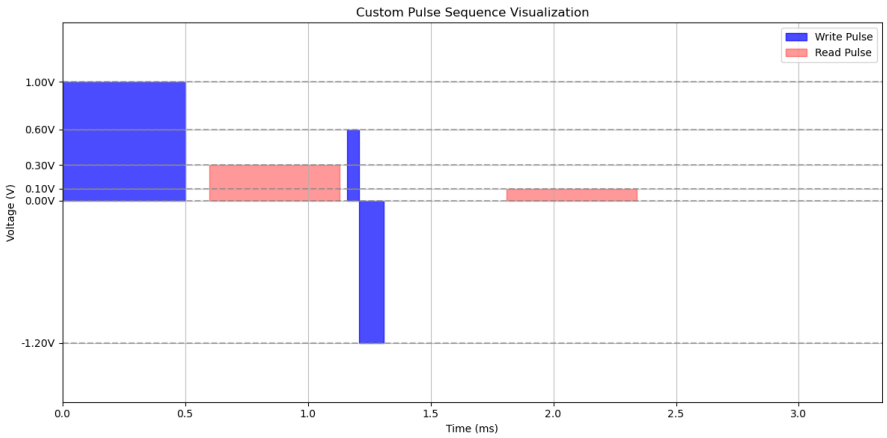

Figure 3.8.: visualization of arbitrary pulse sequence

To run the sequence, plot and save the result:

```
result = pulse_handler.run_relative_sequence(
    row_pin,
    col_pin,
    seq,
    differential=True,
    debug=True)
filename = f"pulse_measurement_{datetime.now().strftime('%Y%m%d_%H%M%S')}"
plotting.plot_measurement_results(result, save_plot=True, filename=filename)
save_measurement_results(result, filename=filename, format="h5")
```

Pin configuration and the configured sequence are passed to run_relative_sequence with the option to choose single sided or differential voltage pulsing. After the sequence is executed, the results are plotted and saved with a user defined file name.

This arbitrary pulse sequencing capability is the foundation for all subsequent measurement protocols. From this point on, only the specific sequence configuration will be shown.

LTP/LTD Measurement

To configure an LTP/LTD experiment to assess gradual switching capabilities in memristive devices, the user specifies the voltages for LTP and LTD pulses, the read voltage, the pulse width and the delays after each pulse type. Additionally, there is an option to add an interphase delay when switching from LTP to LTD pulses. Standard units are volts and nanoseconds if not specified otherwise.

```
# Configure LTP/LTD parameters
ltp_voltages = np.linspace( 1.2, 1.2, 50)
ltd_voltages = np.linspace(-1.6, -1.6, 50)
read_voltage = 0.2
pulse_width_ns = 5_000
read_delay_ns=100_000,
write_delay_ns=50_000,
inter_phase_delay_ns=500_000


# Generate sequence lists
voltages, types, pulse_widths, delays = //
pulse_patterns.create_ltp_ltd_sequence_new(
    read_voltage=read_voltage,
    ltp_voltages=ltp_voltages,
    ltd_voltages=ltd_voltages,
    pulse_width_ns=pulse_width_ns,
    read_delay_ns=read_delay_ns,
    write_delay_ns=write_delay_ns,
    inter_phase_delay_ns=inter_phase_delay_ns
)
```

Retention

For retention measurements, to test how long a device retains its state, the user specifies the set voltage, read voltage, set pulse width and a list of absolute times in seconds at which read operations occur.

```
pulse_handler = PulseMeasurement(arc)
# Configure retention
set_voltage = 1
read_voltage = 0.1
set_pulse_width = 100_000 #in ns
set_pulse_width = 100_000 #in ns
# read_times_sec = [0.0005,0.01,0.02,0.03,0.04,0.05]
read_times_sec = [0.0005] + [0.01 * i for i in range(1, 6)] #in s

# Generate sequence
voltages, types, pulse_widths, delays = pulse_patterns.
    create_retention_sequence(
    set_voltage=set_voltage,
    read_voltage=read_voltage,
    set_pulse_width_ns=set_pulse_width,
    read_times_sec=read_times_sec)
```

Endurance

For endurance protocols, to monitor degradation by repeatedly reprogramming the device with periodic reads in between, the user specifies the number of cycles, set/reset voltage, read voltage, pulse width, delay and read interval. When specifying a read interval, the first and last five cycles are always read.

```
pulse_handler = PulseMeasurement(arc)
# Configure endurance test
num_cycles = 10
set_voltage = 2.5
reset_voltage = -2.5
read_voltage = 0.2
pulse_width_ns=100_000,
std_delay_ns=50000,
read_interval=1

# Generate sequence
voltages, types, pulse_widths, delays = pulse_patterns.
    create_endurance_sequence(
     num_cycles=num_cycles,
     set_voltage=set_voltage,
     reset_voltage=reset_voltage,
     read_voltage=read_voltage,
     pulse_width_ns=pulse_width_ns,
     std_delay_ns=std_delay_ns,
     read_interval=read_interval   # only read in intervals, first
        and last 5 cycles are always read
)
```

Crossbar Measurements

A crossbar class was created for basic operations such as configuring the the channels by loading in a TOML configuration file, weight read out functions on array level and sneak path mitigation channel configurations.

For initializing a crossbar, one can use the configuration loader function in the Crossbar class and either declare the pin assignments manually or load it from a TOML file. These are flexible configuration files in which we define the number of rows and columns and a list each for the actual pin numbers.

```
crossbar = Crossbar.from_config_file(
    instrument=arc,
    config_file="crossbar_config/2x2.toml",  # or manual pin
        assignment
    rows=2,
    cols=2,
```

```
    row_pins=[13, 14],
    col_pins=[17, 18],
    read_voltage=0.1
)
```

Figure 3.9 and 3.10 show example read outs of all cross point conductances and resistances of the dummy crossbar introduced in chapter 3.0.1.

```
conductance_matrix = crossbar.read_conductance_matrix(read_voltage
    =0.1, differential=True)
resistance_matrix = crossbar.read_resistance_matrix(read_voltage
    =0.1, differential=True)
```

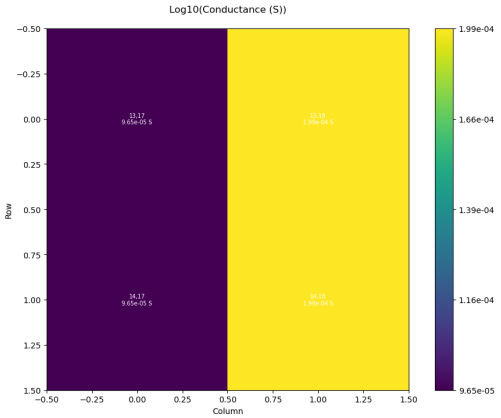

Figure 3.9.: Conductance matrix

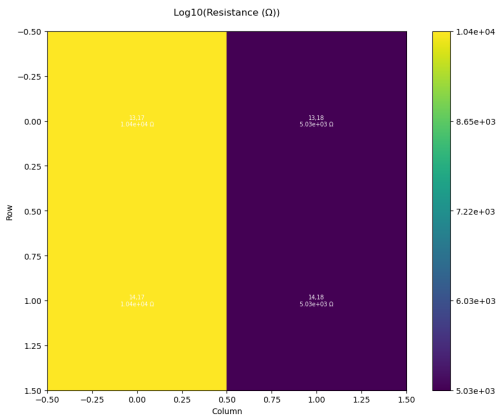

Figure 3.10.: Resistance matrix

In the future, multi-level weight programming schemes on array level can be implemented using the presented arbitrary pulsing framework.

Chapter 4

Results and Discussion

This section presents the experimental results and analysis of the ArC TWO's capabilities for memristive device characterization. First, pulse characterization of the ArC TWO, followed by IV measurements and finally, results are presented for various arbitrary pulsing protocols, such as LTP/LTD, retention and endurance.

4.1. ArcTwo Board Pulse Characterization

The integration of the ArcTwo board being a fully parallel multichannel electronic device testing system makes measurements much easier compared to classical setups using separate waveform generators and oscilloscopes. To achieve this, custom measurement code was developed to generate flexible pulse sequences. Before applying these protocols to real, sensitive devices, This section evaluates the ArC TWO pulse generation capabilities, emphasizing both the advantages of the system and practical challenges encountered during the scope of this project.

Pulse Settling Time

To evaluate the pulse settling behavior, a series of test pulses with varying widths (1,5 and 10μs), from and using differential biasing were programmed, output through the board and captured on an oscilloscope.

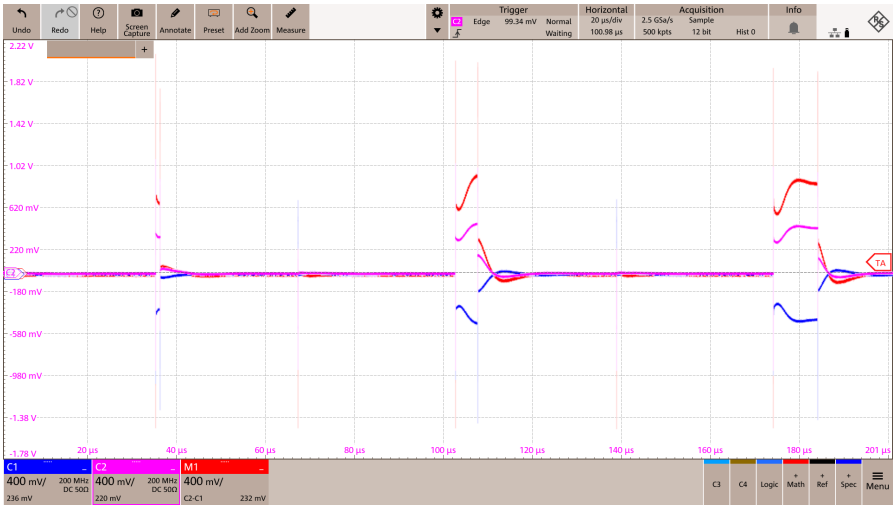

Figure 4.1.: comparison of different pulse widths

When using high-speed pulsing methods (under 500µs), the signal requires up to 5µs to reach and another 5µs to settle at the desired voltage, as observed in Figure 4.1 for the second and third pulses with pulse widths of 5 and 10µs. Larger parasitic capacitances on the ArC and daughterboard would explain this behavior. One way to address this issue was to heuristically compensate for the applied voltage relative to the pulse width. This indicates that precise control over the applied voltage requires careful calibration for each pulse width, highlighting a practical challenge in achieving ideal square pulses when using fast pulsing circuitry of the ArC TWO board. Note that all oscilloscope measurements in this report were performed using 50Ω termination. This provides signal integrity by matching the BNC cables but attenuates the signal due to the inherent voltage divider being created using low impedance termination.

Transient Spikes

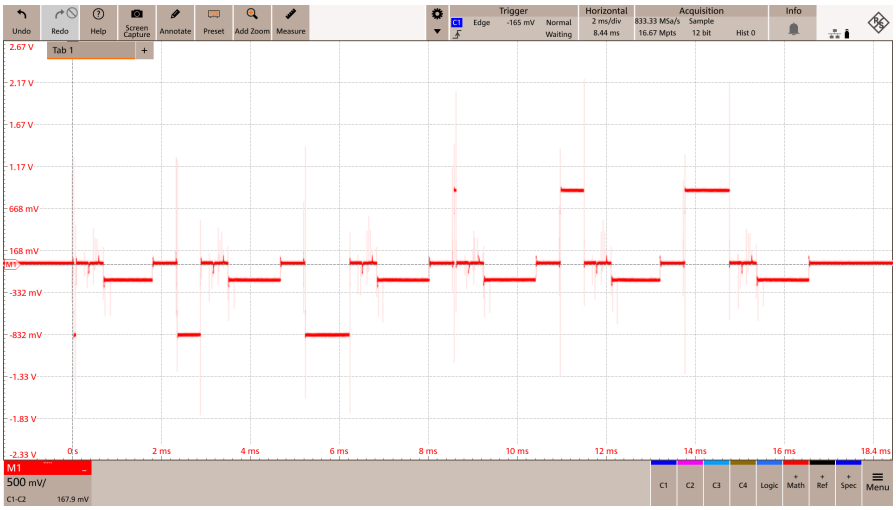

Figure 4.2.: example pulse sequence showing transient spikes

During pulse characterizations using an oscilloscope, transient spikes were observed when using the fast pulsing circuitry. Figure 4.2 shows an LTP/LTD measurement with varying write pulse widths but these observations can be seen for any measurement. These unwanted spikes are problematic for memristive technologies which have very low threshold voltages, and thus could induce unintended switching/programming or even breaking the device. This is further problematic since it is not possible for the user of the ArcTwo board to set an appropriate compliance current for a specific DUT.

According to the developers of the ArcTwo board, these spikes can be explained by two physical mechanisms in the pulse driver circuitry:

Firstly, the spikes in between the two pulses are caused by capacitive coupling during the switching dead time of the driver circuitry. When the FET is turned off before the other is activated, the applied control voltage temporarily couples the output. These type of spikes aren't a big concern since their amplitude is fixed, independent of the pulse voltage and load.

The more problematic spikes are the ones following the rising and falling edges. These are the result of trace inductance between the driving FETs and their decoupling capacitors. Combining this with the very aggressive slew rate causes ringing.

Although the boards developers have installed snubber circuits in all of the drivers, mitigating these effects, claiming to limit these overshoots to 30%. Figure 4.3 shows a full 1µs differential pulse with both transient spikes at the rising and falling edges. For lower voltages these transient spike amplitudes are rather unpredictable and can exceed the desired voltage by over 200% in both directions, resulting in severe overshooting and

undershooting. When zooming into the rising edge of the 1µs pulse (Figure 4.4), the total duration of the over and undershooting is around 45ns.

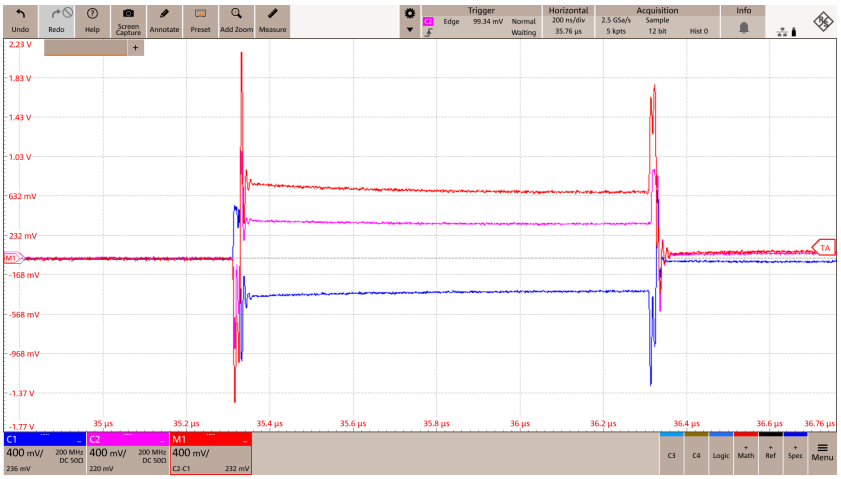

Figure 4.3.: 1µs pulse

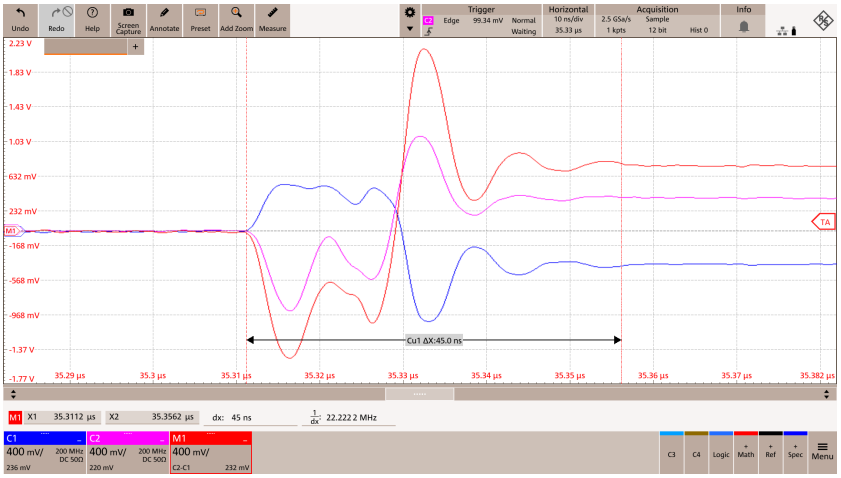

Figure 4.4.: zoomed 1µs pulse at rising edge

Such uncontrolled voltage transients can unintentionally change the state of the device, making it difficult to credit observed changes in conductance only to the intended pulses. This complicates the interpretation of the results, as the the device response might be influenced by these unintended voltage spikes rather than the programmed pulse sequence.

ArC Intruements suggested two solutions for this. Either do some PCB rework by installing an RLC filter or acquire a daughter board with full filters on each channel,

developed by them recently, following similar issues by another customer. These solutions would address this issue by actively dampening high-frequency transients, therefore enhance more reliable device modulation.

4.2. IV measurement

For the actual experiments, IV measurements are presented first. As a reminder, this protocol is a standalone module, not based on the pulsing framework. The initial measurement was performed on the largest VCM device available, which was found to be in its on-state (LRS) prior to the measurement. A full bidirectional voltage sweep was applied using the following configuration:

```
1    max_voltage = -2.5
2    steps = 400
3    sweep_type = "full"  # Options: "hysteresis", "full", "
        positive", "negative"
```

The maximum voltage was set to negative in order to sweep through the negative voltages first, due to the initial on-state of the device.

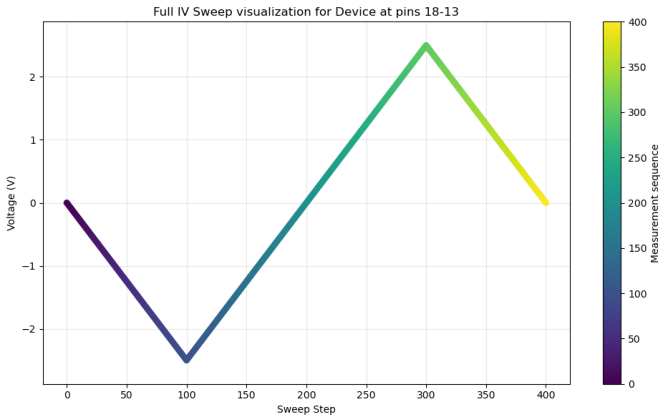

Figure 4.5.: IV sweep visualization

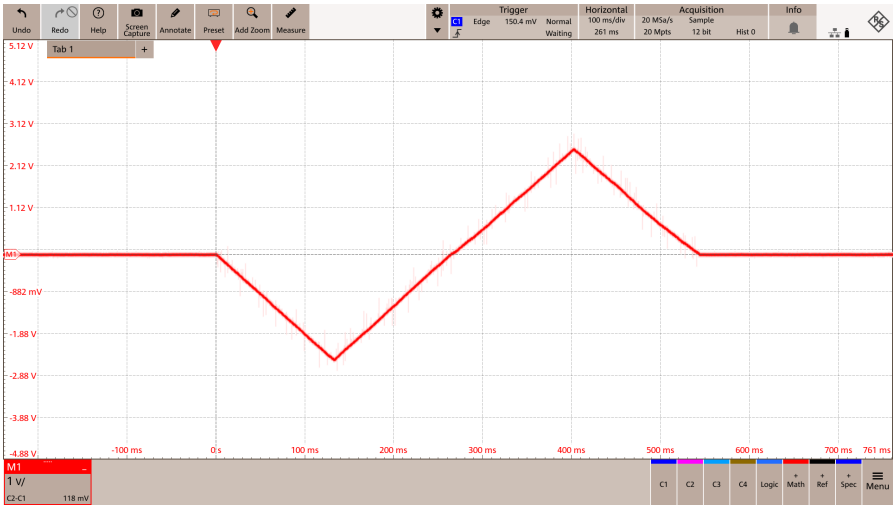

Figure 4.6.: applied voltage sweep measured on the oscilloscope

Figure 4.5 and 4.6 show the visualization of the configuration and the actual applied voltage ramp on the oscilloscope for comparison.

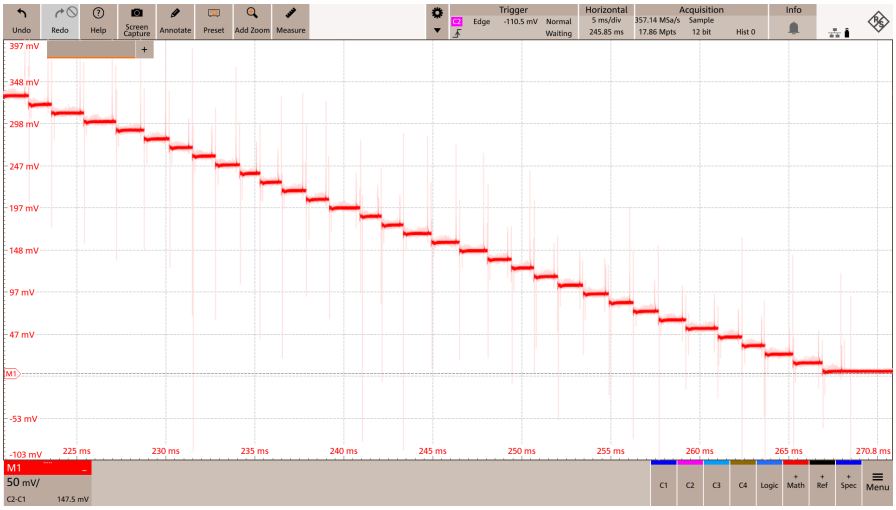

Figure 4.7.: zoomed in on descending ramp in IV

Upon closer inspection of the voltage ramp steps (Figure 4.7), the individual steps are not uniform in duration. This is due to how the read operations are implemented:

```
if v >= 0:
    arc.config_channels([(low_chan, v)], None)
    arc.read_slice_open_deferred([high_chan], True)
else:
```

```
    arc.config_channels([(high_chan, -v)], None)
    arc.read_slice_open_deferred([low_chan], True)
```

It configures the appropriate channel based on the voltage sign and then performs a deferred reading operation for later retrieval of the measurement result from memory. There is no option to configure the integration time of the read operation, hence is fixed by the hardware. While this issue does not compromise the overall switching behavior, it introduces minor asymmetry in the voltage ramp timeline. One way to address this issue is to add an artificial delay after configuring the channel. Although this does not change the integration time, it would produce more consistent timings between steps.

As explained earlier for pulsing, the ramp of these IV sweeps is also suspect to transient spikes at the edges, visible in both Figures 4.6 and 4.7. Their impact on the overall device characterization should be less significant compared to pulsed operations, since the device stays biased for a considerably longer duration during the DC sweep.

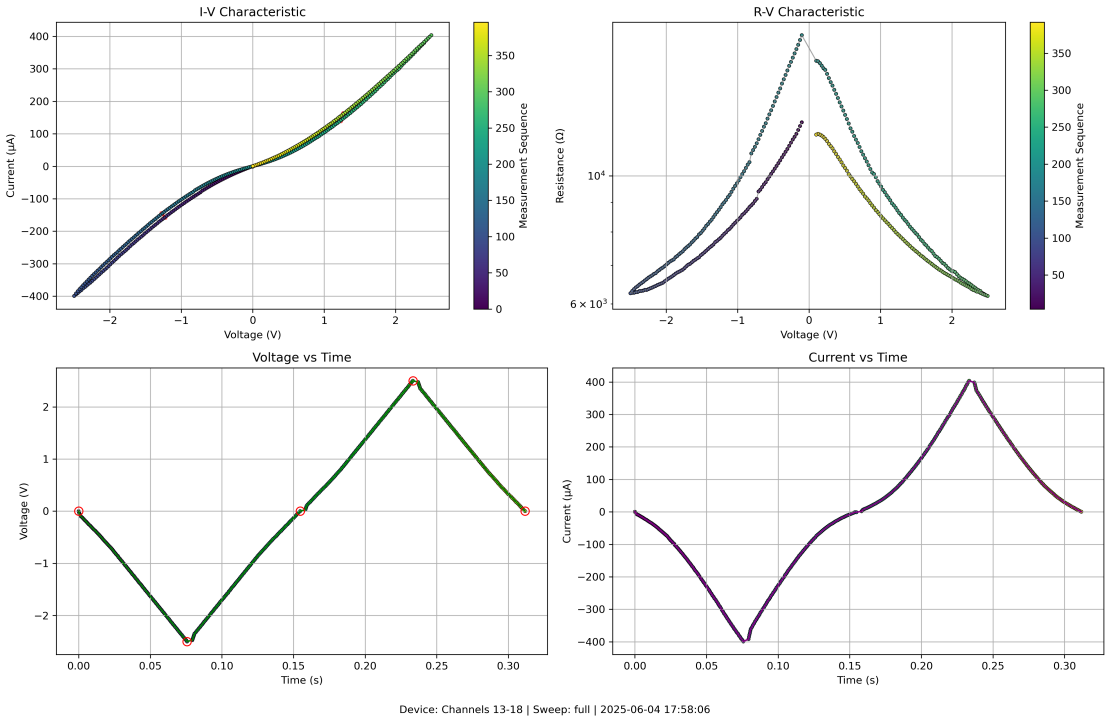

Figure 4.8.: IV measurement using VCM device

The resulting IV curve in Figure 4.8 demonstrates the typical hysteresis behavior for memristive VCM devices, confirming the functionality reproducibility of these voltage sweeps. The device starts in LRS and transitions to HRS when applying the initial negative sweep. The resistance window spans roughly one order of magnitude, suggesting

reliable binary switching. The transition from LRS to HRS occurs around -2V. Maintaining this HRS state when returning to 0V and the preservation of the initial resistance state after a full sweep support the non-volatile characteristic critical for memory applications.

It should be noted that, due to the lack of compliance current capabilities, measurements have to be done with caution to prevent damaging the DUT, especially when reading out at the higher switching threshold voltages and the longer period of time compared to pulse measurements. Note that for this R-V curve, voltage values close to 0V were omitted during plotting, since these calculated resistance values are prone to current noise.

Another measurement was done using a ferroelectric junction with the following configuration:

```
1    max_voltage = 0.5
2    steps = 400
3    sweep_type = "hysteresis"
```

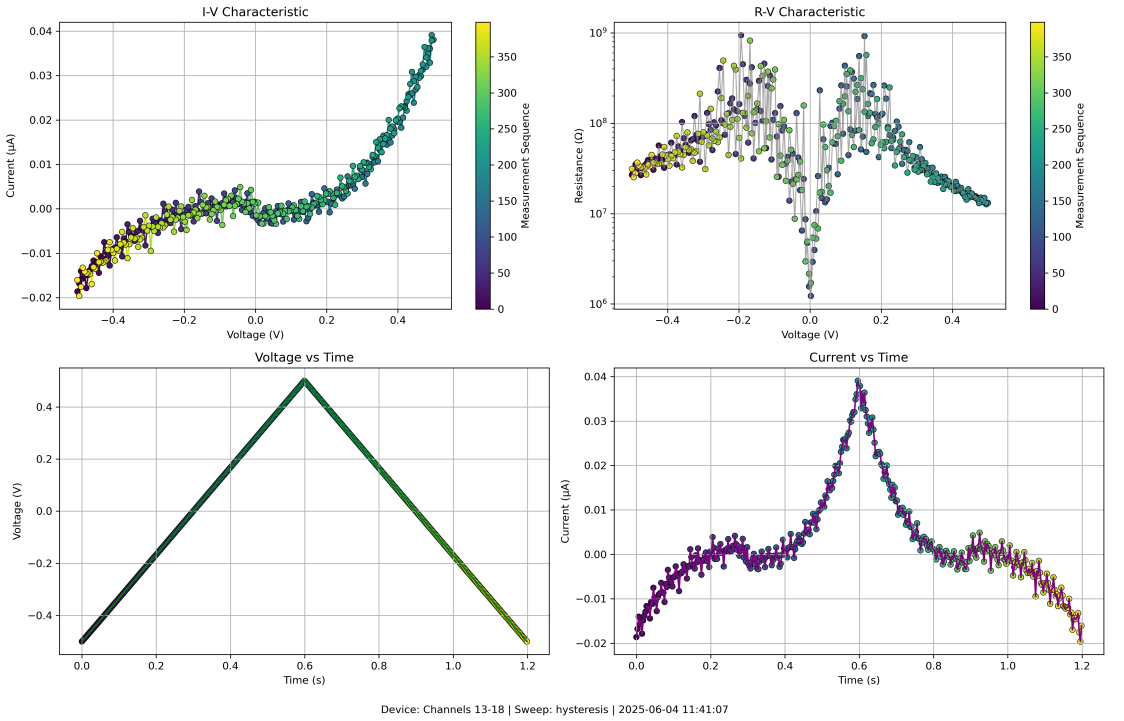

Figure 4.9.: IV measurement using a ferroelectric device

Figure 4.9 shows a very noisy behavior, completely obscuring the hysteresis characteristics if there is any. One major reason is the prevalent noise floor when having such low

currents ranging between 20 and 40 nA. This asymmetry in the current range is consistent with the designed device asymmetry, introduced in the theory section for FTJs. While the measured current range is within the expected off state range, the lack of switching behavior suggest sub-threshold biasing or degradation of the DUT.

4.3. Arbitrary Pulsing

In this section, the capabilities of the developed arbitrary pulsing framework will be looked at. The flexibility of this pulsing framework is crucial for exploring complex memristive phenomena such as synaptic plasticity (e.g STDP [18]) or non-linear device dynamics, often requiring precisely shaped and asynchronous voltage application.

Using the following configuration, each possible transition between write and read pulse operations is shown in a single sequence:

```
1     voltages = [2,1,0.2,0.2,1]
2     types = ['write', 'write', 'read','read','write']
3     pulse_widths = [1_0000,2_0000,None,None,1_000]
4     delays = [0,0,0,0,0]
```

Using the specified parameters, a visualization (Figure 4.10) of the pulse sequence is created before it is executed, as introduced in the methods section. Notice that even though the relative delays are all set to zero in the configuration, the hardware requires some dead or overhead time in between each pulse.

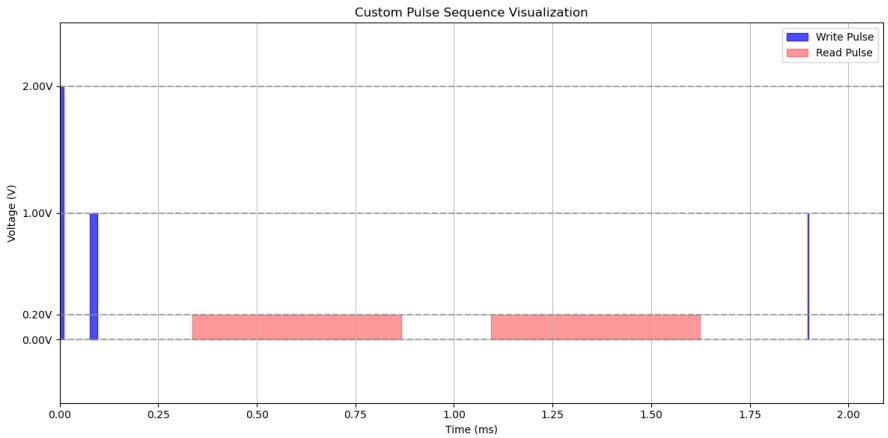

Figure 4.10.: Arbitrary Pulse Sequence Visualization

The ArC TWO does not keep track of time during the execution of commands, thus the visualization serves as a time line verification by incorporating the measured hardware time overhead, but also helps with experimental design, such as intuitive identification of

potential timing overlaps. It ensures temporal integrity before engaging with real devices, without the need of an oscilloscope measurement every single time.

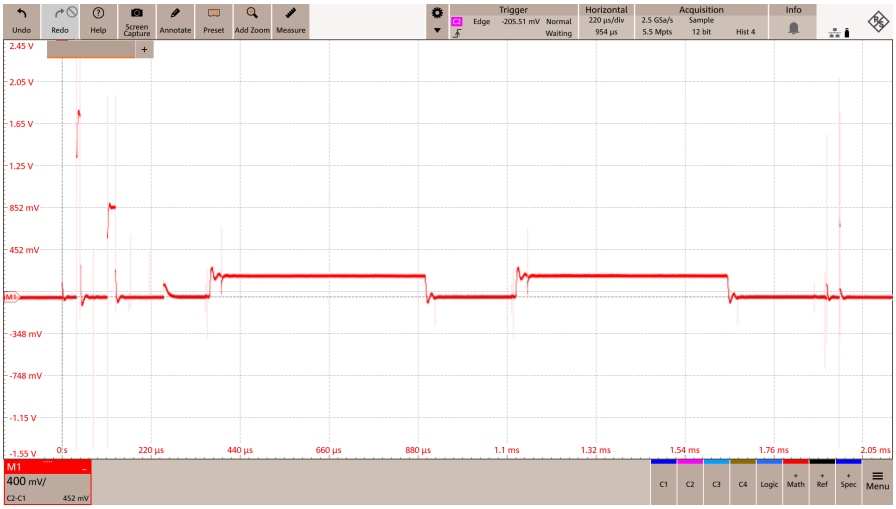

Figure 4.11.: Arbitrary Pulse Sequence on Oscilloscope

When executed on an oscilloscope (Figure 4.11, the resulting wave form closely matches the visualization, validating that the ordering of pulses was preserved and the timing correctly predicted. Transient spikes at each pulse occur as discussed previously. For the spikes, switching between pulse operations is the most prominent during the write-read transition. Another spike phenomena before each write pulse was observed. These are due to a wrong implementation by grounding all channels before executing a write pulse, resulting in sudden discharge of parasitic capacitances. This mistake was only discovered after all measurements were done and is be fixed by removing the unintended grounding instruction. This will require remeasuring and adapting the transition overhead times to keep the timings consistent between visualization and measurement.

4.3.1. LTP/LTD measurement

A VCM device was tested using the implemented LTP/LTD module using the arbitrary pulsing framework. Fixed voltage amplitude pulses of 1.2V for potentiation and -1.6V for depression respectively, were applied, all with fixed 50µs pulse width using differential pulsing. Reading is done at a low voltage of 0.2V. This configuration was chosen to optimize the linearity and dynamic range of the conductance window, based on preliminary sweeps of initial conductance, amplitude, and pulse width.

```
1     # Configure LTP/LTD parameters
2     ltp_voltages = np.linspace( 1.2, 1.2, 50)
3     ltd_voltages = np.linspace(-1.6, -1.6, 50)
4     read_voltage = 0.2
```

```
    pulse_width_ns = 10_000 # write pulse width
    read_delay_ns=100_000,  # delays always after pulse
    write_delay_ns=50_000,
    inter_phase_delay_ns=500_000
```

Figure 4.12 shows the visualization of the configured pulse sequence:

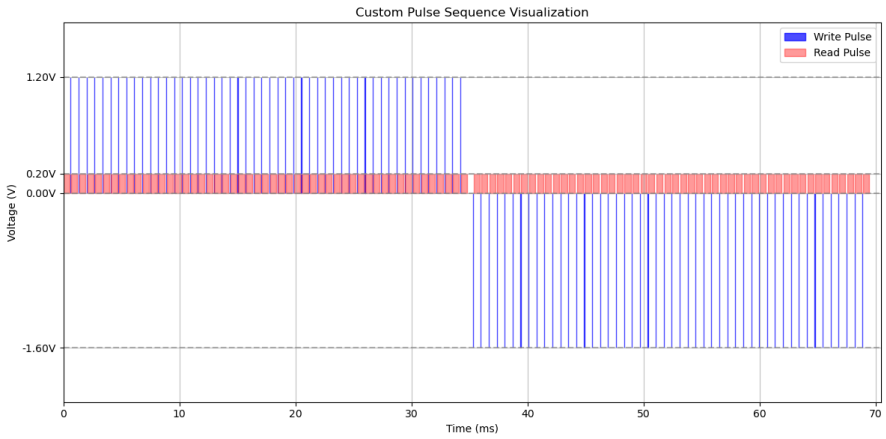

Figure 4.12.: Visualization of LTP/LTD sequence

Figure 4.13 shows the applied pulses on the oscilloscope with less iterations and wider write pulse widths for easier inspection of the reader. This measurement matches the expected pattern: a segment of consecutive positive pulses followed by consecutive negative pulses, all with intermediate read operations. Timing and ordering of the pulses are consistent, including the interphase delay, seen at the transition of positive to negative pulses. All spiking phenomena discussed earlier are present and could potentially affect sensitive devices.

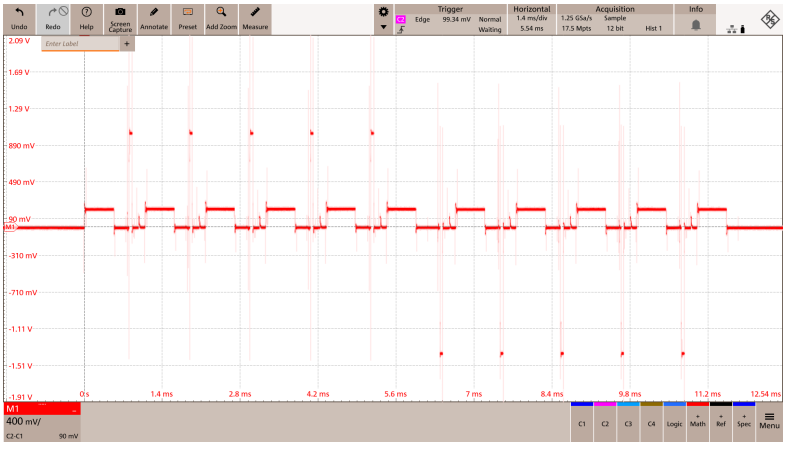

Figure 4.13.: LTP/LTD on oscilloscope

Measurements on an actual VCM device was then performed using the given configuration and the extracted results are presented.

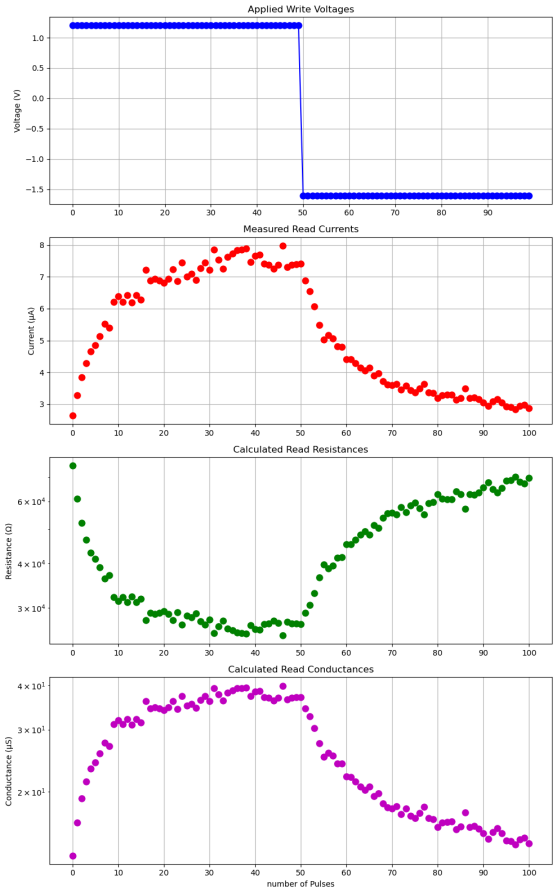

Figure 4.14.: LTP/LTD differential

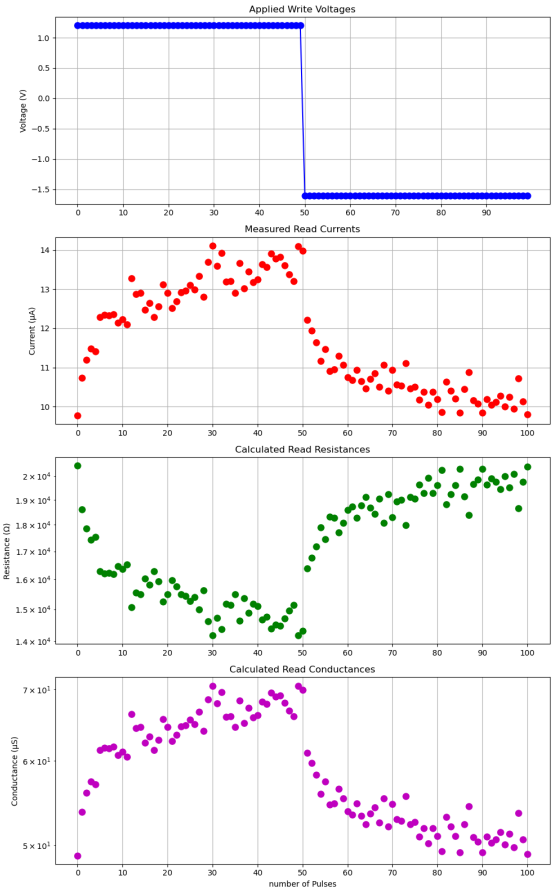

Figure 4.15.: LTP/LTD non-differential

Figure 4.16.: Comparison of LTP/LTD implementations on VCM

Figure 4.14 demonstrates the analog switching capabilities of the VCM device well. The device nearly returns to its initial state, and the conductance/resistance curves show high symmetry with reliable gradual adjustment of the conductance state. This symmetric behavior is critical for neuromorphic applications, where symmetric weight updates are essential in artificial neural networks for effective learning. It also confirms that gradual adjustment of conductance using short pulses works reliably on the ArC TWO board.

A second measurement was performed on the same device using non-differential, or single-sided pulsing. Figure 4.15 shows that the dynamic range of resistive states is narrower compared to the differential case. Although some symmetry is still observed, the resistance updates appear noisier, even though the overall current is higher. This suggests

that field asymmetry in non-differential biasing contributes to less stable switching dynamics, less distinct conductance window and more variation in conductance levels.

That being said, differential biasing becomes impractical when scaling to crossbar arrays. In these structures, wordlines typically carry distinct voltages while all bitlines are tied to a common node. This means that applying symmetric differential voltages to each device is not feasible, since it would require different voltages on the same shared terminal. Therefore, while differential biasing is preferred for individual device testing due to its noise and symmetry advantages, non-differential biasing remains the only practical approach for large-scale crossbar operation, especially when using schemes like V/2 or V/3 to suppress sneak paths currents.

For comparison, an earlier version of the LTP/LTD sequence was also tested on a FTJ device. In this case the parameters were different: the voltage was swept linearly from 0V to ±1.5V with 100 differential pulses each and a fixed pulse width of 5µs. The results are shown in Figure 4.17

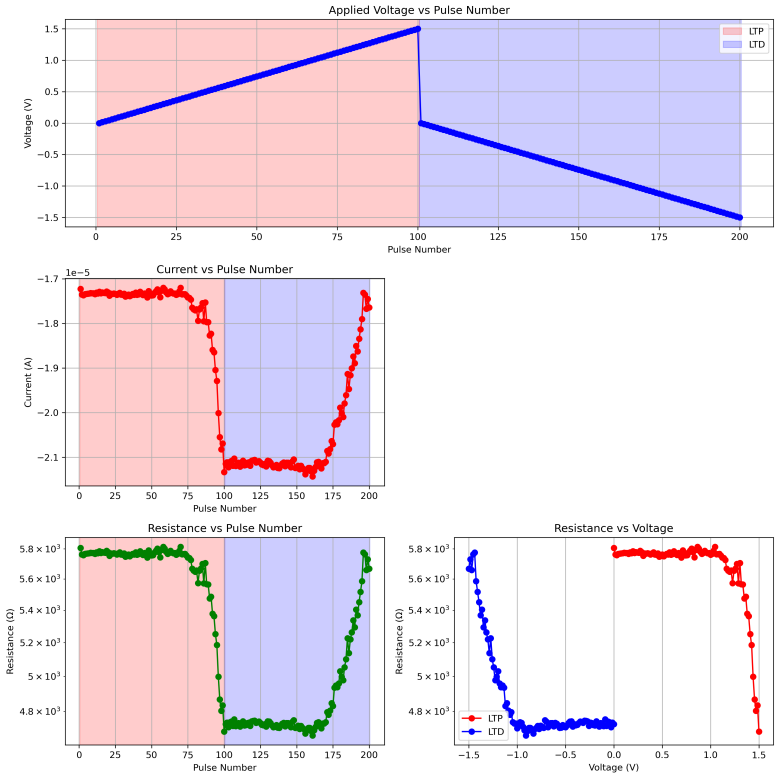

Figure 4.17.: LTP/LTD on FTJ

The FTJ device shows gradual and symmetric switching behavior. Unlike the VCM case, where fixed-amplitude pulses were sufficient to produce gradual conductance changes, the

FTJ device required linearly ramping pulse amplitudes to incrementally change polarization in the ferroelectric.

Ferroelectric switching is governed by a material-specific threshold. Below this threshold, only a few or none of the domains change their polarization direction. When linearly increasing the pulse amplitude, the electric field gradually reaches and overcomes this threshold, enabling controlled and gradual polarization switching by smoother modulation of the tunnel barrier. In contrast, VCM utilizes filament formation through oxygen vacancy movement which relies more on pulse count and duration.

4.3.2. Retention and Endurance

While direct measurements of retention and endurance were not performed on real devices, comprehensive protocols were developed and verified on an oscilloscope. These properties are essential for the practical application of memristive devices in non-volatile memory and neuromorphic computing architectures. Retention describes the ability of a device to maintain the stored resistance value over time while endurance tests investigate device resilience to repeated switching, typically quantified by metrics such as cycles-to-failure (C2F) or time-to-failure (TTF), all metrics important for assessing future commercial viability of memristive applications.

Retention

Retention aims to evaluate the stability of programmed resistance states over a period of time. This is done by programming the device followed by read pulses in regular intervals, monitoring the device. The initial programming pulse is set to 1V with 100µs pulse width while readouts were performed at 0.1V. The first read operation should occur right after its programming to verify the initial state. The reading intervals for this measurement were kept short in the ms regime for easier inspection of the total pulse sequence.

```
1     # Configure retention
2     set_voltage = 1
3     read_voltage = 0.1
4     set_pulse_width = 100_000 #in ns
5     read_times_sec = [0.0005,0.01,0.02,0.03,0.04,0.05]
```

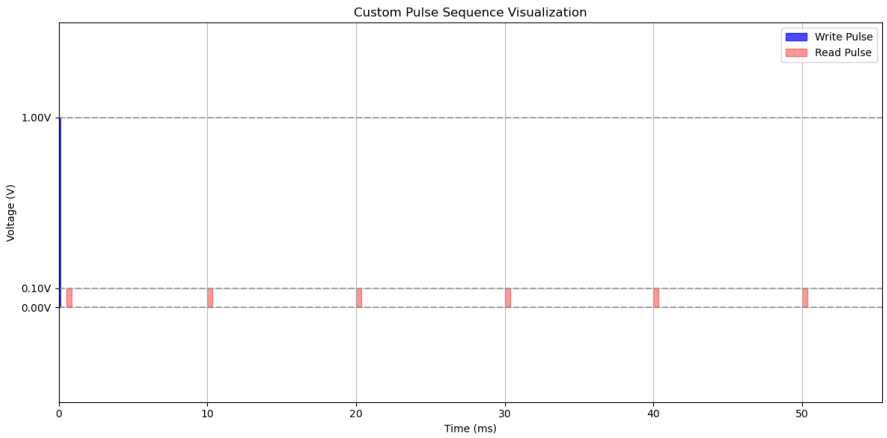

Figure 4.18.: Retention visualization

The visualization in Figure 4.18 shows accurate placement of the read pulses at the intended 10ms second intervals with the initial read right after the programming pulse.

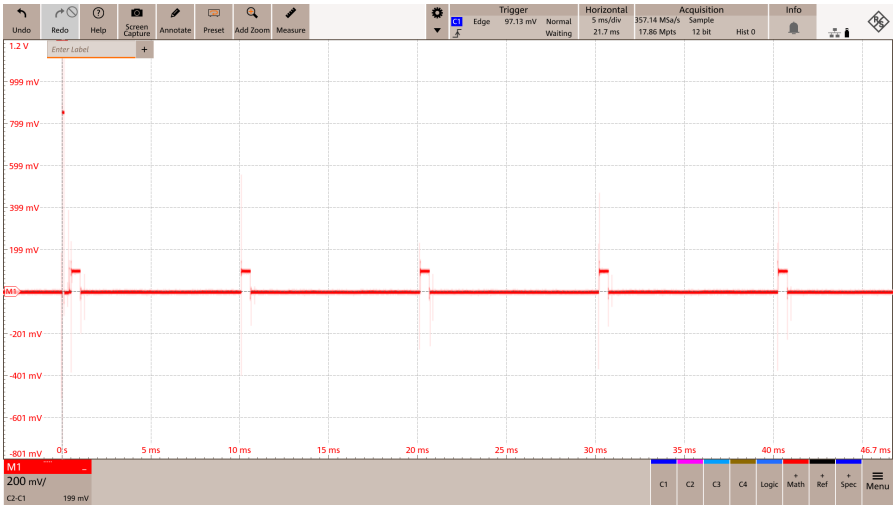

Figure 4.19.: retention test on oscilloscope

A temporary stall in the output waveform of around 3ms was observed during another retention test with the same configuration but 250 number in 100µs intervals of read pulses(Figure 4.20). This gap, occuring after approx. 160 pulses, is caused by the block memory reaching its capacity and stalling execution until a portion of the stored results could be offloaded to the PC. For longer measurements spanning over hours, these stalls should not be a big concern but still have to investigated further in the future and are currently not respected in the visualization tool.

Future work will involve assessing the stability of real devices with this protocol with particular attention to material diffusion and thermal variations, impacting the device state.

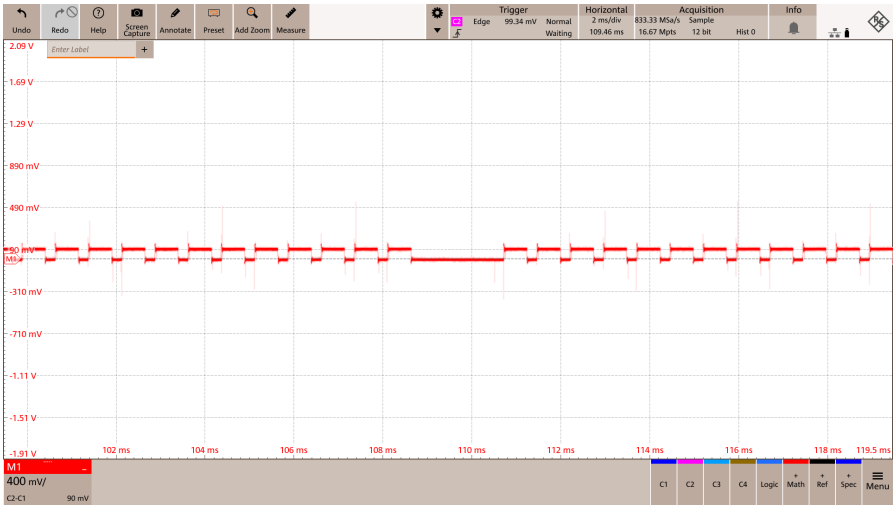

Figure 4.20.: dead time in retention test

Endurance

The endurance protocol is designed to quantify the life span and reliability of a memristive device under repetitive cycling. This involves SET and RESET sequences with interleaved reading pulses in between, monitoring the resistance window over countless cycles until significant degradation or failure occurs. Programming voltages are set to ±2.5V with 100µs pulse width, reading operations are performed at 0.2V and 500µs of delay between each operation with a total cycle count of 10.

```
# Configure endurance test
num_cycles = 10
set_voltage = 2.5
reset_voltage = -2.5
read_voltage = 0.2
pulse_width_ns=100_000,  # in ns
std_delay_ns=500_000, # delay after SET/RESET pulse
read_interval=1  # only read in intervals, first and last part are
    alwyas read
```

Figure 4.21 shows the visualization of the configured retention sequence with a read operation at the start to measure the initial state before the device is programmed.

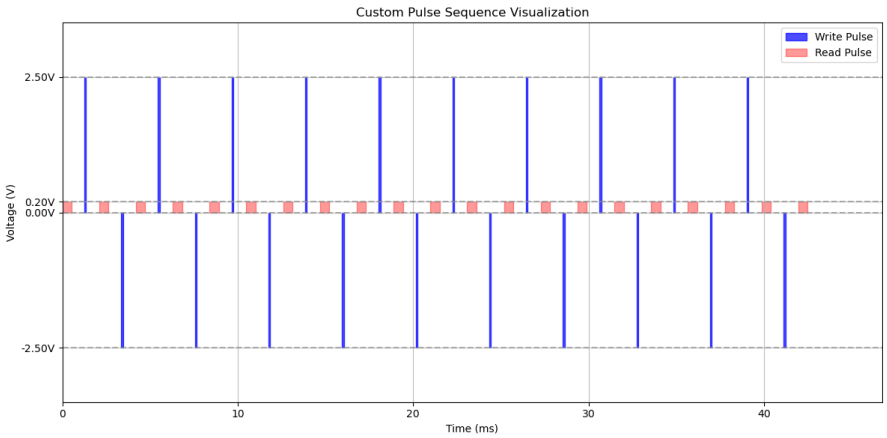

Figure 4.21.: endurance visualization

The resulting measurement on the oscilloscope (Figure 4.22) matches the visualization, confirming intended execution in terms of timing and pulse order and amplitude. Transient spikes are visible, consistent with what was discussed before.

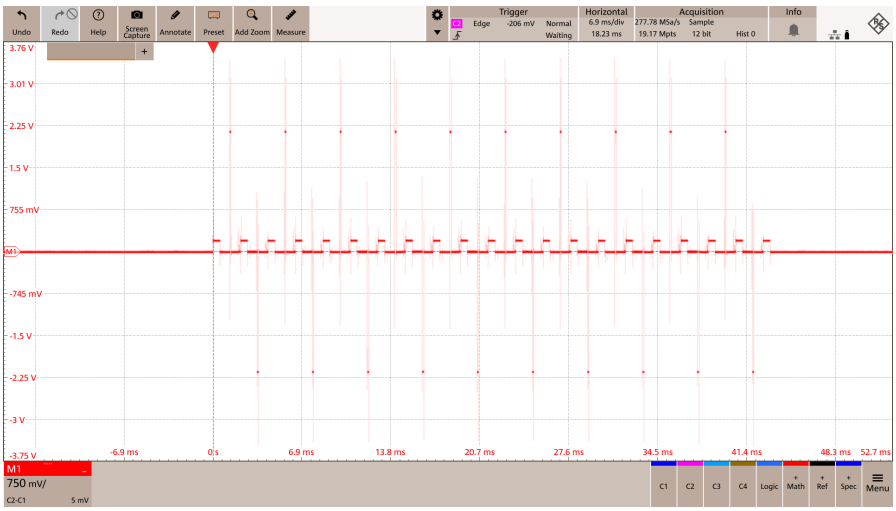

Figure 4.22.: endurance test on oscilloscope

One can also only read after a certain interval (Figure 4.23 and 4.24), minimizing data storage requirements and mitigating the memory buffer problem mentioned in the retention section. The first and last parts are always read for the initial- and final-state assessment. The configuration for each pulse stays the same but the number of cycles was extended to 15 with a reading interval of 5:

```
# Configure endurance test
num_cycles = 15
```

```
set_voltage = 2.5
reset_voltage = -2.5
read_voltage = 0.2
pulse_width_ns=100_000,  # in ns
std_delay_ns=500_000, # delay after SET/RESET pulse
read_interval=5  # only read in intervals, first and last part are
    alwyas read
```

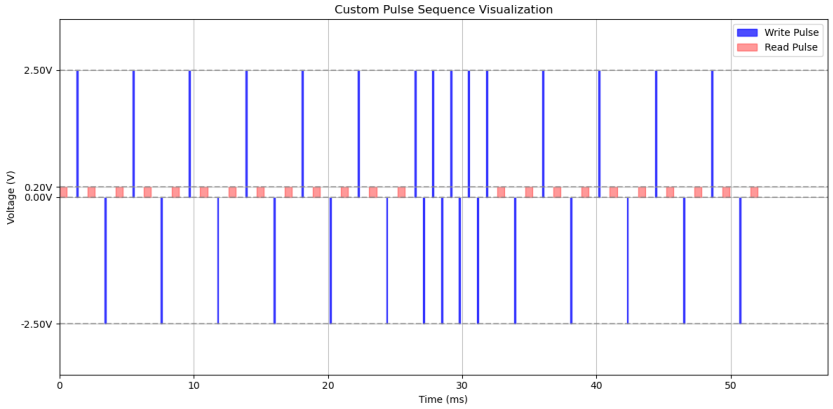

Figure 4.23.: endurance visualization with interval reading

Figures 4.23 and 4.24 for visualization and measurement match in terms of pulse order and timings. In this sequence, read-outs are performed after every pulse for the first five cycles, followed by a block of cycles where read-outs are omitted. At the end, read-outs are again performed after each pulse for the final five cycles to capture the device's final state.

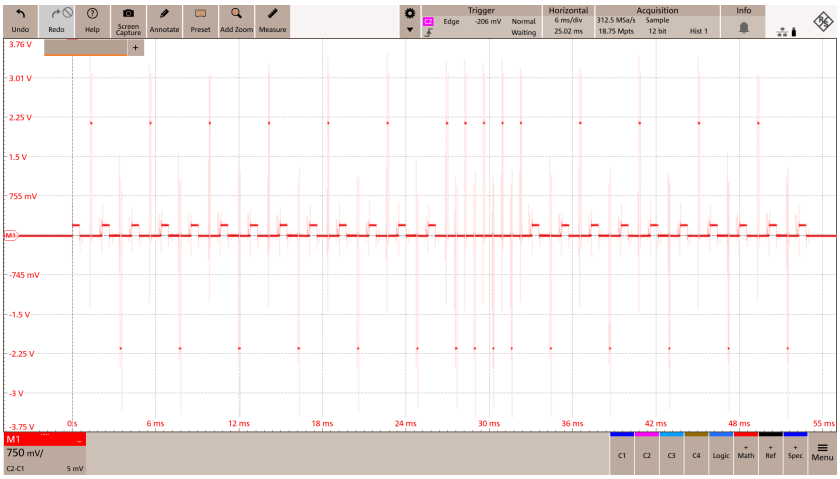

Figure 4.24.: endurance test on oscilloscope with interval reading

Chapter 5

Conclusion and Future Work

The primary goal of this semester project was to develop a flexible and modular framework for characterizing memristive devices and crossbar arrays using the novel ArC TWO platform.

A modular software stack based on the Python wrapper API was implemented, covering essential measurement protocols including IV measurements, Long-Term Potentiation/Depression (LTP/LTD), retention, and endurance. These protocols were successfully integrated with the ArC TWO, allowing a user to configure and execute arbitrary pulse sequences via an intuitive interface using Jupyter notebooks. IV and LTP/LTD measurements were performed on VCM and FTJ devices, confirming correct implementation, demonstrating the suitability of this framework for single device benchmarking.

In addition to single device testing, a crossbar module was implemented. It includes channel configuration, sneak path mitigation strategies and array-level readout operations. Although actual measurements on fabricated crossbars were not possible due to time constraints, functionality was demonstrated using a 2x2 dummy crossbar consisting of potentiometers, mimicking resistive switching. This setup allowed for verification of basic addressing and measuring operations and serves as a proof of concept for future array-level testing.

The project points out some key limitations. The lack of compliance current control poses a significant risk to measuring memristive devices, as they are sensitive to overcurrent. In addition, transient voltage spikes were observed at the falling and rising edges with fast pulses, caused by trace inductances between the driving circuitry and decoupling capacitors. Combined, these limitations currently make safe testing on real devices challenging, making the visualization tool even more important.

For future work, the directions to be taken are evident. First, implementing compliance current control on the circuit level in collaboration with the ArC TWO developers is es-

sential for safe tests on real devices. Second, implementing crossbar-level writing schemes using the proposed biasing schemes would enable actual programming of memristive arrays. Lastly, validating the developed system by testing it on real wire-bonded crossbar arrays will be crucial for benchmarking device performance and further exploration of in-memory computing applications.

In summary, this project successfully established a framework and tool chain for automated and scalable electrical characterization of memristive devices using the Arc TWO board. In spite of certain limitations, the results demonstrate a robust and extendable platform that can hopefully accelerate future research in neuromorphic hardware and crossbar based in-memory computing.

AI Usage Declaration

AI-based Tool	**Use Case**
Github Copilot, Sonnet 3.7	In line code completion and commenting, exploration of ArC repositories
ChatGPT 4o	Citation entry generation for Latex, generation of figure 3.6 and 3.7
Writefull	Overleaf plug in for language feedback and suggestions

Appendix A

Code Structure

A.1. Repository

```
Arc2/
 examples/                      # Jupyter notebooks for usage of modules
    main.ipynb                  # entry-point notebook
 src/                           # Main source code
    analysis/                   # Data analysis and visualization tools
       plotting.py              # Plotting and visualization functions
    instruments/                # Hardware interface and pulse sequence generation
       arc_two.py               # ArC Two device connection and control
       pulse_sequence.py        # PulseSequence class
       pulse_patterns.py        # Predefined pulse pattern generators
    measurement/                # Measurement routines and experiment logic
       base.py                  # Base measurement result classes
       crossbar.py              # Crossbar class
       IV.py                    # IV curve measurement logic
       pulse.py                 # executes PulseSequence object on ArcTwo
    utils/                      # Utility functions and helpers
        data_io.py              # Data saving/loading (e.g., HDF5, config)
        hardware_utils.py       # Hardware-related utility functions
```

A.2. Data Handling

```
root
 data
    voltages     (dataset: float[]) # Applied voltages for each pulse or IV step
    currents     (dataset: float[]) # Measured currents for each pulse or IV step
    resistances  (dataset: float[]) # Calculated resistances for each pulse or IV step
    pulse_widths (dataset: float[]) # Pulse width (ns) for each pulse (0 for IV)
    timestamps   (dataset: float[]) # Timestamp for each pulse or IV step
 metadata         (group)
     For pulse-based measurements:
        low_pin            (attribute: int)         # Low/reference pin used
        high_pin           (attribute: int)         # High/active pin used
        sequence_length    (attribute: int)         # Total number of pulses
        read_pulses        (attribute: int)         # Number of read pulses
        write_pulses       (attribute: int)         # Number of write pulses
        compliance_current (attribute: float)       # Compliance current (A)
        write_voltages     (attribute: list[float]) # List of write pulse voltages
        write_pulse_widths (attribute: list[float]) # List of write pulse widths (ns)
        write_metadata     (attribute: list[dict])  # List of dicts, one per write pulse:
           Each dict contains:
               id           (int)      # Pulse index
               voltage      (float)    # Pulse voltage (V)
               pulse_width  (float)    # Pulse width (ns)
               delay_after  (float)    # Delay after this pulse (ns)
               is_read      (bool)     # False for write pulses
               differential (bool)     # Differential mode used
               type         (str)      # "write"
        read_metadata      (attribute: list[dict])  # List of dicts, one per read pulse:
           Each dict contains:
               id           (int)      # Pulse index
               voltage      (float)    # Pulse voltage (V)
               pulse_width  (float)    # Pulse width (ns)
               delay_after  (float)    # Delay after this pulse (ns)
               is_read      (bool)     # True for read pulses
               differential (bool)     # Differential mode used
               type         (str)      # "read"
     For IV measurements:
         row_pin            (attribute: int)          # High/active pin used
         column_pin         (attribute: int)          # Low/reference pin used
         sweep_type         (attribute: str)          # Sweep pattern ("full", "hysteresis", etc.)
         voltage_range      (attribute: tuple)        # (min, max) voltage range
         steps              (attribute: int)          # Number of steps per segment
         read_delay_ns      (attribute: int)          # Delay between reads (ns)
         compliance_current (attribute: float)        # Compliance current (A)
         reset_voltage      (attribute: float or None)# Optional reset voltage
         measurement_time   (attribute: str)          # Timestamp of measurement
         measurement_type   (attribute: str)          # "iv_curve"
```

Bibliography

[1] J. Backus, "Can programming be liberated from the von neumann style?: A functional style and its algebra of programs," *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, 1978.

[2] G. Indiveri and S.-C. Liu, "Memory and information processing in neuromorphic systems," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1379–1397, 2015.

[3] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[4] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[5] L. O. Chua, "Memristor—the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[6] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature Electronics*, vol. 1, no. 6, pp. 333–343, 2018.

[7] P. Foster, J. Huang, A. Serb, S. Stathopoulos, C. Papavassiliou, and T. Prodromakis, "An fpga-based system for generalised electron devices testing," *Scientific Reports*, vol. 12, no. 1, p. 13912, 2022.

[8] D. Kumar, R. Aluguri, U. Chand, and T. Tseng, "Metal oxide resistive switching memory: Materials, properties, and switching mechanisms," *Ceramics International*, pp. 548–556, 05 2017.

[9] D. F. Falcone, S. Menzel, T. Stecconi, A. La Porta, L. Carraria-Martinotti, B. J. Offrein, and V. Bragaglia, "Physical modeling and design rules of analog conductive metal oxide-hfo2 reram," in *2023 IEEE International Memory Workshop (IMW)*, 2023, pp. 2792–2796.

[10] L. Bégon-Lours, "Neuromorphic Electronics with Oxides: from Materials to AI Hardware," Lecture notes for Course 227-0666-00, 2025, presented as part of the course "Neuromorphic Electronics with Oxides".

[11] L. Bégon-Lours, S. Slesazeck, D. F. Falcone, V. Havel, R. Hamming-Green, M. M. Fernandez, E. Morabito, T. Mikolajick, and B. J. Offrein, "Back-end-of-line integration of synaptic weights using hfo_2/zro_2 nanolaminates," *Advanced Electronic Materials*, vol. 10, no. 5, p. 2300649, 2024, open Access under CC BY 4.0. [Online]. Available: https://doi.org/10.1002/aelm.202300649

[12] S. Stathopoulos, L. Michalas, A. Khiat, A. Serb, and T. Prodromakis, "An electrical characterisation methodology for benchmarking memristive device technologies," *Scientific Reports*, vol. 9, no. 1, p. 19412, 2019. [Online]. Available: https://doi.org/10.1038/s41598-019-55322-4

[13] A. Adeyemo, A. Jabir, and J. Mathew, "Minimising impact of wire resistance in low-power crossbar array write scheme," *Open Access Journal (unlisted)*, Dec. 2017, accepted: July 5, 2017. [Online]. Available: https://radar.brookes.ac.uk/radar/file/d3eb8a21-65ec-4097-aef3-038820d6cd89/1/fulltext-A.pdf

[14] H. Li, S. Wang, X. Zhang, W. Wang, R. Yang, Z. Sun, W. Feng, P. Lin, Z. Wang, L. Sun, and Y. Yao, "Memristive crossbar arrays for storage and computing applications," *Advanced Intelligent Systems*, vol. 3, no. 9, July 2021, licensed under CC BY 4.0 (https://creativecommons.org/licenses/by/4.0/). [Online]. Available: https://www.researchgate.net/publication/352958012_Memristive_Crossbar_Arrays_for_Storage_and_Computing_Applications

[15] ARC Instruments, "ARC TWO Board - Product Page," https://www.arc-instruments.co.uk/products/arc-two/, 2024, accessed: 06-06-2025.

[16] ArC Instruments, "System specifications," https://files.arc-instruments.co.uk/documents/arc2-general/specs.html, ArC Instruments, 2025, accessed: 19 June 2025.

[17] ARC Instruments, "The ARC TWO Platform - Overview," https://files.arc-instruments.co.uk/documents/arc2-general/overview.html, 2025, accessed: 03-06-2025.

[18] G.-q. Bi and M.-m. Poo, "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of Neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998.

[19] D. V. Christensen, R. Dittmann, B. Linares-Barranco, A. Sebastian, M. Le Gallo, A. Redaelli, S. Slesazeck, T. Mikolajick, S. Spiga, S. Menzel, I. Valov, G. Milano, C. Ricciardi, S. Liang, F. Miao, M. Lanza, T. J. Quill, S. T. Keene, A. Salleo, J. Grollier, D. Marković, and N. ... Pryds, "2021 roadmap on neuromorphic computing and engineering," *Neuromorphic Computing and Engineering*, vol. 1, no. 2, p. 022501, 2021.