FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Lehrstuhl für Informatik 3

Rechnerarchitektur

Markus Fritscher

Applicability of RRAM cells for mixed signal neural network inference ASICs

Masterarbeit im Fach Informatik

14. Mai 2022

Please cite as:
Markus Fritscher, “Applicability of RRAM cells for mixed signal neural network inference ASICs,” Master’s Thesis (Masterarbeit), University of Erlangen, Dept. of Computer Science, May 2022.

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Friedrich-Alexander-Universität Erlangen-Nürnberg
Department Informatik
Rechnerarchitektur

Martensstr. 3 · 91058 Erlangen · Germany

www3.cs.fau.de

Applicability of RRAM cells for mixed signal neural network inference ASICs

Masterarbeit im Fach Informatik

vorgelegt von

Markus Fritscher

geb. am 13. März 1990
in Schweinfurt

angefertigt am

Lehrstuhl für Informatik 3
Rechnerarchitektur

Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer:	**Prof. Dr.-Ing. Dietmar Fey**
Betreuender Hochschullehrer:	**Prof. Dr.-Ing. Dietmar Fey**
Beginn der Arbeit:	**1. Dezember 2021**
Abgabe der Arbeit:	**14. Mai 2022**

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.
Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Markus Fritscher)
Erlangen, 14. Mai 2022

Contents

Chapter 1

Introduction

Neural networks have become ubiquitous in the last few years, with applications ranging from autonomous driving to playing games like go. Subsequently, both frameworks for the development of these networks (like Tensorflow or Keras) and hardware for the efficient training and inference of those networks on given data have been developed (like GPUs or TPUs).

Unfortunately performing network inference on portable edge devices still poses problems since space restrictions prevent the user from installing a big battery.

So-called RRAM (Resistive random-access memory) devices might help mitigate this issue since they can be powered off without loosing this data. Recent research has shown that significant power savings can be achieved when storing an ECG network's weight on RRAM devices (Lo3ML). However this technology still is in its early stages and poses some issues like not being a reliable means of storage: A weight which was intended to be a 1 might be read as a 2, subsequently inducing an error into the calculations.

I have already done research in this area, investigating the actual impact of device variations on the accuracy of a given network [1][2]. During this thesis I would like to extend on this work, answering three questions:

- Can we enhance neural networks in such a way that they can cope with the issues of RRAM devices?
- How can we automatically derive device models from measurement data?
- How can we integrate those devices into an ASIC toolflow?

Different neural networks might react differently to device variations, as shown in fig. 1.1. While the network performing on MNIST data seems to cope well with device variations the accuracy shown by the network performing on ECG data quickly deteriorates when device variation becomes larger.

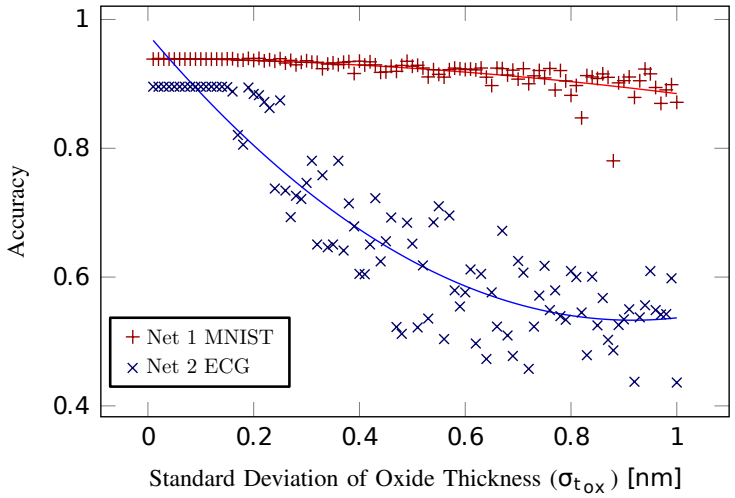


Figure 1.1 – Comparing the effects of device variation on two different networks. The authors picked one specific device parameter, namely the oxide thickness, and evaluated the impact of different spreads on the neural network accuracy. I created this figure as part of my work as an author of [2]

1.1 Motivation

In order to evaluate the applicability of RRAM devices for a given application the researcher has to evaluate the effects of device-level variations on system level accuracies. This usually entails doing a full-stack simulation, embedding a number of layers (Verilog-A, Spice, VHDL, a given Neural Network framework on top); Unfortunately doing a simulation in this manner might be too slow for a meaningful investigation of large systems.

Additionally, currently available RRAM models might not resemble an actual devices behaviour very well since they are fit to very specific scenarios. This poses problems when trying to investigate variance since most models will model this with a mere normal distribution, reducing the benefit of using a physical model in the first place.

A model which is automatically created from raw measurement data on actual devices might help since it properly models variation effects. When this is paired with a circuit parametrization and tightly integrated into a neural network framework this yields a simulation framework which is not only faster but also more accurate than recent approaches. This would enable us to evaluate rather large networks, embedding millions of weights.

While other approaches at modelling - like SystemC - exist, it seems a lot more reasonable to move this modelling straight into the neural network framework as one can then benefit from the huge speedups gained by using GPUs for computation.

Additionally, such a fast model allows for the integration of the model into the very training process of a neural network, enabling the network to optimize for resilience. Recent research suggests that an accurate model is required for this to work.

1.2 Scope of Work

This thesis covers three major contributions:

- Firstly, I want to optimize the simulation framework. Up until now we used the Stanford PDU Verilog-A model, fitted to the IHP devices by John Reuben. While this serves as a good starting point it might not be suitable to investigate the effects of device variations since it is not closely fitted to this part of the data.

  I will - together with IHP - develop models which are tightly fitted to the measurement data created for their memristive devices. This will result in both a Verilog-A model and a model which can be tightly integrated into a Deep-learning framework. The latter will enable me to evaluate the concept of "Fault Aware Training" for large networks.

- I intend to evaluate a number of mitigations for large (at least 100.000 weights) networks. The networks themselves will mostly be provided by Fraunhofer, my work lies within applying both different quantizations and mitigations. In order to do this I will integrate the model created earlier into our adaption of the "brevitas" framework [3]. The mitigation I want to focus on is the so-called fault-aware training as it strongly depends on the accurate modelling of device variance, which I would expect to be a strength of this approach.

- Lastly I will evaluate the possible integration within the ASIC toolflow. I will evaluate whether a purely digital library can be created from the analog cell which has been used within the chip design created by Stefan Pechman within the Lo3ML project.

Setting up a simulation environment will entail running statistics on the measurement data, connecting a mixed signal ADC circuit, parametrizing both and combining those into one coherent model. The major design goal lies within fully automating this process, allowing for the easy creation of models for other devices. The second major goal is being able to use this model transparently within a neural network framework, ideally the network designer does not need to know about the specifics of ASIC design or RRAM devices.

1.3 Related work

Significant effort has been put into the creation of accurate models of memristive devices [4], [5] and the fitting of existent models to real devices [6].

Researchers have also looked into several methods to both train and simulate generic memristive neural networks [7] [8]. Others have looked into specific architectures such as memristive spiking neural networks [9], memory blocks intended for the usage in neural networks [10] or matrix vector multiplication blocks [11].

There have been various attempts at rendering neural networks resilient against device failures, both specific for RRAM devices [2] [12] [13] and for other applications such as providing a framework for the development of fpga systems which implement radiation-hard neural networks [14]. Researchers have also looked into whether reducing the number of states stored within a single RRAM cell might actually increase the overall performance [15].

Unfortunately none of the before-mentioned works cover the entire flow from the automated derivation of device models to the fast simulation of both training and inference of neural networks within a NN framework running on GPUs.

1.4 Results

TBD

1.5 Outline

This thesis is structured as follows:

This section has provided a brief overview into the goals and methods of this work. The next Section will elaborate on the fundamentals required for this work, such as RRAM devices, simulation environments and neural networks. The methodology section will describe the measurement setup and the methodology used for the automated derivation of models and digital standard cells. Subsequently, the results section will describe the results and effectiveness of the described approach.

Chapter 2

Concept

2.1 Simulation framework and model - 50%

Firstly, I want to optimize the simulation framework. Up until now we used the Stanford PDU Verilog-A model, fitted to the IHP devices by John Reuben. While this serves as a good starting point it might not be suitable to investigate the effects of device variations since it is not closely fitted to this part of the data.

I will - together with IHP - develop models which are tightly fitted to the measurement data created for their memristive devices. This will result in both a Verilog-A model and a model which can be tightly integrated into a Deep-learning framework. The latter will enable me to evaluate the concept of "Fault Aware Training" for large networks.

This will entail running statistics on the measurement data, connecting a mixed signal ADC circuit, parametrizing both and combining those into one coherent model. The major design goal lies within fully automating this process, allowing for the easy creation of models for other devices. The second major goal is being able to use this model transparently within a neural network framework, ideally the network designer does not need to know about the specifics of ASIC design or RRAM devices.

2.2 Mitigations - 30%

I intend to evaluate a number of mitigations for large (at least 100.000 weights) networks. The networks themselves will mostly be provided by Fraunhofer, my work lies within applying both different quantizations and mitigations. In order to do this I will integrate the model created earlier into our adaption of the "brevitas" framework [3].

2.3 Integration within the ASIC toolflow - 20%

I will evaluate whether a purely digital library can be created from the analog cell which has been used within the chip design created by Stefan Pechman within the Lo3ML project.

Chapter 3

Fundamentals

3.1 RRAM

3.1.1 theory

- chua
- filament growth, relation to resistance
- store information, up to n bit of information

3.1.2 current state of research, device variations

- ihps depiction of 250nm tech performance
- we need sophisticated writing schemes (-> ispva)
- variation is quite large -> graph measurements (?)

3.1.3 modelling

- modelling is nontrivial, odes involved
- quite a large range of models (see modelling overview paper john linked me) -> MD <-> behavioral/physical Verilog-A models
- most models have issues properly modelling variance, which is one of the reasons for this approach

3.2 Ternary gates

Basic logic gates such as *AND*, *OR*, *NAND* or inverters serve as the basic blocks when designing digital integrated circuits. Memristors might be beneficial when constructing novel logic gates since their inherent multi-state capability allows for the chip-area-efficient implementation of not only binary but also ternary logic. This extends the well-known binary logic (consisting of the states $\{0,1\}$ using a third state, leading to either balanced $\{-1,0,1\}$ or unbalanced ternary logic $\{0,1,2\}$. A truth table for different unbalanced logic operations can be seen in table 3.1. While some ternary operations such as

$$TAND(0,2) \Rightarrow 0 \tag{3.1}$$

$$TAND(2,2) \Rightarrow 2 \tag{3.2}$$

are rather straightforward other operations such as

$$TAND(2,1) \Rightarrow 1 \tag{3.3}$$

$$TNAND(2,1) \Rightarrow 1 \tag{3.4}$$

might be confusing - an inverted *AND* operation seems to yield the same operation as a plain *AND* operation in this special case. This is a result of ternary logic providing more than a simple *yes* or *no* answer to a question.

vin1	**vin2**	**TAND**	**TOR**	**TNAND**	**TNOR**	**TXOR**	**TXNOR**
0	*0*	*0*	*0*	*2*	*2*	*0*	*2*
0	*1*	*0*	*1*	*2*	*1*	*1*	*1*
0	*2*	*0*	*2*	*2*	*0*	*2*	*0*
1	*0*	*0*	*1*	*2*	*1*	*1*	*1*
1	*1*	*1*	*1*	*1*	*1*	*1*	*1*
1	*2*	*1*	*2*	*1*	*0*	*1*	*1*
2	*0*	*0*	*2*	*2*	*0*	*2*	*0*
2	*1*	*1*	*2*	*1*	*0*	*1*	*1*
2	*2*	*2*	*2*	*0*	*0*	*0*	*2*
input signal		output for each individual logic gate					

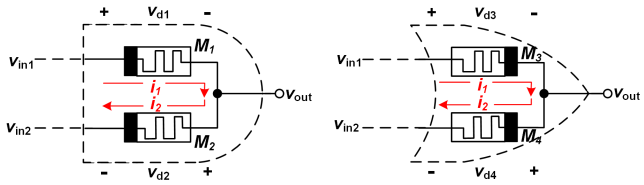
Table 3.1 – Performing different ternary logic operations for the inputs V_{in_1} and V_{in_2} leads to this truth table. Proposed in [16]

This leads to several implications, traditional hardware description languages and tools cannot cope with multivalued logic. Dhande et al. have proposed to use the “high impedance” Z state defined within VHDL as a possible workaround [17]. Unfortunately these proposed constructs cannot be synthesized directly into logic

using actual ternary gates, it serves as a mere simulation environment. Subsequently - as of now - those logic circuits have to be designed manually.

The mapping of logic states into VHDL signals as implemented by Dhande et al. is shown in Table 3.2.

Wang et al. have proposed a concept for the implementation of ternary gates embedding RRAMs in [16]. Two gates, namely a *TAND* and a *TOR* gate are shown in fig. 3.1.



(a) ternary and **(b)** ternary or

Figure 3.1

They consist of two memristors which are connected in series (*TAND*) or in anti-series(*TOR*) in order to implement either a ternary and or a ternary or gate. The novelty about this approach lies within its simplicity - no CMOS components such as transistors are required, the RRAM terminals serve both as inputs and output of the logic gate.

Their simulation results for the proposed *TAND* and *TNOR* gates can be seen in fig. 3.2.

However, actual devices might not match simulation results, especially since most models are optimized for storage instead of logic operations. As part of this thesis I will try to implement these logic gates using IHP technology (see section 5.3).

Wang et al.	Dhande et al
0	0
1	Z
2	1

Table 3.2 – Mapping of the states proposed by Wang et al. to the states as proposed by Dhande et al. This allows for the simulation of ternary logic within a digital VHDL environment without the need for defining new datatypes.

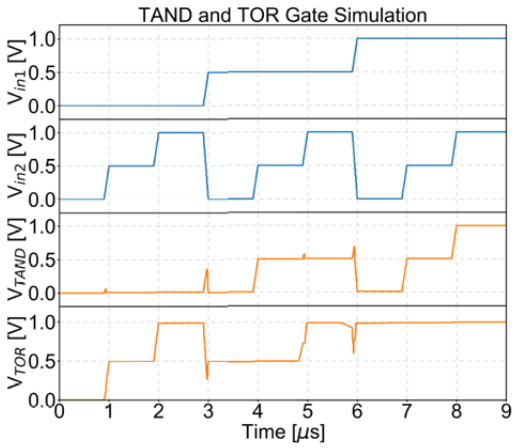


Figure 3.2 – Simulation results for the gates shown in 3.1. The results which are calculated by the memristive gates seem to match the theoretical results shown in table 3.1 well. Figure taken from [16].

3.3 Neural networks

Neural networks are ubiquitous, with applications ranging from autonomous driving to detecting cancer. A survey of relevant developments has been published by Liu et al. [18].

A simple example network is shown in fig. 3.3. Three inputs, namely $x_{1,2,3}$ are provided as inputs. These inputs are multiplied by individual weights (one per arrow), normalized and stored within the brown circles representing the hidden layer. Subsequently, they are multiplied by another set of weights and stored within the two red circles, representing the output layer. These resemble the outputs $y_{1,2}$.

The process which derives suitable weights for a given task is called *training* of a neural network.

3.3.1 Concept

Stacking multiple hidden layers has coined the term *Deep Learning* [19]. Each of the layers provides a different representation of the inputs, yielding the possibility to perform on complex tasks. Concepts such as backpropagation [20] have helped with efficiently training networks.

Famous architectures have introduced different concepts; LeNet-5 (presented by LeCun et al) [21] has introduced the concept of convolutional layers, outperforming competitors on an image classification task. AlexNet (presented by Krizhevsky et al.) [22] introduced both the consecutive stacking of convolutional layers and the

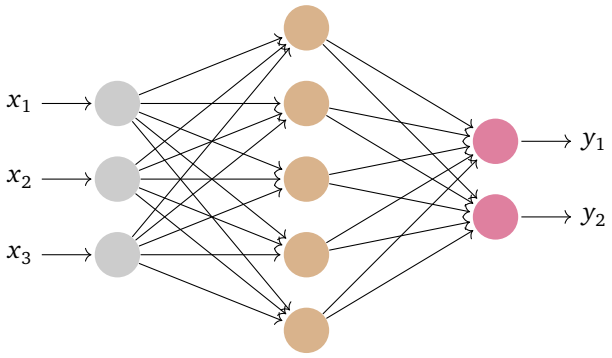


Figure 3.3 – Simple neural network consisting of an input layer (gray) taking three inputs $x_{1,2,3}$, providing an internal representation within a single hidden layer (brown) and providing the outputs $y_{1,2}$ at the output layer (red).

usage of GPUs and has led to the wide-spread adoption of GPUs for the training procedure of neural networks.

Enlarging the network has posed problems regarding overfitting and exploding/vanishing gradients. Researchers have attempted to mitigate this by either adding Residual blocks (which introduce skip-connections during training) [23] or adding Inception cells (enabling the training procedure to determine which filter size is the most appropriate). The latter has resulted in the famous GoogleNet architecture [24]. This network combines multiple different layers (Convolution, Max pooling, Fully Connected, Softmax, ...) into one coherent network which implements the beforemetioned Inception cells.

3.3.2 Quantization

Usually neural networks use single precision IEEE-754 float representation (32 Bit) as their weight storage. However - as described earlier - current RRAM technology only allows for up to 6.5 bit of storage within an individual storage element. Subsequently one has to quantize those weights onto a valuation which can be stored within RRAM devices. This might lead to severe loss in accuracy and require the network to be trained again, taking quantization into account.

There are three major components which can be quantized, namely

- input data
- activations
- weights

all of which result in different architecture considerations.

The concept of having configurable quantizations is not unique to neural networks embedding RRAM. Recent microcontrollers such as the Maxim MAX78000 come with a Neural Network Accelerator unit (see Fig. 3.4) which embeds 448kB of weight storage which can be split upon 1,2,4 or 8-Bit-weights. While dropping to 1 bit weights might seem counter-intuitive it allows for the implementation of significantly larger networks, using up to 3.456 Million weights as opposed to 0.448 Million 8 bit weights, ultimately implementing a vastly bigger network. This leads to an optimization problem, requiring the investigation of different networks.

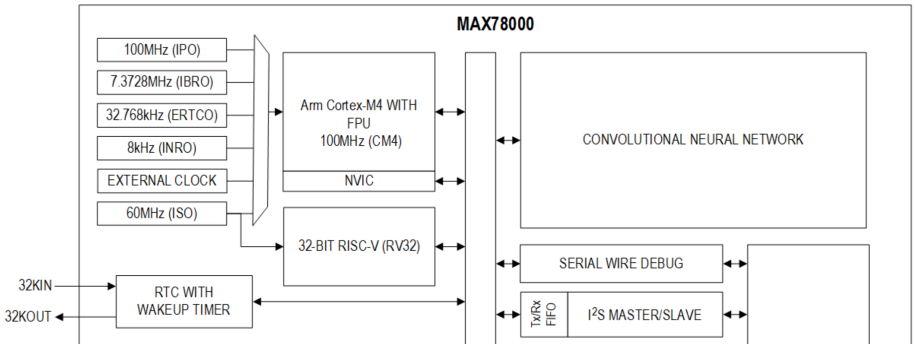


Figure 3.4 – Part of the block diagram for the MAX78000 AI Microcontroller which embeds an AI accelerator using configurable quantization. Image cropped from the datasheet provided by Maxim.

During this thesis I trained several different networks on different valuations in order to investigate the impacts of quantization on popular neural networks, ranging from 1 to 32 bit.

3.4 Simulation environments

Setting up a simulation environment for large memristive systems can cause several problems. The following chapter provides a brief introduction to the problems and possible mitigations.

3.4.1 Layers involved in a memristive design

The layers of abstraction involved when designing or simulating large systems can be seen in Fig. 3.5.

As described in the previous chapter the physical processes happening within a RRAM device cannot be described within a simple formula and subsequently require extensive modelling. Additionally, the RRAM device itself requires appropriate appropriate control circuitry in order to perform meaningful operations such as logic, arithmetic or storage operations. This analog circuit connects blocks such

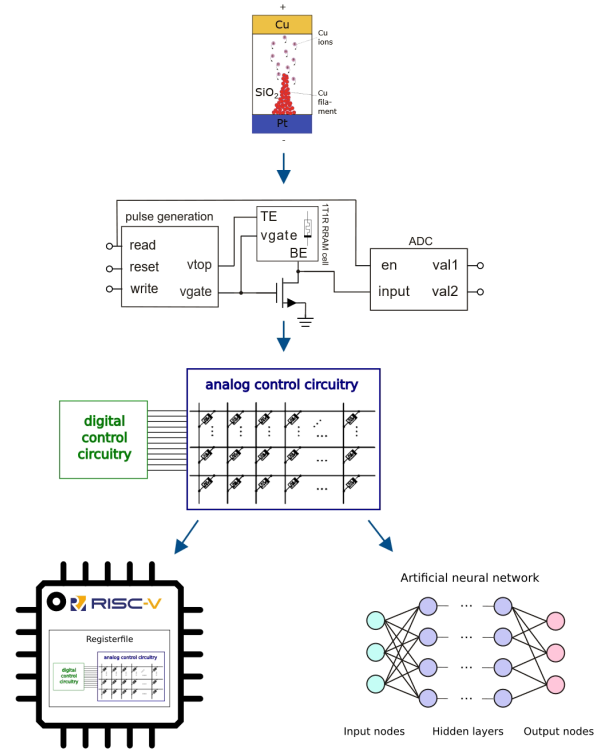


Figure 3.5 – Layers involved when evaluating large designs.

as transmission gates or operational amplifiers in order to do so. This circuit is embedded within a digital control circuitry which decides which combination of pulses is appropriate for a given task at a given time. This circuitry might be described as a digital circuit, using HDL languages such as VHDL or Verilog This circuitry is - in turn - part of a larger system. Examples of such larger systems might be a Risc-V CPU or an architecture performing operations on an artificial neural network.

This leads to the introduction and combination of several descriptive languages:

- **RRAM**: The most common language is Verilog-A since it allows for the proper modelling of differential equations.
- **analog circuitry**: Common descriptive languages are spice or spectre
- **digital circuitry**: Digital designers commonly use VHDL or Verilog
- **Systems/applications**: Large Systems are described in VHDL, Verilog or SystemC. Neural networks tend to be defined with Python or Matlab

3.4.2 Mixed Signal Simulation

A common concept to integrate those different descriptive languages into a coherent simulation lies within mixed signal simulation, which introduces a tightly coupled simulation engine which integrates the different language mentioned above into one coherent simulation. Several languages have evolved to ease integration, like VHDL-AMS [25], Verilog-AMS [26] and SystemC-AMS [27].

I have used Cadence Incisive in my previous works as a means to integrate RRAM devices both within a CPU [28] and within a neural network [29].

Unfortunately combining the simulation accuracy of a Verilog-A simulation with the scale of a system simulation is problematic since tens of millions of devices have to be simulated, leading to simulation times which prevent a reasonable investigation. This becomes even more difficult when taking device variations into account since each simulation has to be run multiple times.

3.4.3 Parametrized circuit evaluation

As doing a mixed signal simulation for the entire circuit is problematic it appears reasonable to do parametrization simulations for a specific circuit and integrate the results within a higher level of abstraction. The concept of such a flow has been shown in an earlier paper [30] and is shown in Fig. 3.6.

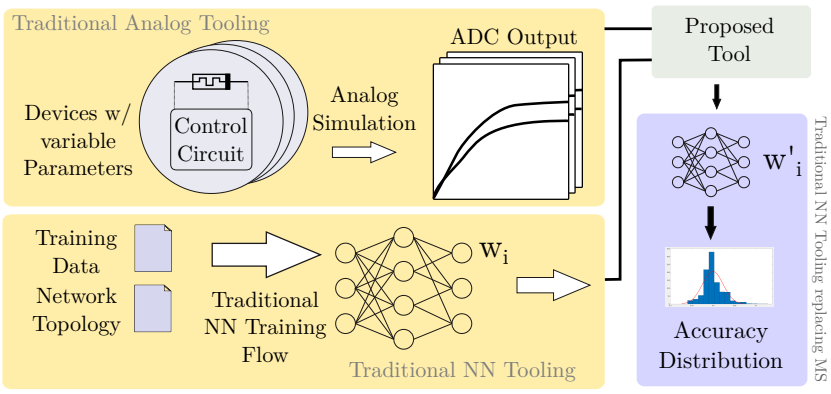


Figure 3.6 – Parametrizing the individual elements instead of running a full simulation yields a vast increase in simulation performance. Figure taken from my publication [30].

This flow consists of the following steps:

- Designing a circuit which embeds the Verilog-A RRAM model
- Running parametrization simulations and extracting the parameters
- Designing a system or a neural network within a high level framework

- Running the simulation within this framework and using the parameters extracted earlier

As a result the simulation time can be reduced by several orders of magnitude.

3.4.4 Using raw measurement data

Significant effort has been put into the accurate modelling of devices. in my recent works I have used the model designed by Reuben et al. [31]. This model is a so-called physical model which has been fit to an actual device. Unfortunately this might not resemble the actual device physics, especially regarding device variations.

Fortunately extensive measurement data is available for RRAM devices. It appears reasonable to directly use this data to automatically derive a model which can be used both within Verilog-A and High-Level investigations (see next section).

3.5 ASIC development

The content in this section largely comprises of works by Jansen [32] and Weste [33].

An ASIC (application specific integrated circuit) can be designed at varying layers of abstraction (see fig. 3.7).

Figure 3.7 – Different layers of abstraction when implementing ASICs

While analog chips require a full custom design which involves the manual placement of each individual component (such as transistors) digital chips are usually designed using a standard cell design flow. These standard cells are provided by the foundry and comprise of a selection of implementations of logic gates such as *AND* or *OR* gates with a varying number of inputs of outputs. These cells contain a behavioral description (such as a truth table) for simulation, a circuit schematic and a circuit layout which can later be used by the design tool to layout the chip. This transformation from logic gate to transistor schematic can be seen in fig. 3.8.

Subsequently the designer can select a combination of these logic cells and does not have to design analog circuits. This is called "logic design".

The so-called "Register Transfer Level" chipdesign methodology allows for the description of hardware in hardware description languages such as VHDL or Verilog.

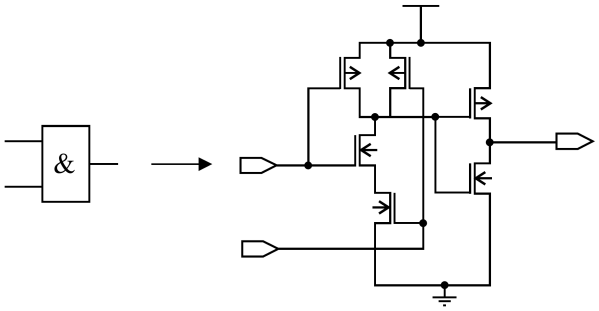


Figure 3.8 – Transformation of an AND gate from logic symbol to transistor schematic [32, p. 39]. This enables the designer to place a selection of gates instead of having to manually place transistors as it would be mandated by a full custom design.

This level is independent of individual logic gates, the designer describes the behavior of a circuit. The synthesis tool transforms this description into a combination of the beforementioned standard-cells. These are then automatically arranged onto a grid of standard cells by the so-called place-and-route tool (see fig 3.9)

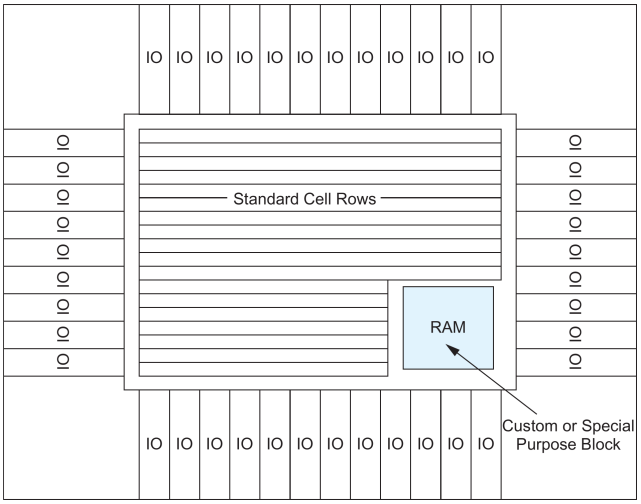


Figure 3.9 – Standard cell chip layout [33, p. 642]. Standard cells are placed on a grid and surrounded by IO cells, forming an IO ring. Custom macro cells such as RAM can be inserted.

IO cells (also provided by the fab) serve as interconnect to other parts on the printed circuit board and implement ESD (electrostatic discharge) protection. Custom blocks (such as a RAM macro cell) can be inserted and automatically connected.

3.5.1 Standard Cell based design

The addition of two numbers can be described in VHDL via the statement

```
1 a <= b + c
```

which will add the numbers *b* and *c* and store the result within *a*.

The synthesis tool has to translate this description into a combination of standard cells. This process is called *synthesis*. A possible synthesis (assuming that a, b and c are single-digit positive integers) is shown in fig. 3.10.

The addition needs to be performed in three steps: Firstly, the two Inputs *a* and *b* are put into a so-called Halfadder. Secondly, this result is reused within a second Halfadder stage, considering the carry bit input. Finally, both the output *Sum* and the carry bit output *c* are calculated. Different topologies have been devised in order to efficiently calculate the sum of multiple digits.

In total five gates are required in order to perform the operation which has been shown in the VHDL listing - two *XOR*, two *ADD* and a single *OR* gate. The beauty of this approach lies within the simplicity: The developers work consists of writing "code", the intrinsicacies of translating the code to actual hardware are handled by the synthesis tool.

Significant improvements have been achieved by providing more sophisticated standard cells, providing many-input logic operations or multiplications. Larger cells, such as memory or error correction blocks, which do not physically fit within the standard cell grid, can be integrated within a macro cell.

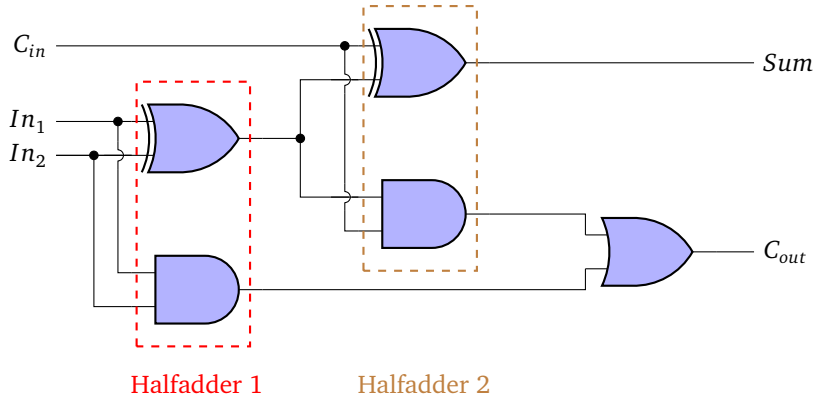


Figure 3.10 – Single-digit Full adder consisting of two half adders. A total of five gates is used to implement this functionality.

3.5.2 Mixed Signal chipdesign

Combining analog circuits with the standard-cell focussed flow mentioned earlier poses problems since the digital flow depends on parametrized standard cells in order to do ensure timing constraints are met. Additionally, drive strength of digital cells are limited, one cannot directly connect a large transistor gate. These analog blocks also do not necessarily fit into the grid defined for the placement of standardcells.

Such a parametrization data does not exist for an arbitrary analog circuit, they are the result of a carefully conducted set of parametrization simulations.

ASIC designers have come up with two possible mitigations, namely the "Big-A" and the "Big-D" design flows. When doing a "Big-A" design the digital designer restricts himself to the generation of appropriate standard cell rows, the IO cells are omitted. The design is handed over to the analog designer, which places the standard cell grid within his analog design and manually connects both IO cells and the analog parts of the design.

When doing a Big-D design the analog designer has to run a parametrization of his circuit blocks, using tools such as *Cadence Liberate*. This results in a macro-block with precise timing information, similar to the RAM cell mentioned earlier. The digital designer can subsequently place this block within his design and perform a fully-automated synthesis and place & route of the design.

Special care has to be taken of the supply voltages since analog circuits might require additional voltages not present within the IO ring of a digital chip. Custom IO cells need to be inserted in order to mitigate this problem.

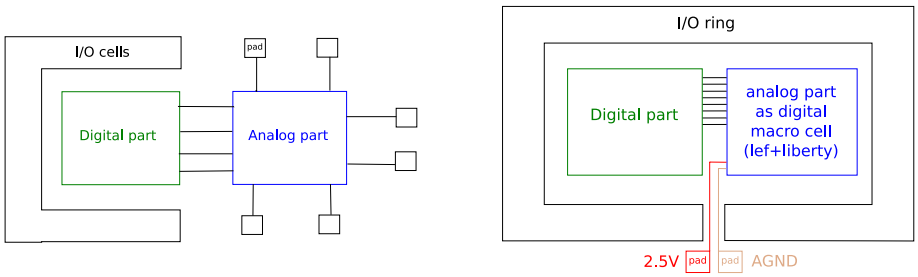
3.5.3 Integrating RRAM devices

One of the main difficulties when introducing rram devices into digital designs lies within the required supply voltages - forming the devices requires multiple Volts, while the maximum Voltage rating for the digital cells is less than 2 V. Two possibilities to circumvent this issue are shown in fig. 3.11.

Cutting open parts of the IO ring (right) enables the designer to stick to his design flow since he does not have to worry about drive strengths or possible ESD issues. He can export his digital design for the analog designer, who will proceed to integrate the digital part into his analog block.

Another option lies within creating a macro cell for the RRAM-part. Unfortunately the digital IO cells do not support providing the required voltages within the chip, so a gap has to be created to provide the appropriate voltages.

However, there is a third option - if the circuits containing the RRAM devices are sufficiently small they might fit within the standard cell grid. This eases the design significantly since they can then be automatically inserted into the design by the synthesis tool. However, this requires the characterization of these cells as a



(a) Big-A **(b)** Big-D

Figure 3.11 – Two design methodologies for the integration of RRAM devices: Biga (left) and bigd(right)

standard cell. The creation of such a library will be showcased in the methodology section.

Chapter 4

Methodology

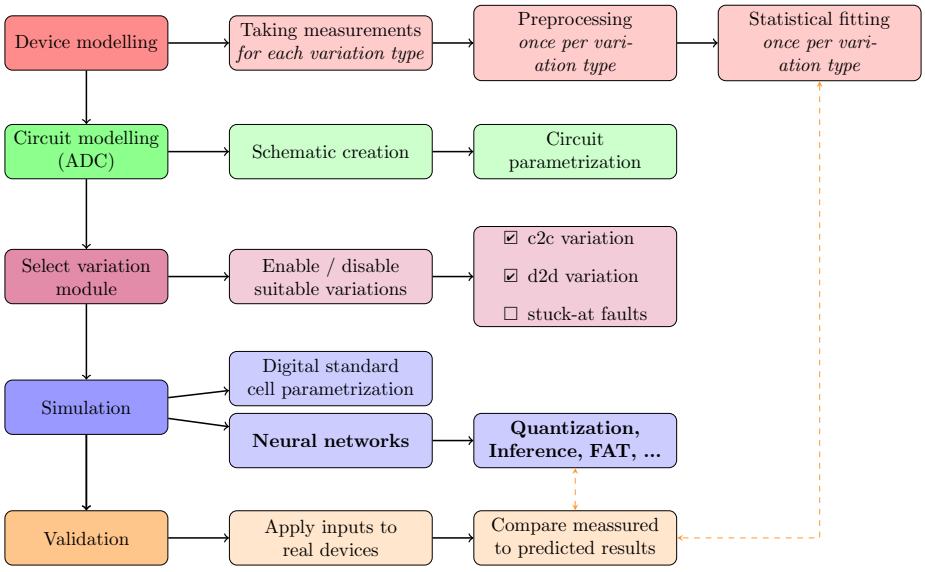


Figure 4.1 – Methodology implemented by this thesis: Adding RRAM devices to an application involves many different aspects such as modelling, device variation, simulation and verification. This thesis proposes a flow which is capable to combine these parts into a coherent flow

The overall methodology implemented by this thesis is depicted in fig. 4.1. It covers a significant part of the steps involved in moving neural networks to an RRAM-based accelerator.

As of now these parts have mostly been investigated independently, which renders integrating these steps difficult. This thesis attempts to mitigate that by providing a flow which starts with raw device measurements, creates a statistical model and allows for the integration of these models into a neural network training environment.

The strongest aspect lies within the flexibility and the simulation speed: This approach is device agnostic, it can be applied to any device (given that measurements are available) and the actual simulation runs on the GPU (and/or on a HPC cluster).

Additionally, individual device measurements have been done to explore the usability of these devices for ternary logic cells. An example of how to derive a digital standard cell library from analog circuits is provided.

4.1 Device modelling

The most crucial part about integrating RRAM devices into a simulation environment lies within accurate models: If a model does not resemble an actual devices behaviour the designed chip is unlikely to provide the expected performance.

This work provides a way to automatically derive a statistical model from raw measurements by implementing three steps:

- Take measurements using fabricated devices
- Preprocess and filter this data
- Fit a statistical model and export its parameters

While investigating individual devices provides a lot of insight (see remarks in section 5.3) deriving statistical methods requires a solid foundation, relying on data taken from hundreds if not thousands of devices. Subsequently this work uses data from 4096 devices set up in a crossbar and fabricated using IHP 250 nm technology (section 5.2) to derive a model.

These measurements yield information regarding

- How many devices do not work at all?
- At which condition will a device alter its state?
- Does applying the same condition to the same device multiple times yield different results?

Statistical distributions can be fit to this information. These distributions can - in turn - be used to automatically derive a model which can be used in different programming languages and environments such as Pytorch, Verilog-A or System-C.

An example for fitting these distributions to measurement data for device to device variations is shown in fig 4.2.

Each device is set to a specific state using the ISPVA algorithm. Subsequently, the current state is determined by using a readout pulse, measuring the current and deriving the resistance. The data is visualized by breaking down the individual

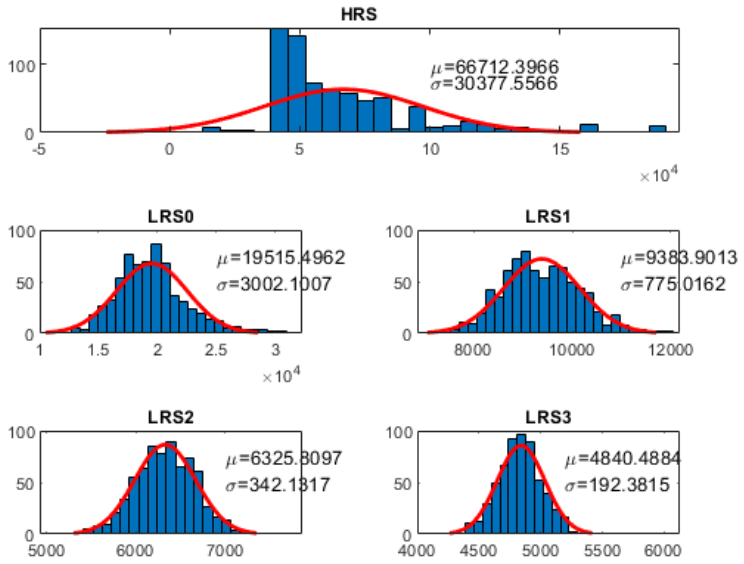


Figure 4.2 – Fitting statistical distributions to measurement data for each state individually. The x axis denotes the resistance (in Ω). The y axis denotes the number of occurrences.

measurements into histograms. Subsequently statistical distributions are fit to the distributions which can later be used to construct an accurate depiction within a simulation. For the sake of simplicity only gaussian distributions have been used for this example, the actual flow tries to fit a multitude of distributions and uses the distribution which yields the lowest root-mean-square error (RMSE) vs the actual distribution of resistances (eq. 4.1); x_i denotes the actual measurement while y_i denotes the prediction given by the model.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{i=N}(x_i - y_i)^2}{N}} \tag{4.1}$$

However, these measurements have to match the intended use-case. This model is fit and suitable for the usage of RRAM devices as mere storage. Unfortunately it cannot be used for e.g. logic circuits - the statistical fittings never “saw” a device’s response within logic circuits and therefore cannot predict how the device would react in such a circuit.

Given that one of the important aspects of this work is to evaluate whether RRAM devices could also be used within logic cells a second type of measurements (using individual devices) have been done in order to investigate the suitability of RRAM devices for logic gates.

4.2 Circuit modelling

RRAM devices cannot perform operations by themselves, they need to be surrounded and driven by appropriate circuitry. This circuit can be vastly different, depending on the actual application - a RRAM crossbar utilized for storage entails different characteristics than a circuit embedding few devices as a logic gate.

During this thesis the RRAM devices are to be used as mere storage, subsequently a control circuit and an ADC are required to convert the analog parts to digital signals. A so-called DC simulation is used to derive a model from this circuit, the results of which look similar to the graph shown in 4.3. This circuit has one input, resembling the voltage dropping across the device, representing the device's state, and three outputs which switch at different input voltages. This allows for the digitization of the current state. A DC simulation renders this behaviour parametrizable by applying a range of input voltages and observing its output.

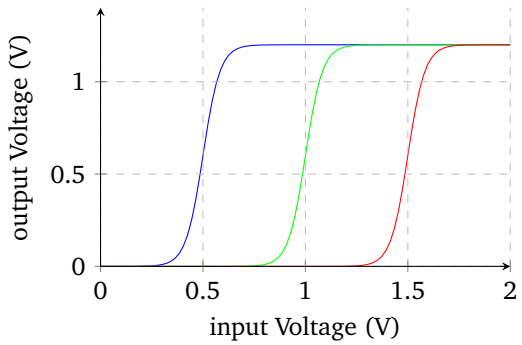


Figure 4.3 – DC simulation of an ADC which is used for its parametrization.

While this is straightforward for storage circuits logic circuits might need a more thorough parametrization (see later section).

4.3 Variation modules

RRAM devices are affected by different kinds of variations. While some devices are largely affected by device to device variations, others might predominantly be affected by cycle to cycle variations. The IHP 250 nm RRAM technology is strongly impacted by stuck-at faults (see section 5.2).

It seems reasonable to be able to investigate different kinds of variations independently, having these switched on or off before (or even during) an investigation. Each variation type is based on fitting a model to measurements as demonstrated earlier.

This methodology provides a “switch” for each variation type in order to accommodate this.

4.4 Simulation

The auto-generated model described earlier can now be plugged into an appropriate simulation environment. The environment-agnostic nature of this statistical model allows for the usage within vastly different simulation environments such as neural network simulations, mixed signal simulation environments or standard cell parametrization flows.

4.4.1 Generic simulation

This subsection will elaborate on how this model works and how it can be integrated in a generic simulation.

Firstly, the tool has to define how many devices are to-be simulated.

Subsequently, a random sample from each distribution is taken for each individual device. If five states are to-be used five samples are taken. A possible result for this is shown in fig 4.4. As this depicts device to device variation this should not change during the simulation. This matrix is stored within memory and reused whenever a device is being accessed.

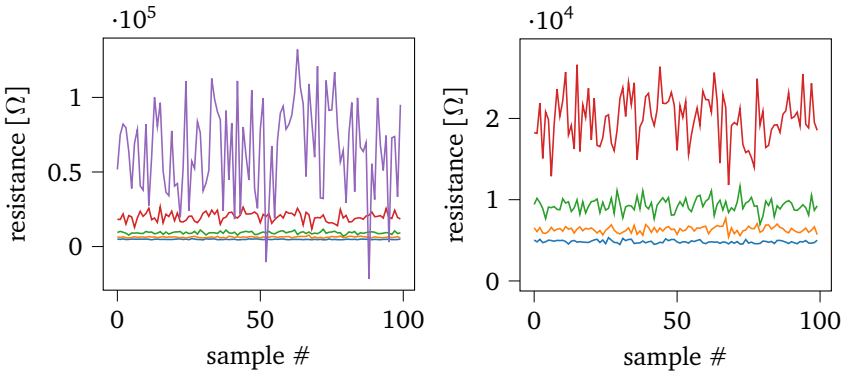


Figure 4.4 – Simulated devices as drawn from the statistical distributions; Each color represents a different state and each sample # represents a different device. The figure on the left depicts both the LRS and the HRS states while the figure on the right merely depicts the LRS states for improved readability.

Different error modes exist, such as stuck-at errors. For these errors the environment draws one sample per device to decide whether it is stuck at one or at zero respectively (fig. 4.5).

This can be repeated for each different type of device fault.

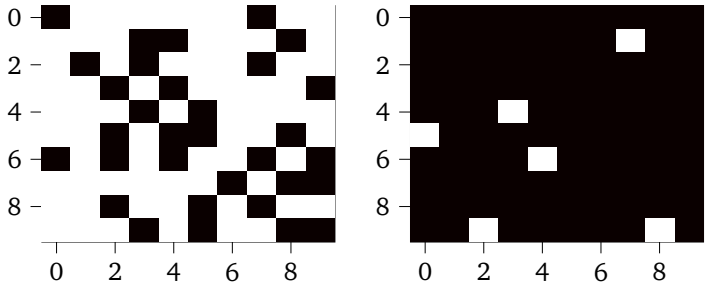


Figure 4.5 – Cells stuck at 0 (left) and cells which are stuck at 1 (right)

The matrices derived above can be used repeatedly during the actual simulation. Whenever a given value - eg a 0 - is stored within a device the simulator combines the beforementioned matrices to determine which resistance would be present for this specific device for this specific storage operation. Combining this with the DC parametrization curve of the ADC shown in fig. 4.3 yields an accurate prediction which value would actually be stored within the device. Doing this for a hundred devices yields the results as depicted in fig. 4.6.

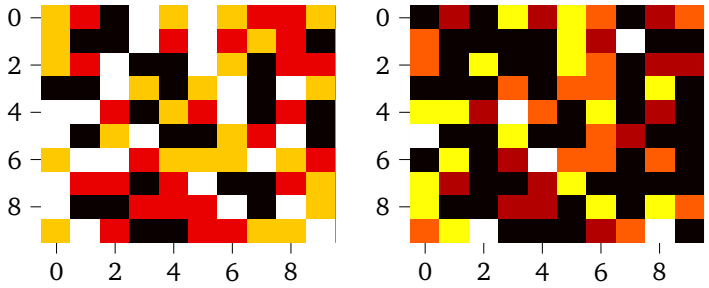


Figure 4.6 – Storing 100 weights in 100 devices: Correct weights (left) and weights as they would be stored within devices (right)

4.4.2 Digital standard cell generation

As lined out in section 3.5.1 a possible way of implementing an ASIC embedding RRAM lies within generating a digital standard cell. A possible way to derive such a cell is shown in fig. 4.7

Figure 4.7 – Standard cell generation

Firstly, an analog circuit which yields the desired behaviour (e.g. a *NAND* gate) has to be designed and subsequently verified using a simulation. Subsequently the

spice netlist which contains all the parts and connections is exported. This netlist is then imported into a circuit parametrization tool such as Cadence Liberate. The tool performs a multitude of different analog simulations for all possible combinations of inputs and derive information regarding its behaviour. This information covers aspects such as propagation delay, input capacitance or current consumption during operation. Lastly, this can be used to both derive a behavioural digital verilog netlist and a so-called liberty description, which can later be used as a standardcell within a purely digital synthesis flow.

4.4.3 NN training/inference environment

The different steps involved in evaluating RRAM-driven neural networks using the proposed approach are shown in fig. 4.8. As a first step the user has to define a network. Subsequently, the applicability of a network for quantization has to be evaluated. This is performed by training the network multiple times, using different valuations, ranging from 1 to 32 bit. If the achieved accuracy is satisfactory the RRAM variation module is added on top of the quantization module. Both the training and the inference will now work as if they were using actual devices as storage.

As soon as the network has achieved sufficient accuracy it can be exported as an onnx file and used within the real application. This onnx can also serve as a basis to synthesize hardware.

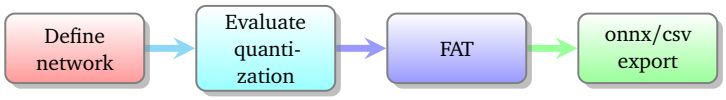


Figure 4.8 – Neural network module

The *Brevitas* environment (provided by Xilinx) which builds on Pytorch has been used as a basis for these implementation efforts [34]. Both custom quantizers and a custom variation module have been added to allow for the evaluation of these devices. These parts have been implemented to be GPU compatible to allow for the fast evaluation of large networks.

4.5 Neural Networks

The forward pass of a simple fully connected layer using three input nodes $y_{1,2}$, six weights w and two output nodes $x_{1,2,3}$ is given by

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{4.2}$$

which can be written shorter in matrix notation:

$$\mathbf{y} = \mathbf{W}\mathbf{x} \tag{4.3}$$

Training this network resembles minimizing the Loss function

$$L(\mathbf{x}, \mathbf{W}, \mathbf{y}) = \frac{1}{2} \|\mathbf{W}\mathbf{x} - \mathbf{y}\|_2^2 \tag{4.4}$$

which requires the calculation of two gradients, namely with respect to the weights

$$\frac{\partial L}{\partial W} = (\mathbf{W}\mathbf{x} - \mathbf{y})\mathbf{x}^\mathbf{T} \tag{4.5}$$

and with respect to the inputs

$$\frac{\partial L}{\partial x} = \mathbf{W}^\mathbf{T}(\mathbf{W}\mathbf{x} - \mathbf{y}) \tag{4.6}$$

Upgrading the weights is performed in multiple steps:

- Calculating the error tensor of a given layer by calculating the gradient with respect to the inputs: $\mathbf{E_{n-1}} = \mathbf{W}^\mathbf{T}\mathbf{E_n}$
- Upgrading the weights by using the gradient with respect to W (η depicts the learningrate): $\mathbf{W^{t+1}} = \mathbf{w^t} - \eta \mathbf{E_n} \mathbf{X}^\mathbf{T}$
- Backpropagating the error tensor to the previous layer

Applying this approach to a multi-layer network is shown in fig. 4.9 and is referred to as “backpropagation” during the so-called “backward pass”

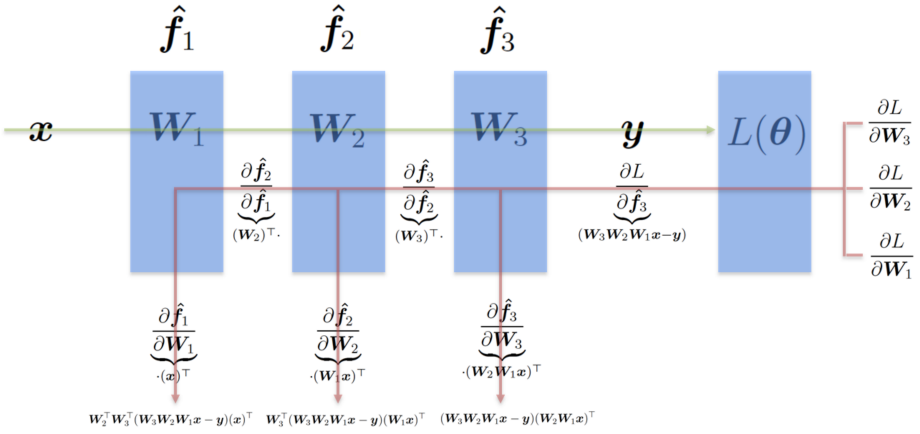


Figure 4.9 – Forward pass (green) and backward pass (red) implementing backpropagation within a multi-layer network. Figure taken from the Deep Learning lecture given by Andreas Maier

4.5.1 Quantizing neural networks

Adding quantization to the inference is rather straightforward since the weight matrices applied during the forward pass (eq.4.2 and eq.4.3) do not depend on other layers. Different quantization algorithms have been evaluated regarding their applicability (see implementation chapter) and implemented as a custom quantizer block.

Given that not all neural networks yield sufficient accuracy when quantized drastically a tool to automatically evaluate different levels of quantizations (for weight, input and activation quantizations) has been devised. This tool runs the evaluation on hundreds of HPC GPUs in parallel to allow for the evaluation of this vast design space within a reasonable timeframe. This leads to a plot as illustrated in fig. 5.19.

Adding quantization to the training procedure is significantly more complicated since the weights itself are used to derive the gradients described in eq. 4.5 and eq. 4.6. Replacing continuous values with a discrete valuation yields non-continuous gradients, possibly interrupting the training process, especially when those are calculated on a per-layer basis via backpropagation.

This problem has been solved by storing two variants of the weights for each layer, namely a quantized and a non-quantized version. The non-quantized version is used to calculate the gradients and to backpropagate the non-quantized error tensors while the quantized version is used for the remaining calculations and the inference. These are synchronized after the backpropagation has been finished. This yields the benefit of maintaining a derivable gradient while ultimately integrating the quantization error.

This basic idea is based on the “straight through estimator” which as been proposed by Bengio et al. [35] and which has been further developed for quantized neural networks by Yin et al. [36].

4.5.2 Adding device variations

While the beforementioned approach works for quantization maintaining a continuous gradient while being true to the device variations is a bit more difficult since errors such as stuck-at errors are - by definition - non-continuous. For others, such as device to device variation, an analytical description of the error might help maintaining the gradient.

On the other hand just keeping with the beforementioned separation does not seem reasonable since the training algorithm would keep trying to change weights which ultimately can never be changed (since the device is stuck).

This approach tries to find a middle ground:

For the forward pass the device faults are applied as described earlier. The full scale of the simulation as depicted in section 4.4.1 has been integrated into a custom pre/postquantizer block utilized within the custom brevitas implementation worked on together with Fraunhofer.

For the backward pass a matrix correlating to the stuck devices is constructed. As a first step each element is initialized with ones (eq. 4.7).

$$\mathbf{G_{modstuck}} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{4.7}$$

Subsequently each weight which is to-be stored within a stuck RRAM device is multiplied by $0 < x < 1$; The precise number is an optimization problem between maintaining continuity and staying true to the devices. Empirically $0.5 < x < 0.7$ has yielded good results. This yields a matrix as shown in eq. 4.8 which is reused for the remainder of the training procedure.

$$\mathbf{G_{modstuck}} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} = \begin{pmatrix} 0.7 & 1 & 1 \\ 1 & 0.7 & 0.7 \end{pmatrix} \tag{4.8}$$

During each backpropagation step each gradient is multiplied with this matrix, which encourages the training procedure to not depend too much on individual weights. This can be further enforced by repeatedly reinitializing the devices during the training procedure, causing different devices to be stuck.

The other types of device errors are integrated in a similar way, by introducing further modifications to the gradient without breaking continuity up to a point where the neural network is untrainable.

4.5.3 Fault Aware Training

The concept of Fault aware training has been proposed by Zahid et al. **??** as a means to mitigate faults caused by radiation faults within an FPGA performing neural network inference. This thesis implements this approach for the training of neural networks storing its weights on (possibly faulty) RRAM devices.

While computation speed might not have a strong impact on doing mere inference on small neural networks, implementing FAT for large networks mandates the implementation within a sufficiently fast NN framework. Subsequently great care has been applied during the implementation of the beforementioned steps to ensure this requirement is being fulfilled. Each element of data is stored within a pytorch tensor and merely tensor operations are utilized during the gradient calculation to ensure that the required calculations can be performed in a fast and GPU-compatible manner.

Training the LeNet network on the MNIST dataset for 20 epochs using this implementation leads to the graph shown in 4.10. As one can see the network seems

to train fine up until 20% of broken devices but seems to run into gradient issues when the percentage is increased further.

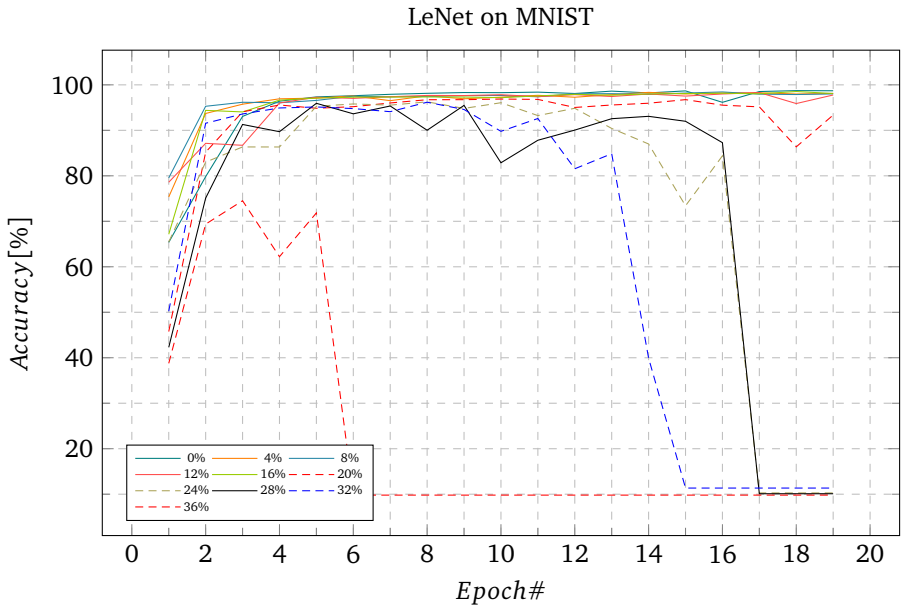


Figure 4.10 – We trained the LeNet network on the MNIST dataset while taking both device variations and stuck-at errors into account. Different lines represent different percentages of stuck devices. The network seems to train successfully up until 20%.

4.5.4 Validation

Validation is an important aspect of science, especially when constructing a model.

In order to validate the usability of this approach a new set of measurements which has not been used for deriving the model has been done. The model is used to derive both stored matrices and to train neural networks embedding fault aware training.

Ideally, the predictions of stored matrices strongly correlates with the measurements and the neural networks which have been trained using FAT yield higher accuracies than the ones which have not been trained embedding FAT.

Chapter 5

Meassurements and Implementation

This chapter provides a description of methods utilized to achieve the goals of this thesis. It starts out with an explanation regarding the required measurements and proceeds to elaborate how the adjacent circuitry has been parametrized. Subsequently it provides a description of how this data is used to derive and integrate this model into a neural network framework. Finally a description of how to automatically generate digital standard cells is given.

5.1 Measurements - concept

As mentioned earlier creating an accurate model for a RRAM device is a nontrivial task, especially when variations are to be taken into account. Subsequently i base my own modelling efforts on two different types of measurement setups which serve as a reliable ground truth. The first meassurements are taken on devices intended for storage and which are arranged in a crossbar. These devices are fabricated using IHP 250 nm technology. The second meassurements are taken on individual devices performing logic operations and which have been fabricated using IHP 135 nm technology.

These different technology nodes had to be used since there is not (yet) enough data on the comparatively new 135 nm devices in order to do a meaningful statistical analysis.

5.2 Measurements - Devices in a crossbar - provided by IHP

The first kind of measurements used for these modelling efforts were taken on a 4kbit RRAM crossbar by IHP (see figure 5.1). Each individual RRAM device is connected

in series with a transistor, resembling the well-known *1T1R* structure. Devices can be accessed independently by applying appropriate voltages to the bitline (BL) and wordline (WL) connections. The NMOS device connected to the wordline doubles as a current compliance device, limiting the current which can flow through the device. This is important since too large currents might destroy the device (see later section).

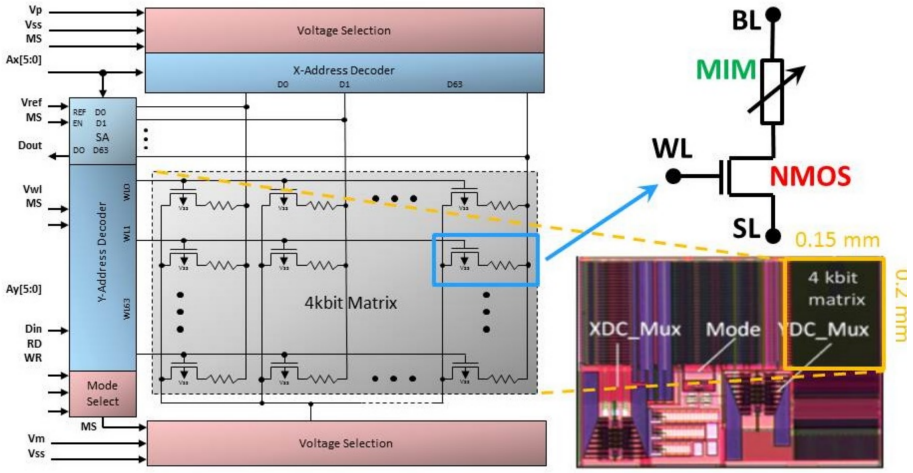


Figure 5.1 – The block diagram of the 4 kbit memory array is shown on the left, while the image on the upper right depicts the schematic of an individual 1T1R cell. The image on the bottom right depicts the actual chip. (Image taken from [37])

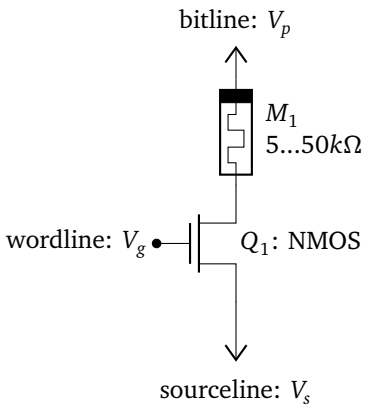
The writing scheme used during this measurements is the so-called *ISPVA* [38]

A more sophisticated version, namely M-ISPVA, has been investigated by Perez et al. and yields promising results, especially for multi-level operation [39]. Plain ISPVA uses one specific compliance current for the different states, ultimately causing problems when dealing with variance[???]. M-ISPVA mitigates this problem by utilizing multiple different compliance currents, depending on the state which is to be written to the device.

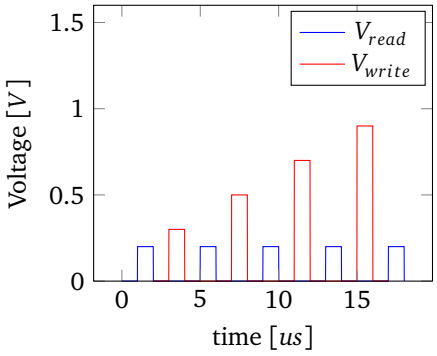
5.2.1 measurement equipment

The RIFLE SE system (provided by N-Plus-T) is being used to do the crossbar measurements. The block diagram of this system is shown in Fig. 5.3. It provides a fully integrated memory measurement system, providing 32 digital control outputs and several analog arbitrary waveform generators which can supply voltages ranging from -12 to 12 volts at up to 100 mA. It can be programmed via a LabView interface.

The main advantage of using such a fully integrated system lies with the reduced complexity - the researcher can focus on the measurement task at hand instead of



(a) 1T1R structure: The transistor Q_1 is connected to the memristor M_1.



(b) ISPVA algorithm: constant read (shown in blue) and stepwise increasing write pulses (shown in red) are applied alternately until a given current threshold is reached during the read operation.

Figure 5.2 – 1T1R structure (left) and the ISPVA algorithm (right). The ISPVA pulses are applied to the bitline (V_p) while a constant voltage V_g is applied to the wordline, ensuring an appropriate compliance current. V_s is connected to ground. The voltages at V_p and V_s are swapped during a reset operation.

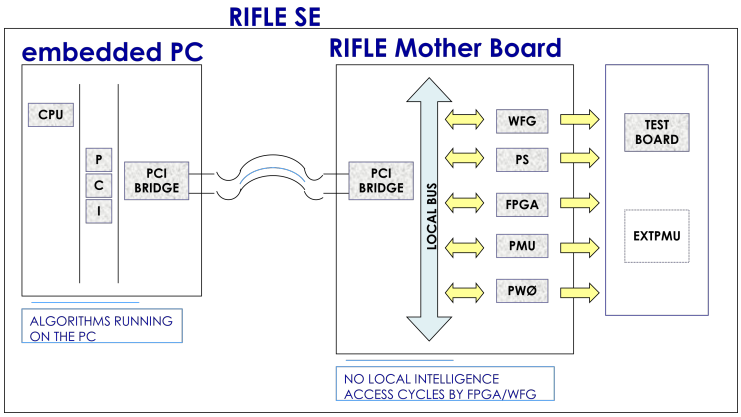


Figure 5.3 – Blockdiagram depicting the RIFLE SE. Taken from the manual.

having to deal with cable connections and a multitude of measurement equipment. The researcher can perform both individual set or reset or an entire ISPVA algorithm from an easy to use LabView interface. Unfortunately the device cannot provide measurements which are as precise as a dedicated SMU (see keithleysection) which is why it is merely used for the crossbar operation.

5.2.2 results

This section describes the measurements for the 4k array. It will begin with a coarse evaluation and proceed with a more detailed investigation.

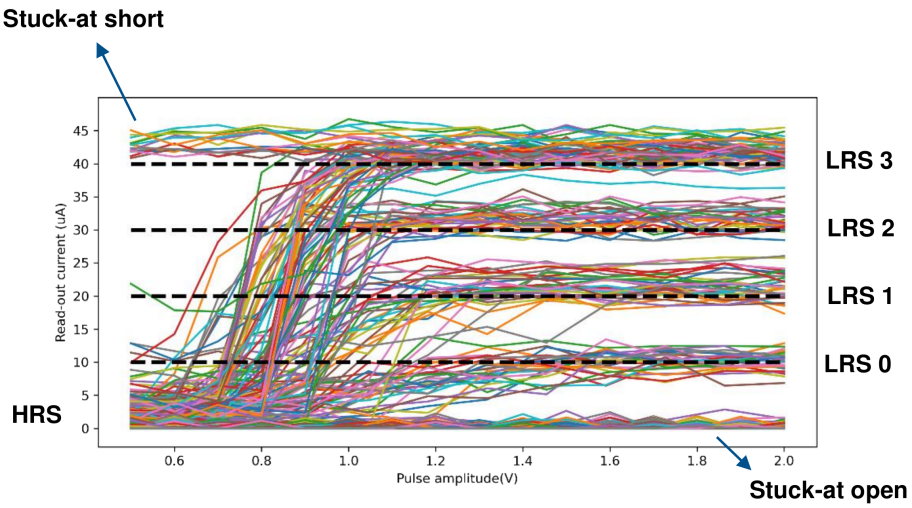


Figure 5.4 – Results: Applying the ISPVA to each device individually leads to this figure. Some devices switch for a lower-than-average V_p, while others seem to require a higher V_p. Another group of devices seems to be stuck at either HRS or LRS3. This figure has been created by Jianan Wen (IHP).

The measurement results for applying the ISPVA algorithm to each individual device are shown in fig. 5.4. While some devices switch to LRS0 at a very low V_p of 0.6V most will switch at around 0.7V. A fraction of devices requires a V_p of 0.8V to 1.6V to switch to LRS0. Higher voltages are required to achieve the LRS states 1 to 3. It becomes apparent that the device variation is significantly more severe than for other semiconductor devices like e.g. transistors.

It is noteworthy that - in addition to mere variation - another kind of error occurs: Some devices will never change their state, they are either stuck at the HRS state or at the LRS3 state, mo matter the amplitude of V_p. Isolating those faults leads to the figure shown in fig. 5.5.

One can deduct that about 25% of devices are stuck at the HRS state while 5% of devices are stuck at the LRS3 state. This indicates that strong error correction is required.

Further separation is required when investigating the variability of non-stuck RRAM devices. The two major variabilities consist of a) device to device (d2d) and cycle to cycle (c2c) variation. While the former describes variations in between achieved readout currents for different devices the latter describes variations in between individual cycles on a single device.

Deriving these statistics for the beforementioned 4k-array leads to the two graphs shown in fig. 5.6. The graph shown on the left depicts d2d variation; 1024 out of the 4096 devices have been selected to determine inter-device variation. There seems to be significant variation, leading to overlapping readout current between the different

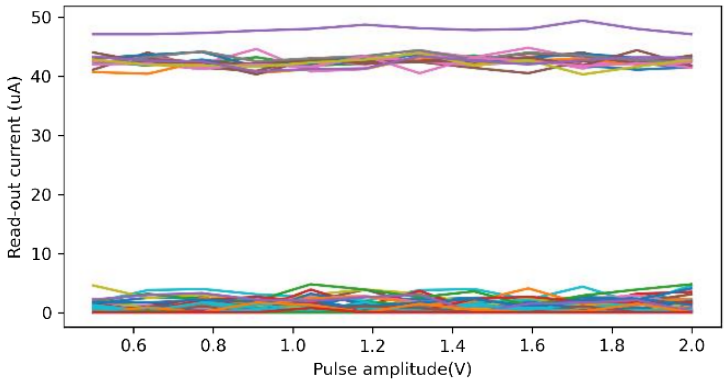
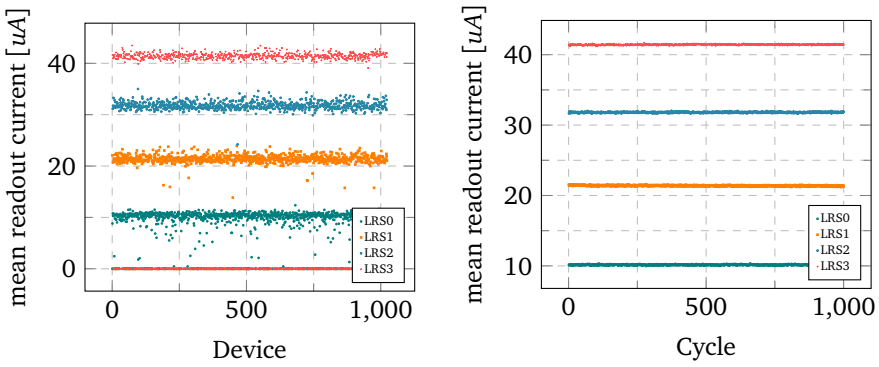


Figure 5.5 – Results: Isolating the devices which are stuck at the HRS or the LRS3 state. About 30% of devices seem to be affected. This figure has been created by Jianan Wen (IHP).

states. The graph shown on the right depicts c2c variation. Valid devices have been selected and a thousand cycles have been applied to an individual device. Each cycle seems to be similar, there is no overlap between states.



(a) 4 × 1024 valid devices

(b) 1000 cycles on valid devices

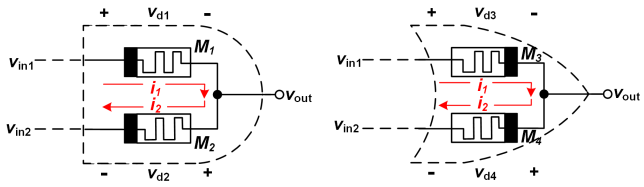
Figure 5.6 – Results: Mean readout currents for both Cycle to Cycle (c2c) (left) and Device to Device (d2d) variability. The d2d variability poses a stronger impact on the devices. Figure created by me using data provided by IHP.

The strongest impact on overall device performance is given by the stuck-at errors. While d2d variation also seems to play a role c2c variation seems to be negligible. There are other kinds of variation (like shifting device resistance during storage) not covered here which could be relevant for future work.

5.3 Individual devices: evaluating gates

In Section 3.2 i introduced the concept of ternary gates and a possible implementation. During this section i will describe my atempt at implementing and meassuring these gates using IHP technology.

Wang et al. have proposed a concept for the implementation of ternary gates embedding RRAMs in [16]. Two gates, namely a *TAND* and a *TOR* gate are shown in fig. 5.7.



(a) ternary and **(b)** ternary or

Figure 5.7

5.3.1 Meassurement Setup

asdf

5.3.1.1 Keithley 4200-SCS

asdf

5.3.1.2 LeCroy Oscilloscope

asdf

5.3.1.3 Multimeter

asdf

5.3.1.4 Wafer prober

asdf

Figure 5.8 – Measurement flow

5.3.2 Measurements

Multiple steps have to be performed before the actual measurement can be done (Fig. 5.8). The devices are not "initialized" straight after fabrication, the filament structure has to be initialized using a *Forming* operation, which can be performed using the Keithley SMU unit in DC mode.

Subsequently, to identify a faulty device, a *Set* and *Reset* operation are performed with the SMU. This should yield a curve which is similar to the one shown in fig 5.10c. A strong deviation from this curve indicates a device fault and suggests choosing a different device.

Unfortunately these preliminary steps take significant amounts of time, about $^3/_4$ of time spent doing these measurements has been spent on selecting, initializing and validating devices.

The next steps depend on the intended measurement, while some can be done with the SMU (DC) others have to be performed using the PMU (AC). The latter requires swapping cables from the SMU to the PMU unit and connecting the Oscilloscope.

Future work should probably add both automated wafer prober control and an automated switching unit, preventing the need to manually swap cables. This should speed up measurements significantly, allowing to focus on the actual investigation.

5.3.2.1 Validation: Reproducing measurements

The first step lies within reproducing the measurements which were already taken at the IHP lab in order to verify that a) the device is working properly and b) the measurement setup works as intended. Subsequently a *forming*, *set* and *reset* pulse has been applied to the device. The circuit configuration for this meassurement is shown in fig 5.9.

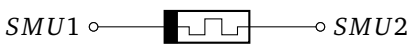


Figure 5.9 – RRAM device connected to two SMU units

The set operation for a single device is shown in Fig. 5.10. Given that this is a DC measurement the SMU is set to subsequently apply 200 individual voltages and wait until the current settles at a given value. The input voltages are depicted in

Fig. 5.10a and range from 0 Volt to 1 Volt. The current flowing through the device during this operation is shown in Fig. 5.10b, it ranges from 0 A to 5e-4 A. The current does not increase linearly with voltage which indicates that the rram device does not act as a simple resistor.

A very common way to plot characteristics of a RRAM device is shown in Fig. 5.10c, namely a DC curve, providing the Voltage on the x, and the current on the y axis. One can clearly recognize the nonlinear behaviour of this device, a resistor would yield a straight line.

According to ohms law the relationship between voltage and current is given by

$$U = R \times I \tag{5.1}$$

which allows for the calculation of the resistance of the device at the present time. Doing this calculation for the provided combinations of voltage and current leads to the figure shown in Fig. 5.10d. Unfortunately this seems to only hold until around sample #75, where a strong increase of current occurs. It is difficult to give a precise reasoning for this behavior, but there are two possible answers: The SMU is hitting its current limit and subsequently cannot properly measure the current. This appears likely given that it never exceeds a very specific value. Ohms law only holds for linear relationships and cannot be applied here, even for individual samples derived by a dc measurement. This appears reasonable since the RRAM device is per definition a non-linear device.

5.3.2.2 Evaluating Series operation

The meassurements in the previous section suggest that a device is operating between 1 kΩ and 10 kΩ. Subsequently the total maximum resistance of two devices operating in series is given by

$$R_{total} = R_{rram1} + R_{rram2} = 10\,\text{k}\Omega + 10\,\text{k}\Omega = 20\,\text{k}\Omega \tag{5.2}$$

which limits the maximum current to an individual device. The same problem arises regarding the voltage at an individual device, which is given by

$$V_{rram2} = \frac{R_{rram2}}{R_{rram1} + R_{rram2}} \times V_{in} \tag{5.3}$$

The question arises whether this comparatively small current and voltage are sufficient to still set and reset an individual device. The circuit configuration utilized for answering this question is shown in fig 5.11. If a single RRAM device can be operated with a 10 kΩ resistor in series series operation should be feasible.

The circuit is driven by two SMU units and a similar DC measurement as shown above is undertaken. The voltage is increased up to 40 V in order to evaluate which voltage is required for a series operation. The results for a reset operation are shown

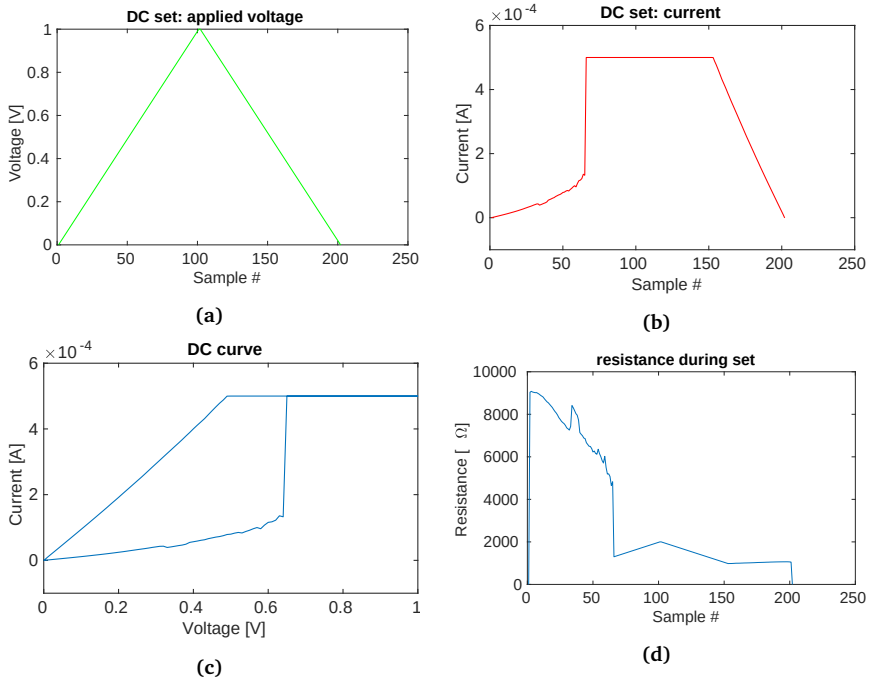


Figure 5.10 – Applying a DC set operation to a single device leads to the curves shown above.

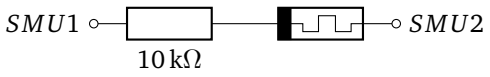


Figure 5.11 – RRAM device in series with a resistor

in fig. 5.12. This is equal to doing a set operation in the circuit configuration shown before.

As one can see the device resets at about 2 V which is well within specifications of the IHP analog library. Subsequently series operation of two devices appears to be feasible.

Given that this individual device was run beyond its specifications (way past 5 V) it is “broken”, it cannot be set or reset anymore. However, this measurement was necessary to determine feasibility of this approach.

5.3.2.3 Evaluating speed of resistance change

The SMU units will apply a given voltage at a given maximum current and wait until the DUT has converged on a given current draw. While this is useful for static parametrization it does not yield any answers regarding timing behaviour.

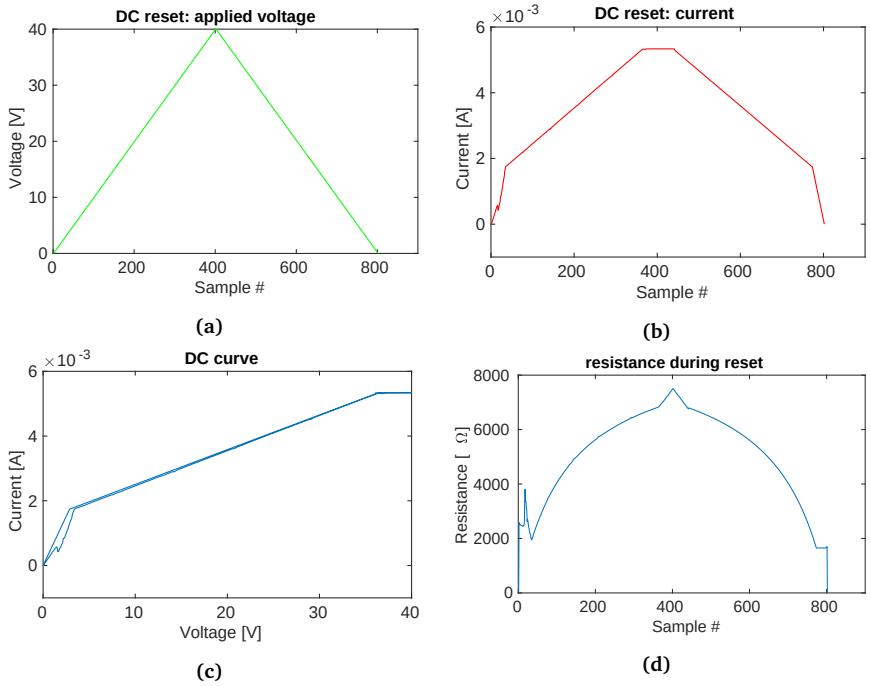


Figure 5.12 – Applying a DC reset operation to a device in series with a resistor leads to the curves shown above. Nonsurprisingly the resistance calculation does not yield meaningful results.

The PMU units can apply arbitrary waveforms such as ISPVA pulses and meassure both voltage and current over time, rendering them an ideal candidate to observe timing behavior. Unfortunately they do not support limiting current, which can lead to

- broken devices due to overcurrent
- measurement artifacts due to the voltage collapsing due to current draw

The circuit configuration for this measurement is shown in fig 5.13. The device terminals are connected to PMU instead of SMU units. The device has been formed, set and reset using the SMU units before applying these meassurements. At this point the programs used by IHP to drive the Keithley unit were not sufficient anymore, I had to write my own programs to control the device according to the requirements.

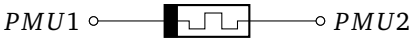


Figure 5.13 – RRAM device connected to two PMU units

ISPVA writing pulses ranging from 0 V to 0.8 V using a pulse width of 10 µs have been applied (Fig. 5.14a) and the observed current has been measured (Fig. 5.14b).

Signal postprocessing has been applied in order to render the graph more readable (Fig. 5.14c, Fig. 5.14d). The current resistance is calculated using Ohm's law and shown in fig. 5.14e.

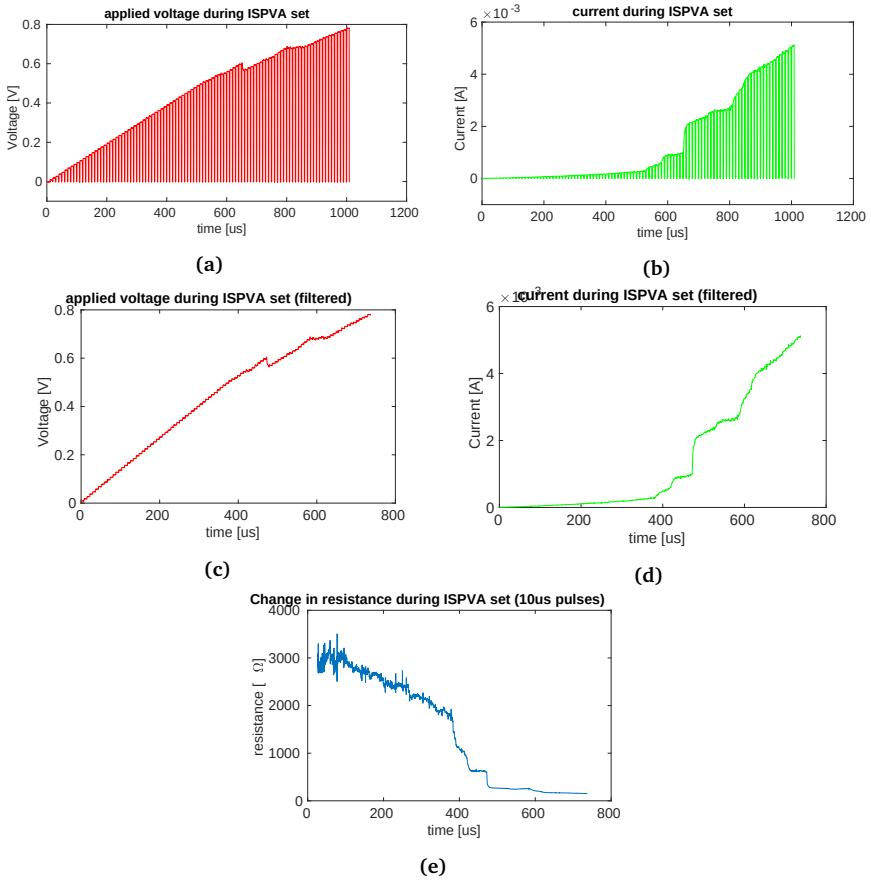


Figure 5.14 – Using the PMU units to apply IPSPVA writing pulses of increasing amplitude to a device leads to the graphs shown above. The second row shows filtered signals which allows for the calculation of resistance in the third row. Applying this filtering leads to a skew in the time axis, which is why the unfiltered signals have to serve as a reference.

As opposed to the previous measurements swapping form the SMU to the PMU allows for a timescale on the y axis instead of a mere sample#. The PMU seems to have problems with maintaining a steady voltage increase after 600 µs which is in line with the sudden increase in current. The change in resistance happens in different speeds:

- 0 µs to 600 µs: The change in resistance happens rather slowly up until a certain voltage threshold

- 600 µs to 800 µs: The change in resistance speeds up when a certain voltage threshold is reached
- 800 µs to 1000 µs: The applied voltage causes the current to be too large, the device is broken at this point. It cannot be reset to high resistance anymore.

This emphasizes one of the main problems when dealing with RRAM devices: Programming these devices is rather slow while staying within small voltage/currents, but increasing the applied pulse amplitudes might not just speed up the programming cycle, it might also irrecoverably break a device, possibly rendering the entire chip unusable.

This is why actual designs will not blindly increase pulse amplitude but measure the current resistance using a read pulse in between the writing pulses. The algorithm has to always start at low amplitudes since the setting voltage might be quite different in between devices (see the d2d variation in the previous section). It also emphasizes the requirement for strict current limiting - devices which are using a 1T1R configuration can be used with significantly higher writing pulse amplitudes.

5.3.3 Ternary logic

The previous sections have proven that series operation of devices might be possible. However, great care has to be applied not to break the involved devices. This section will describe the steps taken to investigate the memristive gates shown in fig 5.7.

Both involved devices have to be initialized individually and are validated using both a *set* and a *reset* operation as depicted in fig 5.8. The circuit configuration used for the initialization is shown in fig 5.15 while the cable setup is shown in fig. 5.17.

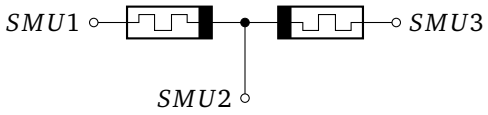


Figure 5.15 – Two RRAM devices in series connected to SMUs

The initialization begins by only applying voltage and a ground connection to $SMU1$ and $SMU2$ and letting $SMU3$ float. Subsequently, $SMU1$ is left floating and the pulses are applied between SMU and $SMU2$. A total set of six programs had to be written for the Keithley to perform these operations.

Subsequently the logic inputs shown in table 3.1 have to be applied to each cell while observing the signal at the facing electrodes (see Fig. 5.17).

However, the Keithley only provides two PMU channels, so a workaround to record three channels had to be found. The solution is depicted in the cable diagram shown in 5.18. An oscilloscope is connected to three nodes and set to record

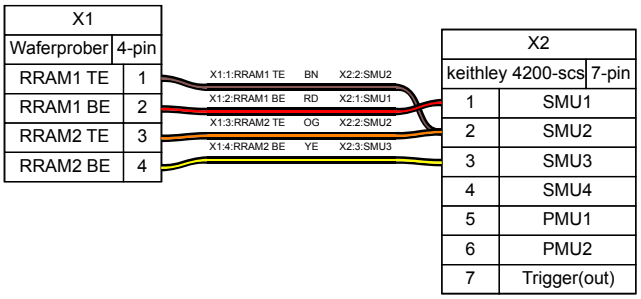


Figure 5.16 – The blockdiagram of the 4 kbit memory array is shown on the left, while the image on the upper right depicts the schematic of an individual 1T1R cell. The image on the bottom right depicts the actual chip. (Image taken from [37])

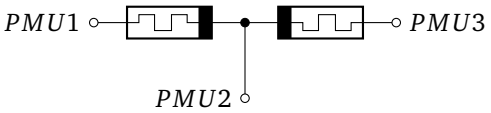


Figure 5.17 – Two RRAM devices in series connected to PMUs. Unfortunately the Keithley 4200 only provides two PMU units, requiring a different approach.

waveforms both to a csv file and to a screenshot whenever a trigger signal is generated by the Keithley 4200. A set of programs has been written to:

- apply the appropriate voltages using the PMUs
- create a trigger signal for the oscilloscope
- automatically store the signals recorded by the oscilloscope to a csv file
- store the current levels as recorded by the PMU since the oscilloscope can only record voltages unless provided with a current probe

Unfortunately no automated wire connecting/disconnecting device was available at the lab, and subsequently the PMUs and SMUs had to be rewired manually for each experiment. This was especially problematic since the beforementioned challenges regarding current limiting caused the destruction of a significant amount of devices.

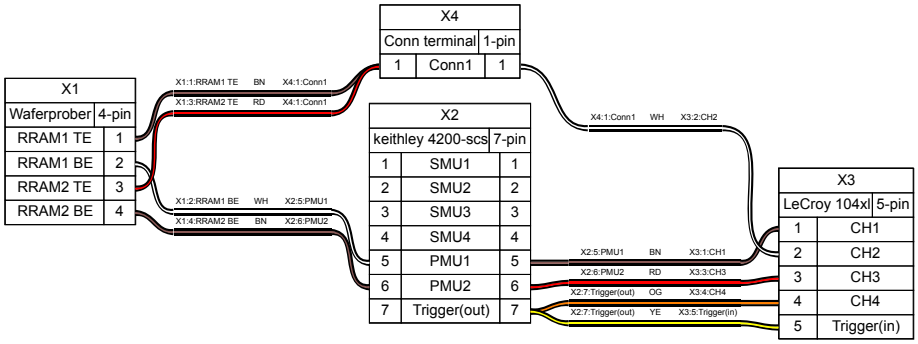


Figure 5.18 – The blockdiagram of the 4 kbit memory array is shown on the left, while the image on the upper right depicts the schematic of an individual 1T1R cell. The image on the bottom right depicts the actual chip. (Image taken from [37])

5.4 ASIC NN interference

5.4.1 parametrizing the ADC

5.4.2 deriving a model from measurement data

5.4.3 integrating the parametrization data into pytorch

5.4.4 quantization

As mentioned earlier in section 3.3.2 there are three major components within a neural network which can be quantized, namely the

- input data
- activations
- weights

5.4.4.1 Framework

I used our adoption of the brevitas framework running on the HPC cluster in order to vary these different aspects. This required both the implementation of a scripting interface in order to evaluate these options automatically and the automated generation of jobscripts which could be run on the cluster. Finally, the data has to be parsed automatically in order to yield meaningful plots.

An individual training can be run by calling eg.

```
1 python nn_benchmark/main.py --experiments outputfolder ↘
      --network QuantAlexNet --dataset CIFAR10  --epochs 50 ↘
      --acq 9 --weq \$WEQ --inq 9 2>\&1 > output.log
```

Which would train a single network using the AlexNet architecture on the CIFAR10 dataset for 50 epochs, using 9 bit quantization for the input data, the activations and the weights. The trained network and the training logs are stored in "outputfolder". The "2>&1" string moves outputs given by stderr to stdout while the last ">" will store the outputs of the training log within "output.log".

Given that the evaluation of different quantization combinations is required this will lead to the training of

$$n_{weights} \times n_{activations} \times n_{inputs} \tag{5.4}$$

networks. 32768 individual networks need to be trained in order to evaluate the entire space of possible quantizations. This clearly cannot be done on a commodity desktop, which is why i extended the framework, allowing for the training of the individual network on the "Alex" cluster which was recently acquired by the RRZE (specifications shown in table 5.1)

CPU	2×AMD EPYC 7713 (64 cores each)
Memory	512 GB DDR4
GPU	8×NVIDIA A40 (48GB DDR6)
Local storage	7 TB NVMe SSDs
Operating system	AlmaLinux 8

Table 5.1 – Alex GPU compute node

5.4.4.2 Results

I will start by setting those three components to the same valuation and proceed to do a more detailed investigation afterwards.

I began with training differently quantized fully connected neural networks on MNIST data. The network's accuracy during different training epochs can be seen in fig. 5.19 (left). This architecture seems to greatly benefit from moving from 1 to 2 bit, but yields diminishing returns when going forward. However, even the 1-bit network yields an accuracy of about 92%, which might be suitable for certain usecases.

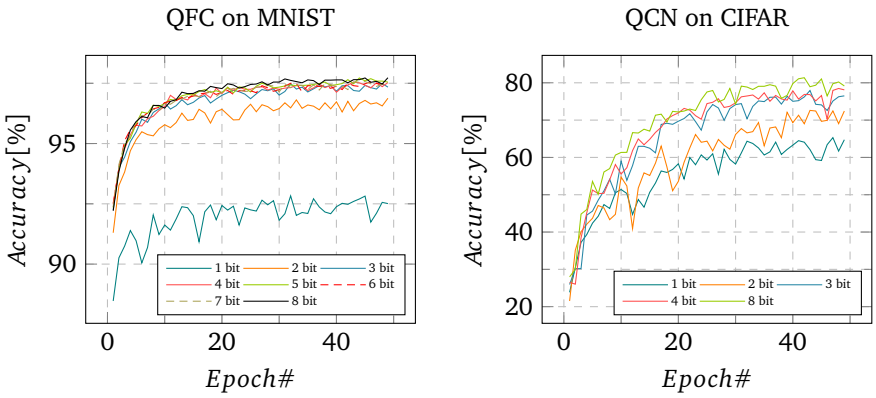


Figure 5.19 – Training a fully connected network on MNIST data (left) or a convolutional network on the CIFAR dataset (right) utilizing different quantizations each for the weight storage yields the results shown above. There seems to be a strong correlation between achieved accuracy and quantization.

Neural network architectures have advanced significantly since the first fully connected networks, a major step forward being convolutional networks. The impact of quantization on a convolutional network operating on the CIFAR dataset can be seen in fig. 5.19 (right). The difference between quantizations is significantly larger than for the QFC network, accuracy ranges between 60% and 80%.

It seems appropriate to use different quantizations for the input data, activations and weights in order to possibly safe chip area. An investigation of different options for the LeNet network can be seen in fig. 5.20.

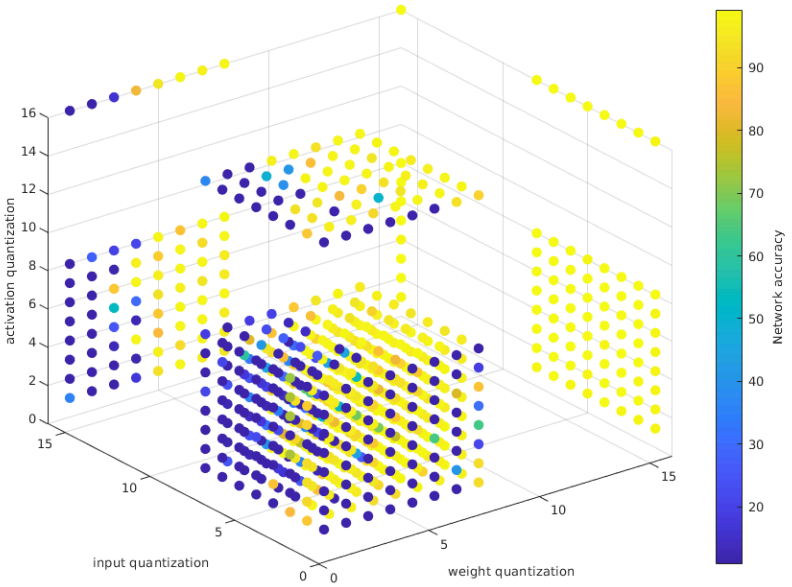


Figure 5.20 – asdf

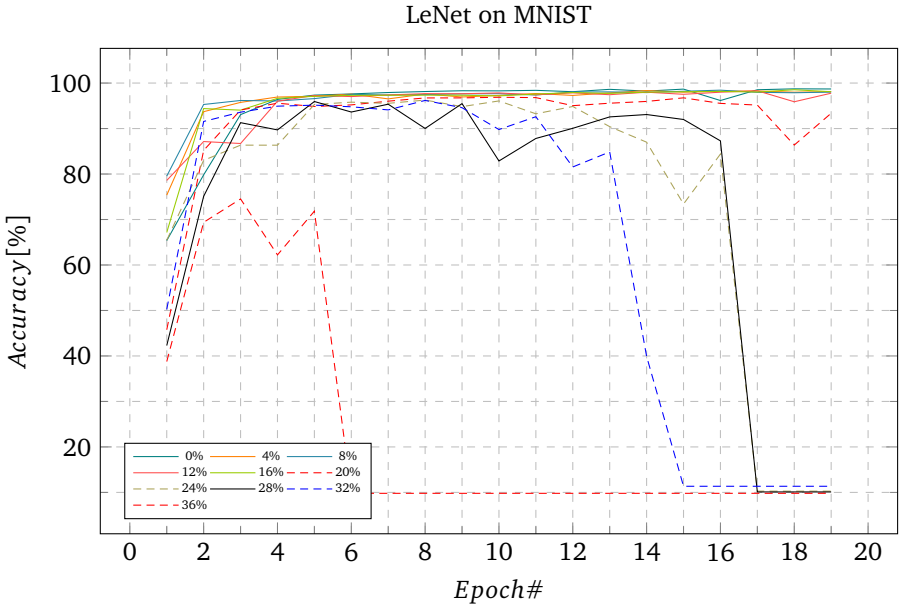


Figure 5.21 – Evaluating different percentages of stuck-at-faults for the RRAM devices fabricated at IHP leads to the figure shown above. The training process ceases to work when a certain threshold of faulty devices is reached.

List of Figures

List of Tables

Bibliography

[1] M. Fritscher, J. Knödtel, M. Mallah, S. Pechmann, E. P.-B. Quesada, T. Rizzi, C. Wenger, and M. Reichenbach, "Mitigating the Effects of RRAM Process Variation on the Accuracy of Artifical Neural Networks," in *21th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, 2021 (cit. on p. 1).

[2] M. Fritscher, J. Knödtel, D. Reiser, M. Mallah, S. Pechmann, D. Fey, and M. Reichenbach, "Simulating large neural networks embedding MLC RRAM as weight storage considering device variations," in *Proc. of 12th IEEE Latin America Symposium on Circuits and Systems*, 2021 (cit. on pp. 1, 2, 4).

[3] M. M. Markus Fritscher, *custom brevitas*, `https://gitlab.cs.fau.de/cs3/lodric/training-fat/-/tree/master/`, 2021 (cit. on pp. 3, 5).

[4] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Team: Threshold adaptive memristor model," *IEEE transactions on circuits and systems I: regular papers*, vol. 60, no. 1, pp. 211–221, 2012 (cit. on p. 4).

[5] Z. Jiang, Y. Wu, S. Yu, L. Yang, K. Song, Z. Karim, and H.-S. P. Wong, "A compact model for metal–oxide resistive random access memory with experiment verification," *IEEE Transactions on Electron Devices*, vol. 63, no. 5, pp. 1884–1892, 2016 (cit. on p. 4).

[6] J. Reuben, D. Fey, and C. Wenger, "A modeling methodology for resistive ram based on stanford-pku model with extended multilevel capability," *IEEE Transactions on Nanotechnology*, vol. 18, pp. 647–656, 2019 (cit. on p. 4).

[7] B. Li, Y. Wang, Y. Wang, Y. Chen, and H. Yang, "Training itself: Mixed-signal training acceleration for memristor-based neural network," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2014, pp. 361–366 (cit. on p. 4).

[8] C. Lammie, W. Xiang, B. Linares-Barranco, and M. R. Azghadi, "Memtorch: An open-source simulation framework for memristive deep learning systems," *Neurocomputing*, vol. 485, pp. 124–133, 2022 (cit. on p. 4).

[9] D. Querlioz, O. Bichler, and C. Gamrat, "Simulation of a memristor-based spiking neural network immune to device variations," in *The 2011 International Joint Conference on Neural Networks*, IEEE, 2011, pp. 1775–1781 (cit. on p. 4).

[10] S. Pechmann, T. Mai, J. Potschka, D. Reiser, P. Reichel, M. Breiling, M. Reichenbach, and A. Hagelauer, "A low-power rram memory block for embedded, multi-level weight and bias storage in artificial neural networks," *Micromachines*, vol. 12, no. 11, 2021. [Online]. Available: `https://www.mdpi.com/2072-666X/12/11/1277` (cit. on p. 4).

[11] L. Xia, P. Gu, B. Li, T. Tang, X. Yin, W. Huangfu, S. Yu, Y. Cao, Y. Wang, and H. Yang, "Technological exploration of rram crossbar array for matrix-vector multiplication," *Journal of Computer Science and Technology*, vol. 31, no. 1, pp. 3–19, 2016 (cit. on p. 4).

[12] J. J. Zhang, K. Basu, and S. Garg, "Fault-tolerant systolic array based accelerators for deep neural network execution," *IEEE Design Test*, vol. 36, no. 5, pp. 44–53, 2019 (cit. on p. 4).

[13] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, "Vortex: Variation-aware training for memristor x-bar," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6 (cit. on p. 4).

[14] U. Zahid, G. Gambardella, N. J. Fraser, M. Blott, and K. Vissers, "Fat: Training neural networks for reliable inference under hardware faults," in *2020 IEEE International Test Conference (ITC)*, IEEE, 2020, pp. 1–10 (cit. on p. 4).

[15] J. Knödtel, M. Fritscher, D. Reiser, D. Fey, M. Breiling, and M. Reichenbach, "A model-to-circuit compiler for evaluation of dnn accelerators based on systolic arrays and multibit emerging memories," in *2020 9th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, 2020, pp. 1–6 (cit. on p. 4).

[16] X.-Y. Wang, P.-F. Zhou, J. K. Eshraghian, C.-Y. Lin, H. H.-C. Iu, T.-C. Chang, and S.-M. Kang, "High-density memristor-cmos ternary logic family," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 1, pp. 264–274, 2021 (cit. on pp. 8–10, 36).

[17] A. Dhande, R. Jaiswal, and S. Dudam, "Ternary logic simulator using vhdl," in *4th International Conference: Sciences of Electronic, Technologies of Information and Telecommunications*, 2007, pp. 25–27 (cit. on p. 8).

[18] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017 (cit. on p. 10).

[19] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015 (cit. on p. 10).

[20] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993 (cit. on p. 10).

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998 (cit. on p. 10).

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012 (cit. on p. 10).

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778 (cit. on p. 11).

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions*, 2014. arXiv: 1409.4842 [cs.CV] (cit. on p. 11).

[25] E. Christen and K. Bakalar, "Vhdl-ams-a hardware description language for analog and mixed-signal applications," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 46, no. 10, pp. 1263–1272, 1999 (cit. on p. 14).

[26] P. Frey and D. O'Riordan, "Verilog-ams: Mixed-signal simulation and cross domain connect modules," in *Proceedings 2000 IEEE/ACM International Workshop on Behavioral Modeling and Simulation*, IEEE, 2000, pp. 103–108 (cit. on p. 14).

[27] A. Vachoux, C. Grimm, and K. Einwich, "Analog and mixed signal modelling with systemc-ams," in *2003 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 3, 2003, pp. III–III (cit. on p. 14).

[28] M. Fritscher, J. Knödtel, M. Reichenbach, and D. Fey, "Simulating memristive systems in mixed-signal mode using commercial design tools," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 225–228 (cit. on p. 14).

[29] J. Knödtel, M. Fritscher, D. Reiser, D. Fey, M. Breiling, and M. Reichenbach, "A model-to-circuit compiler for evaluation of dnn accelerators based on systolic arrays and multibit emerging memories," in *2020 9th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, 2020, pp. 1–6 (cit. on p. 14).

[30] M. Fritscher, J. Knödtel, D. Reiser, M. Mallah, S. Pechmann, D. Fey, and M. Reichenbach, "Simulating large neural networks embedding mlc rram as weight storage considering device variations," in *2021 IEEE 12th Latin America Symposium on Circuits and System (LASCAS)*, IEEE, 2021, pp. 1–4 (cit. on p. 14).

[31] J. Reuben, M. Biglari, and D. Fey, "Incorporating variability of resistive ram in circuit simulations using the stanford–pku model," *IEEE Transactions on Nanotechnology*, vol. 19, pp. 508–518, 2020 (cit. on p. 15).

[32] P. D.-I. D. J. (Dirk Jansen (auth.), *The Electronic Design Automation Handbook*, 1st ed. Springer, 2003 (cit. on pp. 15, 16).

[33] N. Weste and D. Harris, *CMOS VLSI Design*, en, 4th ed. Upper Saddle River, NJ: Pearson, Mar. 2010 (cit. on pp. 15, 16).

[34] A. Pappalardo, *Xilinx/brevitas*, 2021. [Online]. Available: `https://doi.org/10.5281/zenodo.3333552` (cit. on p. 26).

[35] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013 (cit. on p. 28).

[36] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," *arXiv preprint arXiv:1903.05662*, 2019 (cit. on p. 28).

[37] Ó. G. Ossorio, E. Pérez, S. Dueñas, H. Castán, H. García, and C. Wenger, "Effective reduction of the programing pulse width in al: Hfo<inf>2</inf>-based rram arrays," in *2019 Joint International EUROSOI Workshop and International Conference on Ultimate Integration on Silicon (EUROSOI-ULIS)*, 2019, pp. 1–4 (cit. on pp. 32, 43, 44).

[38] E. Pérez, A. Grossi, C. Zambelli, P. Olivo, and C. Wenger, "Impact of the incremental programming algorithm on the filament conduction in hfo₂-based rram arrays," *IEEE Journal of the Electron Devices Society*, vol. 5, no. 1, pp. 64–68, 2017 (cit. on p. 32).

[39] E. Perez, C. Zambelli, M. K. Mahadevaiah, P. Olivo, and C. Wenger, "Toward reliable multi-level operation in rram arrays: Improving post-algorithm stability and assessing endurance/data retention," *IEEE Journal of the Electron Devices Society*, vol. 7, pp. 740–747, 2019 (cit. on p. 32).