Department of Physics and Astronomy
Heidelberg University

Bachelor Thesis in Physics
submitted by

Nils Bergler

born in Eberbach (Germany)

2025

Testing full-custom SRAM for Neuromorphic Hardware

This Bachelor Thesis has been carried out by
Nils Bergler
at the
Kirchhoff-Institute for Physics in Heidelberg
under the supervision of
Prof. Dr. Johannes Schemmel

Abstract

In this thesis the Static Random Access Memories (SRAMs) of the BrainScaleS-2 neuromorphic hardware platform are tested and characterized with respect to the timing configurations as well as the hyperparameters supply voltage and temperature. A software library is presented for testing SRAMs on neuromorphic hardware in general and in particular the SRAMs of the BrainScaleS-2 platform implemented. Using this library all SRAMs of the platform are analyzed with respect to the timing configurations and the impact of the supply voltage and temperature on the reliability of the memory. The analysis over 14 systems finds a strong impact of the read timing on the reliability for two of the five tested SRAMs, while the remaining timing parameter show a weaker impact. Here we also found an anomaly regarding one of the SRAM that showed an unsuspected behavior under certain timing parameters. Furthermore, bugs in the digital logic in one of SRAM controllers were found. The tests under varying hyperparameters found an acceleration regarding the read operations with increasing supply voltage and temperature.

Zusammenfassung

In dieser Arbeit werden die Static Random Access Memories (SRAMs) Speicher der neuromorphen Hardware-Plattform BrainScaleS-2 getestet und charakterisiert. Es werden Messungen im Bezug auf die Zeitkonfigurationen der Speicher sowie der Hyperparameter Versorgungsspannung und Temperatur durchgeführt und analysiert. Hierzu wird eine Softwarebibliothek zum Testen von SRAMs auf neuromorpher Hardware vorgestellt und die SRAM Speicher der BrainScaleS-2 Plattform implementiert. Die Analyse über 14 Systeme zeigt einen starken Einfluss der Lesezeitkonfiguration auf die Zuverlässigkeit bei zwei der fünf getesteten SRAMs, während die restlichen Zeitkonfigurationen einen schwächeren Einfluss aufweisen. Zudem wurde eine Anomalie in einem der SRAM Speicher gefunden, die unter bestimmten Zeitparametern ein unerwartetes Verhalten zeigte. Darüber hinaus wurden Fehler in der digitalen Logik eines weitern SRAM Speichers gefunden. Die Tests unter variierenden Hyperparametern zeigen eine Beschleunigung der Leseoperationen mit steigender Versorgungsspannung und Temperatur.

Contents

Chapter 1.

Introduction

Spiking neural networks (SNNs) are biologically inspired neural networks that mimic the behavior of the brain. Aside from simulation on classical computers, SNNs can also be implemented in (neuromorphic) hardware, which can show advantages in energy efficiency and presents a novel approach to machine intelligence. To realize neuromorphic hardware there is a need for fast and reliable memory, as it needs to reference definitions and configurations of the neurons and synapses, which define the behavior of a SNN. On the BrainScaleS-2 neuromorphic hardware platform, these memories are realized as multiple Static Random Access Memories, covering different aspects of the system. They are implemented as full-custom design with tight margins to achieve high energy and area efficiency, which also makes them unique in regard to the behavior of the configuration and the impact of hyperparameters. Hence, we need tests of these SRAMs to ensure the correct operation and characterize them with respect to their configuration and hyperparameters.
Another aspect is that with complex systems like neuromorphic hardware, we need to be able to ensure the reliability of all parts of the system. This includes the behavior of the memory and under different conditions, such as varying supply voltage and temperature. Thus, necessitating a way to test and analyze the behavior of the SRAMs and give optimal configurations for the memory.
This thesis aims to test the SRAMs of the BrainScaleS-2 (BSS-2) neuromorphic hardware platform and characterize them with respect to the timing parameters. Furthermore, we investigate how reliable the SRAMs are under varying supply voltage and temperature, and how the behavior of the SRAMs changes under these conditions. With this information we can then determine a stable and reliable configuration for the SRAMs of the BSS-2 platform and understand the reliability of this configuration under different conditions.
This work will present an implementation of a software library for testing and characterization of the SRAMs of the BSS-2 chip integrated into the software stack of the platform.
In Bergler (2024), we already have shown that the synapse memory of the BSS-2 platform showed faults for certain timing configuration and we devised a calibration algorithm to ensure fault free operation of the memory. In this thesis we will extend this analysis to all remaining SRAMs of the BSS-2 platform and carry out the investigation of the hyperparameter described above.

Chapter 2.

Background and Materials

2.1. SRAM Theory

2.1.1. SRAM basics

Static Random Access Memory is used widely in modern computer systems for its speed and size. It is composed of many identical cells each holding a singular bit of information. The nowadays most common SRAM cell is the 6T-cell, composed of 6 transistors, 4 of these form two interlocked CMOS inverters which create a bistable circuit that can be used to store binary data. The remaining two transistors are the access transistors connecting each of the CMOS inverters to the two bitlines carrying the data and their gates connected to the wordline used for selecting a cell (Bosio et al., 2012). There are other types of SRAM cells, such as the older 4T-cell or less common used cells with more transistors trading size for increased reliability (Rathi et al., 2023).

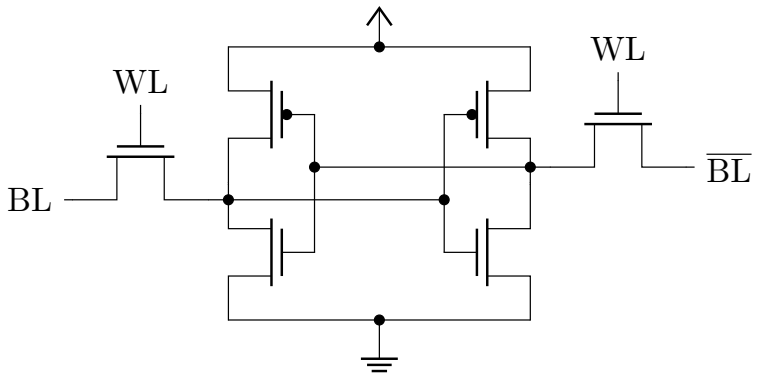


Figure 2.1.: The standard 6T SRAM cell. WL is the wordline and BL the bitline value.

To access a specific cell an address decoder is used to convert an address to the correct word- and bitline to be selected on the SRAM. For simple SRAM the address selects just row and column, in more complex cases the SRAM may be organized in deeper structures such as blocks, with multiple stages of the address decoder descending a binary tree.
The read and write operations on a SRAM cell have to be done with special SRAM controllers to ensure the correct operation of the memory, because we need precise timings

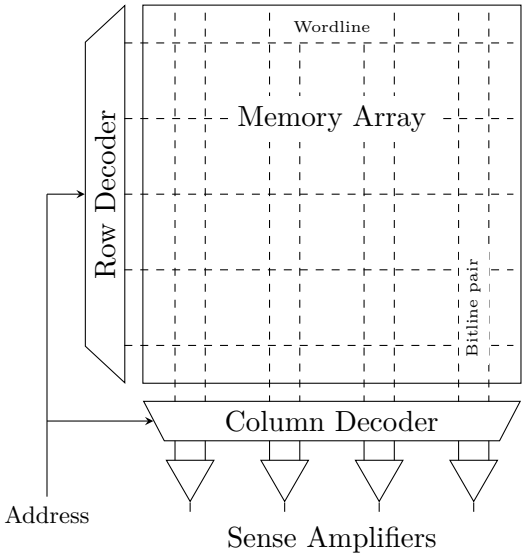


Figure 2.2.: Functional diagram of a SRAM. Adapted from Dounavi and Tsiatouhas (2023).

of the cells access to ensure the correct operation of the memory. The basic operations are:

— **Write**: To write a value to a SRAM cell the bitlines are driven at the desired logical value and the wordlines activated. On the side of the inverters carrying a logical "1" a bitline charged to a logical "0" then forces the value onto the inverter, if the pull-up transistors of the inverter is weaker than the access transistor. The changed inverter then forces the opposite state onto the other inverter, bringing the cell into a stable state.

— **Read**: To read the value in a cell, the bitlines are charged to the supply voltage and then disconnected from the charging circuit. After the wordline is activated, the side of the logical "0" starts to discharge the bitline, causing a voltage difference between the bitlines. A sense amplifier is then used to sense the voltage difference and determine the read value. Importantly the pull-down transistor of the inverter must be strong enough to prevent altering the logical value in the cell.

2.1.2. SRAM timings

All SRAM operations need to be timed correctly to ensure correct operations. First a setup time is needed before the activation of the wordlines to ensure the SRAM has reached the desired initial state for the operation. For a read operation this would include the time to charge the bitlines and ensure complete equalization between them. For a write operation this would include the time until the bitlines have reached the desired threshold of the driven logical values. Second, the SRAM also needs time during

wordline activation, which ensures that a sufficient voltage difference is present to create a well-defined state of the memory, which during a write operation must be long enough to possibly flip the state of the cell and for a read operation must be long enough to reach a large enough voltage difference between the bitlines to be sensed. Furthermore, during all of this, the data and address supplied to the SRAM has to be valid and constant (Pavlov and Sachdev, 2008).

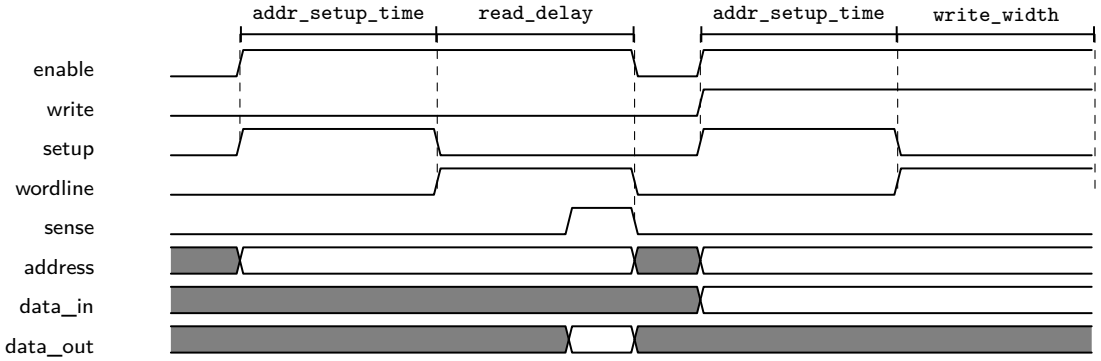


Figure 2.3.: The sequential steps of a SRAM read and write operation, with the timings of the BSS-2 SRAM controller included.

In linear/small signal approximation for any of these operations the rate to cause a desired voltage difference ΔV is given by

$$T = \frac{C \cdot \Delta V}{I} \tag{2.1}$$

where C is the capacitance in question and I the typical current (Pavlov and Sachdev, 2008). For example, in the case of a read operation C, is large due to the size of the bitlines and I small due to the size/speed of the transistor, which are designed to be as small as possible. Therefore, one wants to choose ΔV as small as possible, such that the sense amplifier can determine the read value as early as possible.
The dynamics of the operation may also vary over the SRAM due to variations in the parameters of the transistors or not all positions in the array being dynamically equivalent (e.g., a cell at the start of a bitline vs at the end).
Aside from the sensing threshold of the sense amplifier, one also has to consider the static noise margin (SNM) of these operations. One can only be certain about the state of a cell if ΔV has a sufficient distance to the desired value larger than the expected noise. The SNM is then defined as the maximum distance a voltage can vary under the worst possible conditions, such that the desired outcome is still achieved. One example where this is relevant is the write operation, where the SNM is the induced voltage difference on the cell minus the maximum voltage difference needed to switch the cell. If the SNM is too small, stochastic faults can occur (Pavlov and Sachdev, 2008).

As shown, there are many factors that come In with these timings, and can practically only to be determined by measurement. For this reason the analysis of the behavior of the SRAM timings will be the main concern of this thesis.

2.1.3. SRAM faults and tests

As can be seen from the previous section SRAMs are highly optimized circuits, especially regarding transistor size. Thus, small deviations or incorrect operation can cause faulty read or write operations. Conceptually faults can be assigned to the following operations:

— **cell/hold**: A faulty cell may change its state over time or be stuck at one value.

— **address decoder**: The address decoder may select the wrong, multiple or no cell at all.

— **read**: A faulty read operation could return the wrong value and/or change the value inside the cell.

— **write**: A faulty write operation could fail to write a certain value/transition or write the wrong value to a cell.

In detail, these faults can be attributed to different parts of the SRAM e.g., faulty transistors or bad timings of the operations, and can occur as different fault modes. The most basic fault modes are

— **stuck-at fault**: A cell is stuck at on value

— **stuck-open fault**: A cell is not addressable at all

— **data-retention fault**: A cell changes its logical value after some time

— **destructive read fault**: A read operation changes the value of a cell

— **incorrect read fault**: A read operation returns a different value than stored in the cell

— **transition fault**: A cell/write fails to switch the logical value

Faults in a cell can also be dependent of the states of other cells (coupled faults) or depend on the dynamics of multiple operations (dynamic fault) e.g., fast repetition of the same value. Furthermore, faults may occur only stochastically (Pavlov and Sachdev, 2008).

2.1.4. Hyperparameters of SRAM

Supply Voltage The supply voltage is the most important electrical hyperparameter when designing SRAMs. We know from basic transistor theory that a MosFet in saturation i.e. $U_{\mathrm{GS}} > U_{\mathrm{th}}$ and $U_{\mathrm{DS}} > U_{\mathrm{GS}} - U_{\mathrm{th}}$, obeys the equation (up to polarity for NMOS/PMOS and ignoring channel length modulation and velocity saturation)

$$I_{\mathrm{DS}} = \frac{K}{2}(U_{\mathrm{GS}} - U_{\mathrm{th}})^2 \tag{2.2}$$

By changing the supply voltage we directly change the U_{GS} and U_{DS} voltages applied to the transistor during the operations and which impacts the minimal time for an operation given by Equation (2.1). We expect the SRAM to become faster with increasing supply voltage, but usually the design goal is to minimize the supply voltage, to minimize power consumption (Srinu, Rao, and Sekhar, 2024). If the supply voltage is low enough, the SRAM may even fail completely the voltage becomes too small relative to the SNM, causing read, write and cells to flip randomly.

Temperature Temperature is the most important non-electrical hyperparameter for semiconductor devices, as it is the dominant parameter influencing the solid state physics and statistics of semiconductors. The most direct effect can be seen in the behavior in transistors. The coefficient K in Equation (2.2) for a species n of charge carriers is given by

$$K = \mu_n C_{\mathrm{ox}} \frac{W}{L} \tag{2.3}$$

with C_{ox} the gate capacitance, W and L the width and length and μ_n the carrier mobility of the transistor. While the former are mainly fixed by geometry the carrier mobility varies strongly with temperature. It's given by

$$\mu_n = \frac{q\tau_n}{m_n^*} \tag{2.4}$$

with m_n^* the effective mass and q the charge of the charge carriers and τ_n the mean free time (Sze and Lee, 2012). The mean free time is limited by scattering processes, which are mainly composed of impurity scattering and phonon/lattice scattering of the charge carriers. Theory shows that in approximation we expect the mean free time due to impurities to be proportional to $\tau_{\mathrm{imp}} \propto T^{3/2}$ and due to phonons to be proportional to $\tau_{\mathrm{ph}} \propto T^{-3/2}$. The rates of these effects combine via Matthiessen's rule (Hunklinger and Enss, 2023)

$$\frac{1}{\tau} = \frac{1}{\tau_{\mathrm{ph}}} + \frac{1}{\tau_{\mathrm{imp}}} \tag{2.5}$$

At operating temperature, phonon scattering dominates, we hence expect the mobility to become lower with increasing temperature. Hence, SNM also tends to decrease with increasing temperature (V. Singh, S. K. Singh, and Kapoor, 2020; Arandilla, Alvarez, and Roque, 2011).
The effect is in competition with the threshold voltage U_{th} in Equation (2.2) as it is also temperature dependent, and will typically decrease with increasing temperature. For U_{GS} near the threshold this could lead to an inversion of the temperature effect and speed up the transistor with rising temperature (Zu et al., 2016). Other effects may also be caused by the subthreshold leakage currents i.e, $U_{\mathrm{GS}} < U_{\mathrm{th}}$, as they also increase with temperature (Wolpert and Ampadu, 2011).
For high fields (high voltages and/or short channel) the mobility is also limited due to the velocity saturation effect (Wolpert and Ampadu, 2011). The saturation velocity decreases with increasing temperature, which leads to a decrease of current and hence speed of the transistor.

2.2. The BrainScaleS-2 Platform

The tests of the SRAMs were performed on the BSS-2 platform, a neuromorphic hardware platform developed by the Electronic Visions group. The BSS-2 platform implements 2×256 on-chip analog neurons which can be interconnected via synaptic connections to form SNNs in hardware. The neuron and synaptic properties are defined using multiple parameters and settings, stored on-chip. The information for the synaptic connections is also defined on the chip, interconnecting the two sets of neurons. Furthermore, there are multiple analog state variables such as the membrane potentials, which are digitized using a parallel ADC with 1024 channels called column analog-to-digital converter (CADC) (Pehle et al., 2022).

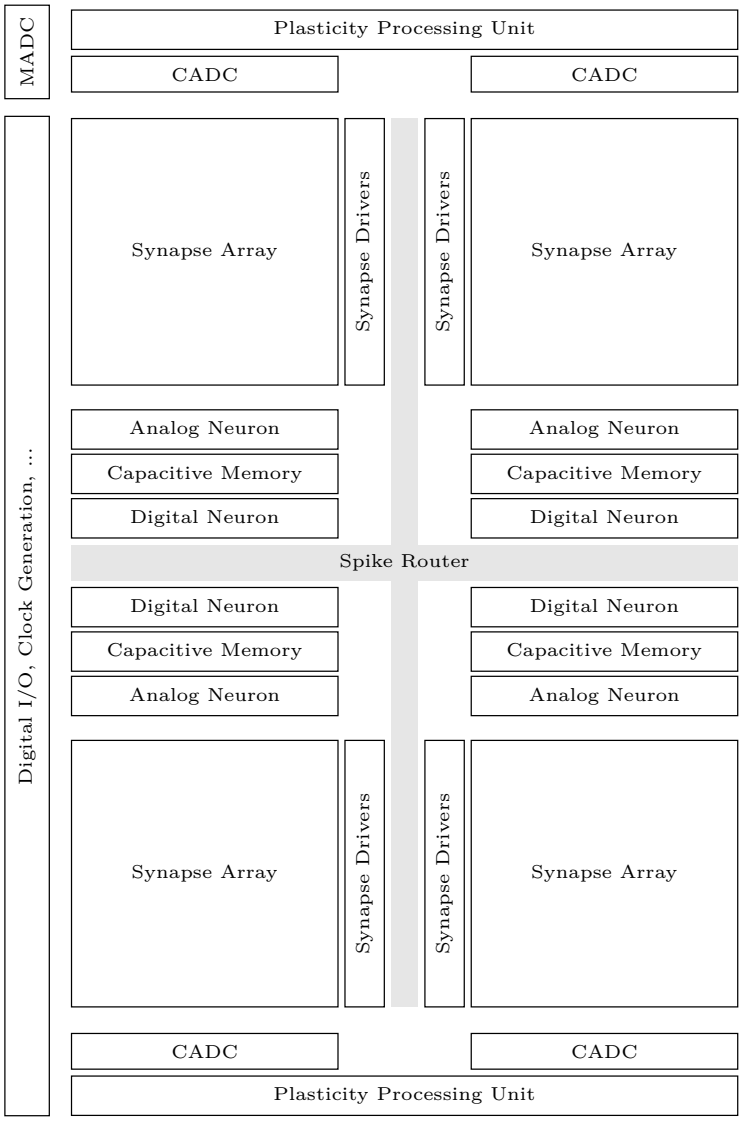


Figure 2.4.: Schematic structure of the BSS-2 ASIC. There are SRAMs for the synapse array, synapse driver configuration, CADC offset and capacitive memory as well as the digital and analog neuron configuration. Adapted from Pehle et al. (2022).

The chip is organized into 4 quadrants on which each neuron is associated with a column, by which the SRAMs cells are indexed. The SRAMs are either addressed by quadrant or hemisphere (north or south).

2.2.1. SRAMs

The BSS-2 chip uses multiple SRAMs memories to store data used for defining SNNs and the configurations of on-chip functions:

— **synaptic memory (SynRam)**: Stores the addresses and weights for the synaptic connections. Per hemisphere there are two 256×256 cell SRAMs for storing the weights and address data for each possible synapse.

— **CADC offset**: Stores the offsets of the CADC channels. Each of the 1024 channel is assigned a signed 8 bit integer for setting the offset, arranged in one SRAM.

— **digital neuron configuration**: Stores the configuration used for the digital parts of the neuron. For each neuron there are 4×6 bits and 2×4 bits as local blocks.

— **analog neuron configuration**: Stores the configuration of the neurons for the analog operation. Per neuron, there are locally 6 bytes.

— **capacitive memory (CapMem)**: Stores the voltages for a DAC with 24 channels per neuron and a resolution of 10 bit per channel.

— **synapse driver configuration**: Stores the configuration of the synapse drivers. Each driver has a block of 2 bytes and 3 bytes local to each driver, of which 2 bits are unused.

The SynRam was already investigated during the preceding internship and will no longer be considered in this thesis. All the SRAMs are accessed using SRAM controllers, with configurable timings for the access operations. For the considered SRAMs the basic controllers are identical, realized as a state machine with digital timings between the steps of an operation. The used timings are:

1. `address_setup_time`: The number of cycles the logic waits before activating the wordlines. 4 bits are used for the timing, giving a range of 0 to 15 cycles.

2. `read_delay`: The number of cycles the wordline is activated after the `address_setup_time` for a read operation. 8 bits are used for the timing, giving a range of 0 to 255 cycles. After this the SRAM goes into idle.

3. `write_width`: The number of cycles the wordline is activated after the `address_setup_time` for a write operation. 4 bits are used for the timing, giving a range of 0 to 16 cycles. After this the SRAM goes into idle.

The timings can also be seen in Figure 2.3.
From a user/machine-learning perspective the SynRams are the most interesting SRAMs which get frequently altered during learning. The rest of the configurations are usually static and only change during setup/calibration of the system. Performance wise we would therefore be interested in setting the timings of the former SRAMs to the minimum possible, while the latter SRAMs could be set to a more relaxed timing if it provides a benefit in reliability.
Considering a clock frequency of 125 MHz one cycle takes 8 ns, a read operation takes at least two cycles and up to 256 cycles or 2 µs for the maximum timing of a read operation.

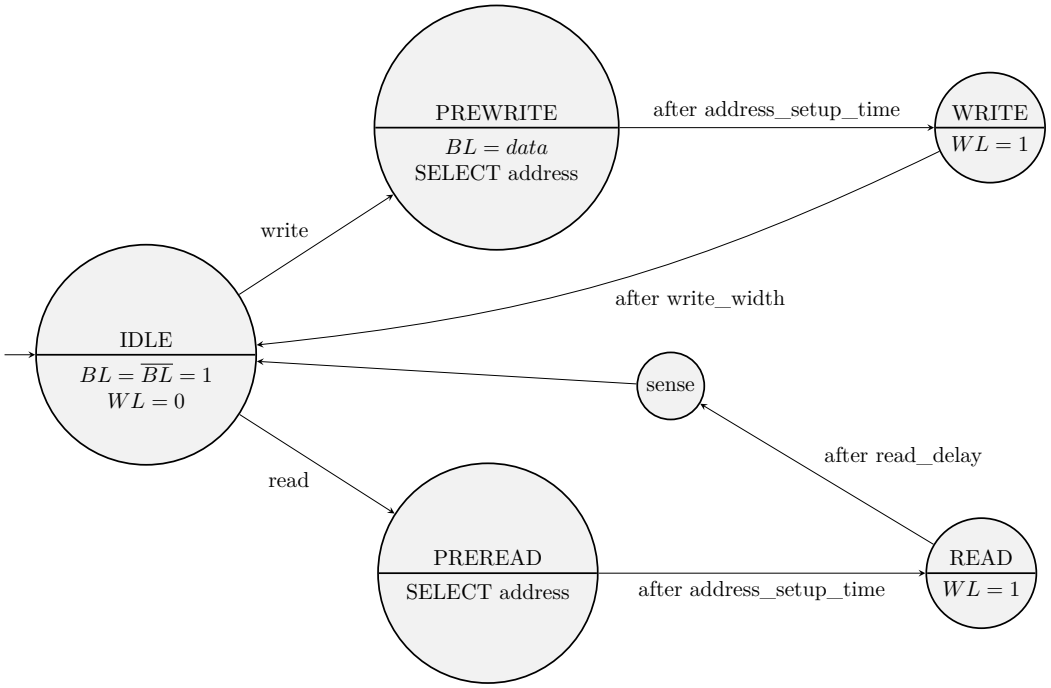
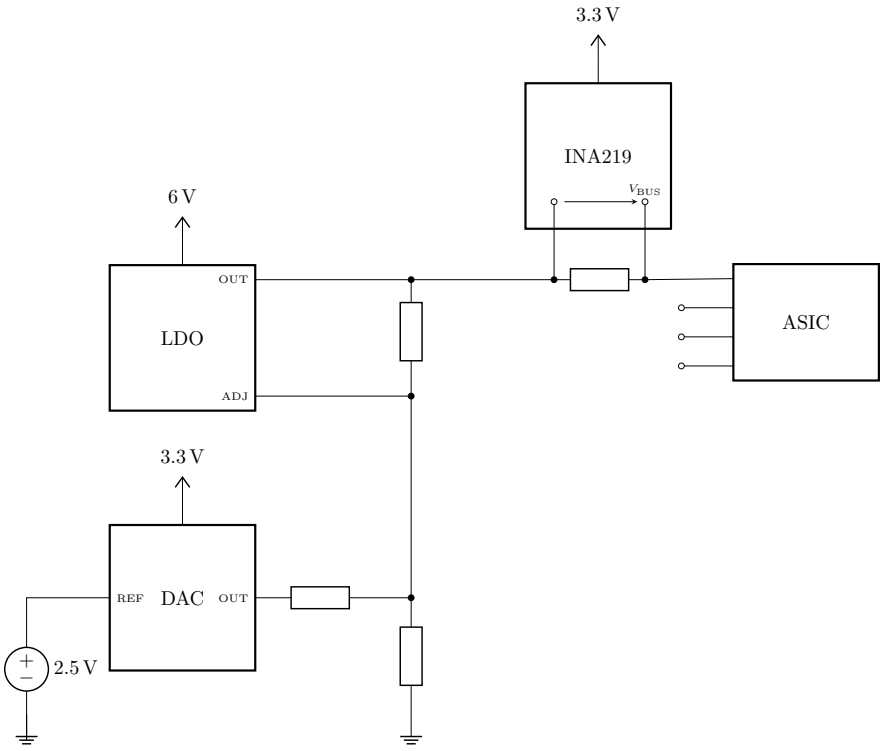


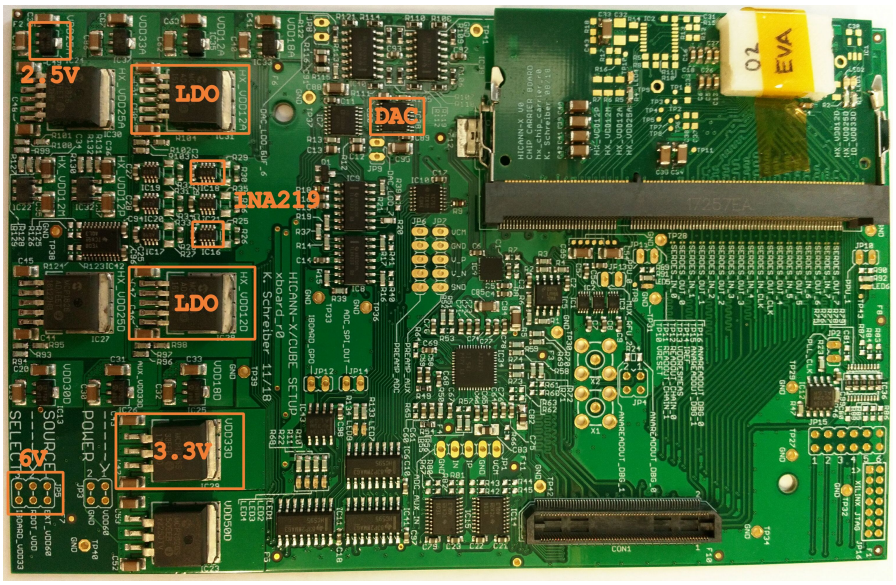
Figure 2.5.: State machine handling the read and write operations of the SRAM on the BSS-2 platform. Each state is held for the duration of the set timings and sets the according signals for the operation to the SRAM.

2.2.2. SRAM supply voltages

The BSS-2 chip is bonded onto a carrier board which itself is connected to a supply board called xBoard which delivers voltage and current to the system. The SRAMs are supplied with nominal 1.2 V by either the digital `vdd12` or the analog `vdd12a` voltage. These are generated using a low-dropout regulators (LDOs) which will adjust their voltage to match a fixed voltage on a adjust input. The feedback loop of the output to the adjust input is a simple voltage divider from output to ground, biased by the output of a digital-to-analog converter (DAC) over a resistor. Using the DAC the output voltage of the LDO can be adjusted this way.
The resulting voltage can be measured using an on-board INA219 chip as the bus voltage V_{BUS} in Figure 2.6a.



(a) Control circuit of the SRAM supply voltage. The LDO outputs a voltage at `OUT`, such that the `ADJ` input is at a fixed refrence voltage. The DAC can then adjust the output voltage by biasing the voltage divider. Using the INA219 the voltage supplied to the SRAM can be measured at V_{BUS}.



(b) The supply board called xBoard of the BSS-2 platform. The ICs involved in the supply voltage of the SRAM are highlighted. Original picture by Korbinian Schreiber.

Figure 2.6.: Supply voltage circuit of the SRAM on the BSS-2 platform.

2.2.3. Software

The BSS-2 system is accessed using an FPGA for sequentially executing instructions on the hardware. These instructions are defined by playback programs which are build using the BSS-2 software stack, which implements an abstraction of the hardware in CPP (Müller et al., 2020):

— **Coordinates**: The *halco* library of the software stack abstracts the structure of the hardware and provides classes for addressing the compontents of the BSS-2 system. See Electronic Visions (2024a) for more information.

— **Containers**: The *haldls* library of the software stack abstracts the data structure and addresses of the hardware and provides classes representing the data of the hardware components for each coordinate. See Electronic Visions (2024b) for more information.

Playback programs are then built using methods for defining write, read or barrier FPGA instructions on the hardware coordinates, with the data encapsulated by the containers. There are further libraries in the software stack which for example handle the compilation of the FPGA instructions or further abstract the hardware. The details of this are not important for the procedure of this thesis.

Chapter 3.

Methods

3.1. SRAM Tests

The basic concept of functional testing is to write a pattern/series of known values to the memory and compare them to subsequent reads, identifying any deviations from the previously written values as faults. Due to the various complex fault behaviors that can arise in n SRAM, there are many possible test patterns, designed to find certain sets of faults (Bosio et al., 2012). In this thesis I used the following two methods for testing:

1. *ValueSweep*: This test writes every possible value to segments of the SRAM, e.g. 8-bit integers, across the entire array. This tests for the ability of each block to hold and read each desired value and detects couple faults within each segment. This test may miss faults dependent on a more complex sequence of values or coupling faults between segments, but is very efficient at ensuring the basic functionality of the memory. The test gets more inefficient the wider the words become, because the test length increases exponentially. The test can be seen as a type of pseudo exhaustive test (Cook et al., 2012).

2. *PseudoRandom*: This test writes a sequence pseudo random values to the memory. In theory this test can find any fault given enough time, but perform worse at finding faults of low complexity compared to deterministic methods (David, Fuentes, and Courtois, 1989). The values are *pseudo* random to make the test repeatable.

It is possible to use these tests during this thesis due to the relative small size of the SRAM considered. For larger memories usually march tests are employed, as they are more efficient in terms of total operations used, but are designed for only specific sets of faults (Bosio et al., 2012). Due to my software implementation in *python*,tests operating on arrays are more efficient (by passing the calculations to C libraries), hence I choose these over the today more common march tests. A more detailed discussion of different patterns can also be found in my internship report (Bergler, 2024). Aside from functional tests there also exist parametric test, such as analyzing the current consumption of the memory (Pavlov and Sachdev, 2008).
To gain an understanding of the behavior of the SRAMs I performed sweeps of the timings of the SRAMs, recording the amount of faults for each configuration. To analyze the behavior I did 2D sweeps of `write_width` and `read_delay` for all setups and repeated them for all `address_setup_time`, recording the amount of faults each cell returned

for each configuration. Each program executed on the hardware contained at most 25 tests for different `read_delay` at fixed `write_width`. Here I used the *ValueSweep* for the CADC SRAM as it showed the same behavior for *PseudoRandom*, but is more efficient. The *PseudoRandom* test was used for the remaining SRAMs as for the neuron configuration and synapse driver SRAMs it was more efficient to implement (due to the software abstraction having no direct mapping of the memory) and for the CapMem the *ValueSweep* pattern did not show all faults.

3.2. Supply Voltage

To analyze the behavior of the SRAM with respect to the supply voltage I performed sweeps of the SRAM timings at different voltages. The supply voltage was changed with the DAC to bias the LDOs adjust input and was measured using the INA219 Texas Instruments, 2015 chip. The voltage is measured before each set of tests run for the sweeps, which is also read using the FPGA. I choose to fix `address_setup_time` as it showed the least impact on the behavior of the SRAM in the previous measurements and varied `read_delay` and `write_width` for the sweeps, to keep the runtime of the tests manageable.
To make reliable assumptions about the measurement of the voltage supplied to the SRAM I measured the supply voltages of the SRAM voltage supply circuit. As we can tell from Figure 2.6a, the interesting voltages are the 6 V xboard supply, the 3.3 V supply of the DAC and INA219, as well as the 2.5 V reference of the DAC. I measured the voltages using the Keithley 2100, which offers a high enough accuracy

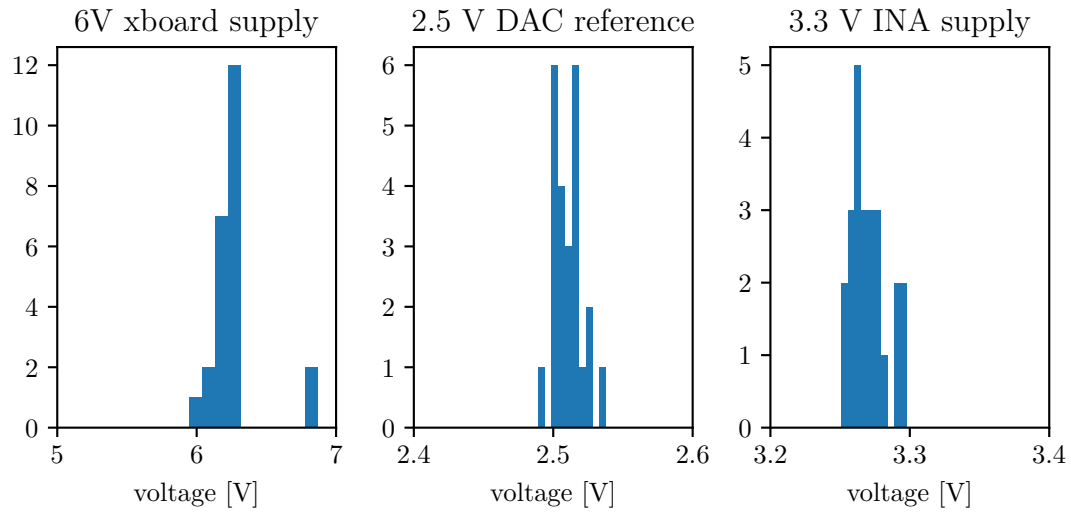


Figure 3.1.: Distribution of the supply voltages measured over 24 setups.

Figure 3.1 shows the distribution of the measured voltages over all available setups. The key takeaways are that the Voltages are on average slightly biased above or below their

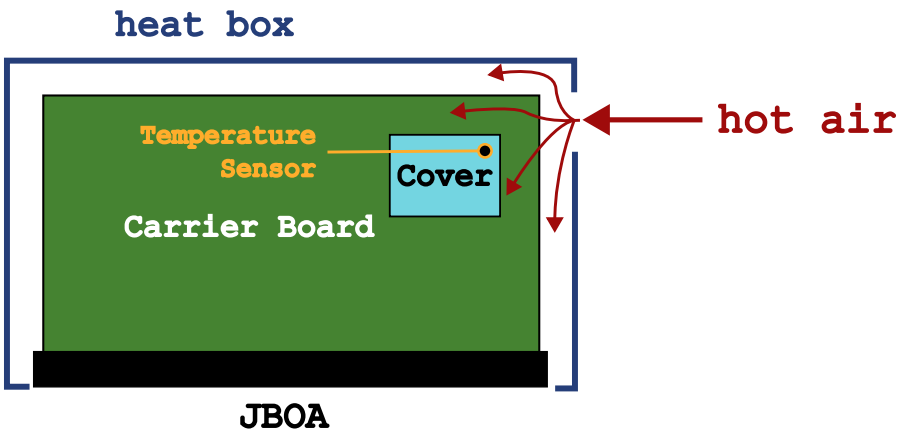
nominal value, but the distribution is relatively narrow, except for the xboard supply which shows singular outliers. The latter seem to be caused by faulty power supplies, which have been replaced after the measurements.
The INA219 chip has the following specifications according to Texas Instruments (2015):

$$\begin{aligned} \text{Offset} &: 5\,\text{mV} \\ \text{Accuracy} &: 4\,\text{mV/LSB} \pm 0.5\% \\ \text{PSRR} &: 10\,\mu\text{V/V} \end{aligned}$$

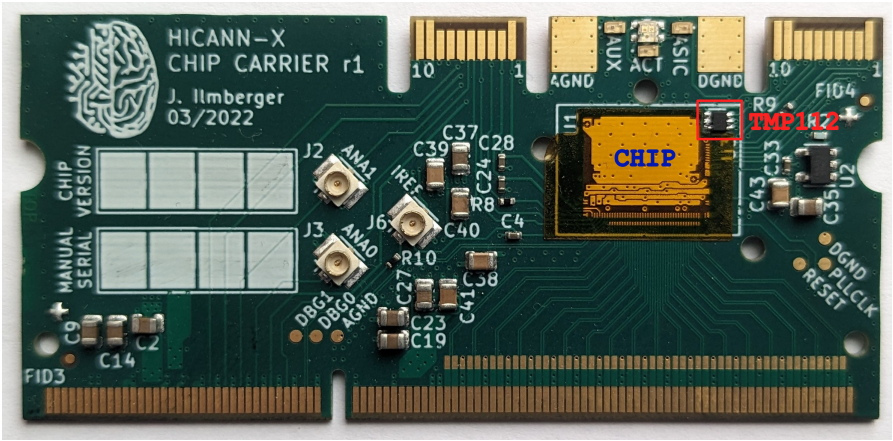
Given that the voltage measured is at most 1.3 V we can expect the error for the measurement to be no larger than 10 mV. in the 10 V range to ignore its error for my purposes. From this I could then estimate the error of the INA219 measurement.

3.3. Temperature

To analyze the behavior of the SRAM with respect to the temperature I, heated the carrier board ASIC of the BSS-2 system using a hot air gun, which by varying the distance and setting allows achieving different temperatures. This way we are not restricted by the temperature limits entire setup and do not need to heat it up entirely. The temperature can be measured using an TMP112 sensor on the carrier board with 0.5 °C accuracy (Texas Instruments, 2024). It is placed under the protective cover of the chip, and therefore should match the temperature of the chip closely. This experiment is carried out on the newer jBOA platform as it allows for easier access to the chip. To ensure better stability of the temperature and thermal equilibrium with the carrier board I also constructed an enclosure fitted over the chip carrier board with an inlet for the hot air. A full schematic of the enclosure can be found in Figure A.1.
Again the temperature is measured before each set of tests run for the sweeps, which is also read using the FPGA. I choose to again fix `address_setup_time` as it showed the least impact and only considered the CADC as it was the most sensitive to the timing configuration in the previous measurements.



(a) Experimental setup for heating the BSS-2 system.



(b) Carrier board of the BSS-2 chip with the TMP112 sensor highlighted.

Figure 3.2.: Temperature sweep setup for the BSS-2 system.

Chapter 4.

Software Implementation

In this chapter, the implementation of the memory tests from Section 2.1.3 for the SRAMs of the BSS-2 platform and the respective implementation of the timing configuration sweeps are described.
Due to the wide range of behaviors and faults of SRAM the aim was to take a unified approach for implementation of the tests, decoupling the actual test methods and patterns from the hardware specific implementation, to allow flexible applications of the tests to different SRAMs and easy implementation of further SRAMs. Furthermore, the unified code paths reduce duplicate code and prevent undetected errors in the implementation of the tests, enhancing the comparability of the results. To achieve this, we implemented a library for testing the SRAMs of the BSS-2 platform using *python* together with *numpy* (Harris et al., 2020) to handle the construction and evaluation of the playback programs for testing. To allow for easy implementation of further memories I used an object-oriented approach, defining abstract classes for the tests and sweeps, which can be implemented for each memory.

4.1. Memory Tests

The memory tests are implemented using the following classes:

— TestPattern: This class defines the sequence of values to be written to the memory as an iterator. It has a shape property used during building, to allow the pattern to be used for any memory shape. But it may also be handled differently in an implementation if the pattern is shape specific. For example, this could be a *ZeroOne* pattern, which returns an array of all 0s and then all 1s in the given shape.

— TestMethod: This class defines the sequence of operations to be performed with one array of data. The possible operations are defined in the Enum *Actions*.

— RamTest: This class represents one instance of a test. It provides a `build` method for constructing a playback program using the given pattern and method. It stores the expected data and the corresponding tickets for the read operations. The provided `evaluate` method is used to retrieve the read data and compare it to the expected data after execution. If a fault is found, the written and read data is passed to a callback function, which can be supplied to `evaluate`.

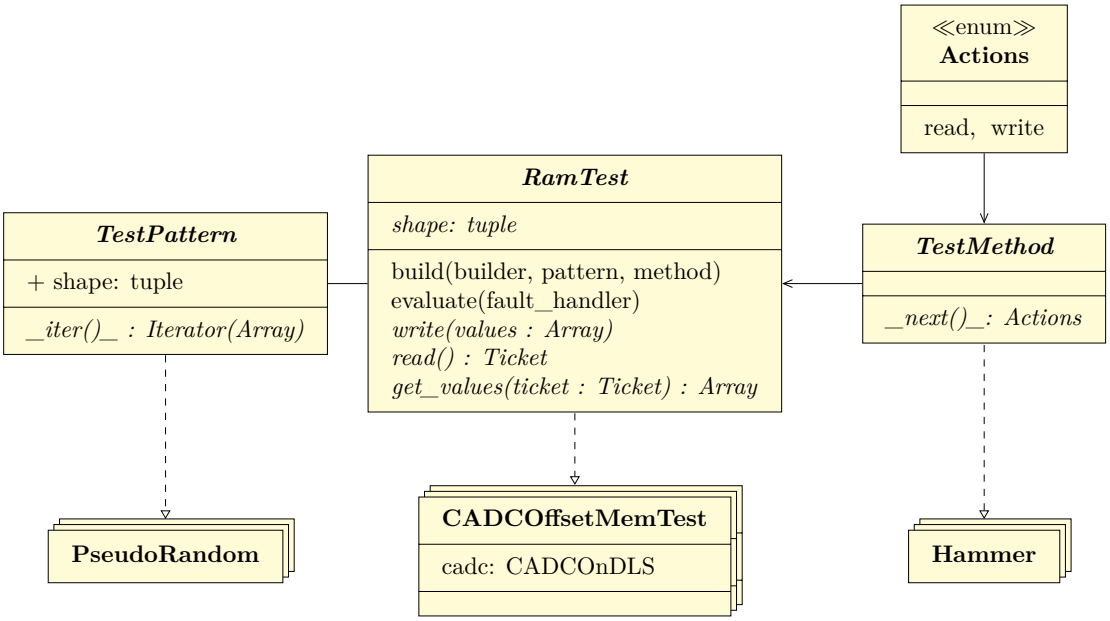


Figure 4.1.: UML diagram of the memory test library with implementations for *RamTest*, *TestPattern* and *TestMethod*.

For a specific SRAM, one then has to implement the abstract interfaces which add the memory specific `write` and `read` operations to the playback program as well as a `get_values` method for retrieving and reconstructing of the values from the tickets. With this structure,new SRAM for can be added for testing without needing to reimplement the patterns and methods, allowing for quick implementation of all SRAMs used in neuromorphic hardware. The methods could also be implemented in other languages and only be exposed to python, making it possible to improve the performance of the tests. This was done for converting between the hardware containers and integer arrays for the CapMem, CADC and synapse driver SRAMs and the methods for adding writes and reads of arrays to a playback program was completely implemented in C++ for the two neuron configurations. To achieve this, the fields of the configuration containers are mapped to an integer in the order defined by the SRAM and then assigned to an array, see Figure 4.2. This way we can achieve a representation of the SRAM where, excluding unused bits, the SRAM cells are represented by the bits of the integers, and can be treated like any other integer SRAM. The downside of this approach is, that this leads to very wide cells in the array, making it unsuitable for doing for example a simple sweep of the values, although one could still, e.g. in a pattern, divide the cells into smaller segments and then sweep these segments instead.

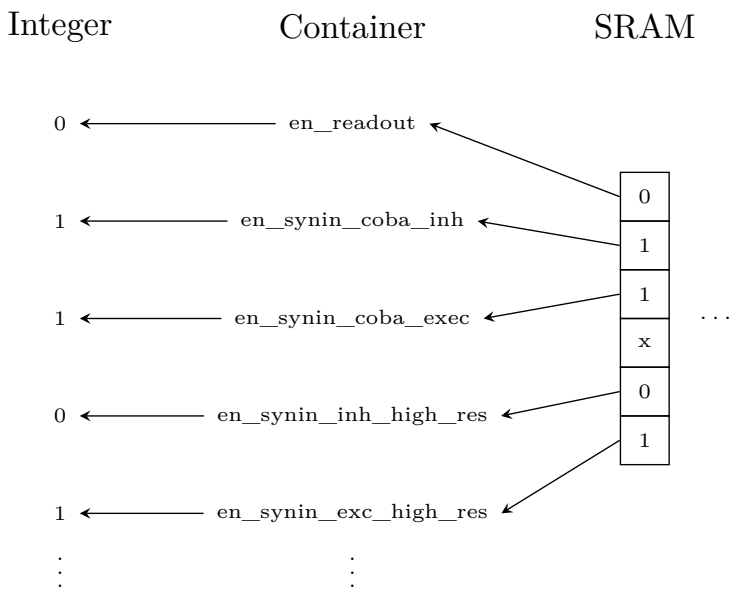


Figure 4.2.: Mapping of the fields of the configurations to the integer array. The SRAM of each configuration is mapped over the container into an integer number where unused bits are skipped. Each complete configuration then represented as a field within the integer arrays of a test pattern.

4.2. Sweeps

To further our understanding of the SRAM, we are interested how the timings of the SRAMs impact the reliability. The implementation of the sweeps is done similarly to the *RamTest* class, with an abstract *SRAMTimingSweep* class. This class defines the universal functionality of building and running sweeps and method interfaces for creating the hardware specific tests and setting the timings. Furthermore, *SRAMTimingSweep* provides a standard method for plotting the data assuming the timing parameters are ordered, which may be overwritten if the timings are ordered differently (e.g., SynRam where the configuration is a set of bitflags activating equivalent transistors, see Bergler (2024)).
Here, we make use of the abstraction of *RamTest*, making the implementation of further memories again relatively easy. The entire library is organized in modules containing the specific implementations of *RamTest* and *SRAMTimingSweep* for each memory, reflecting the structure of the BSS-2 hardware and making the SRAMs indepedent of each other.
The implementations again use the hardware containers and coordinates for the timing configurations defined in the BSS-2 software stack. As part of this work, the CapMem timing configuration containers and coordinates were implemented and commissioned, as it was missing from the software stack.

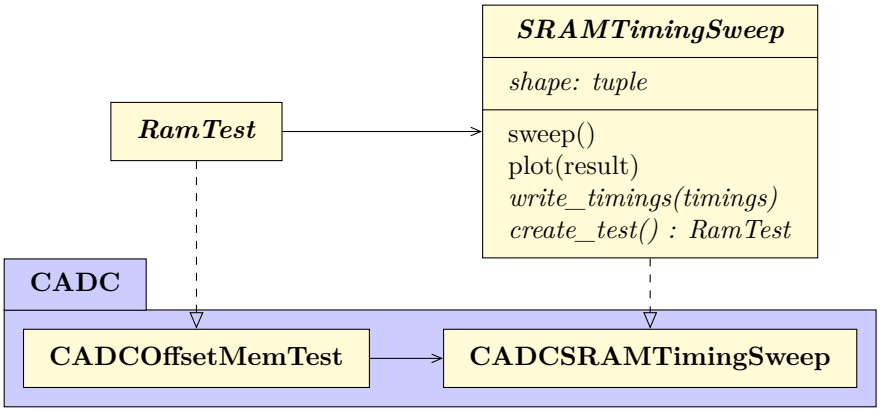
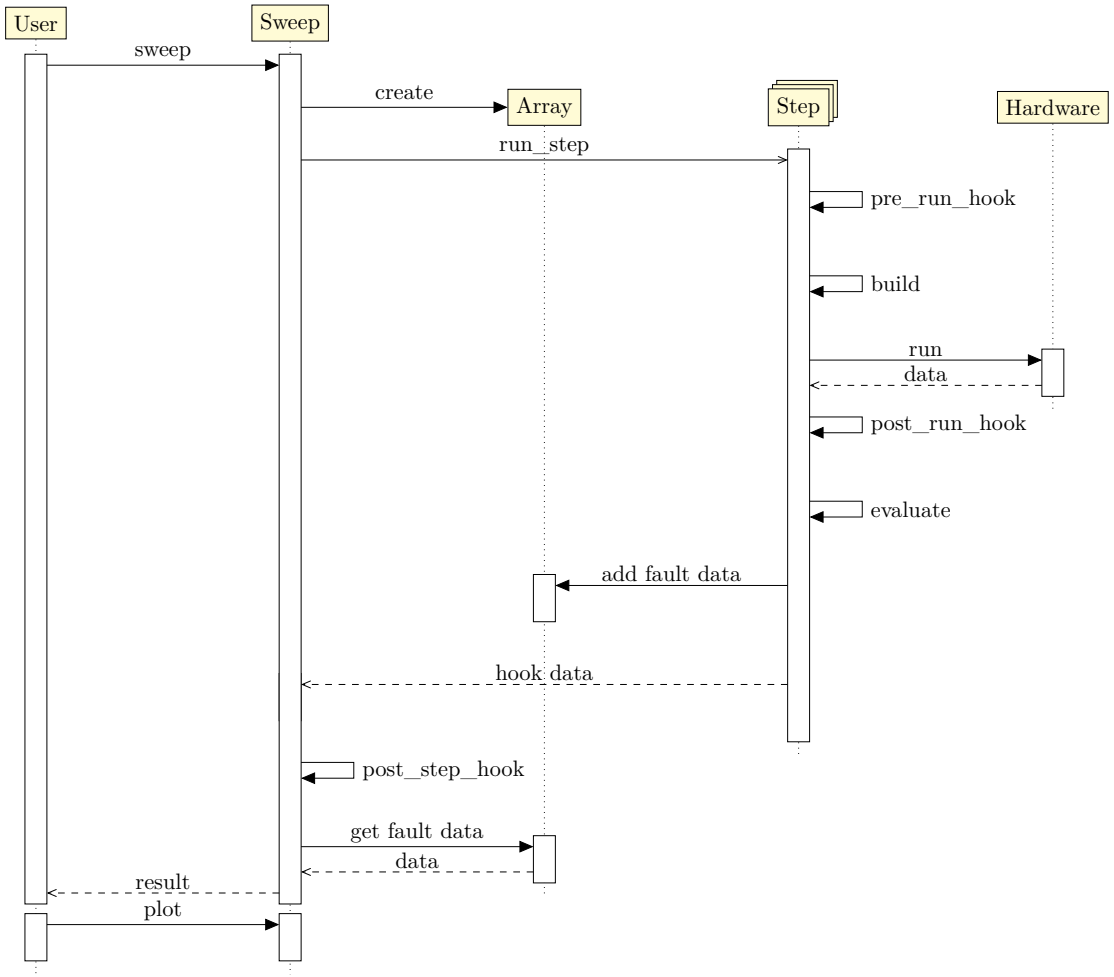


Figure 4.3.: UML diagram of the *SRAMTimingSweep* class with the implementations for one memory organized into their own submodule.

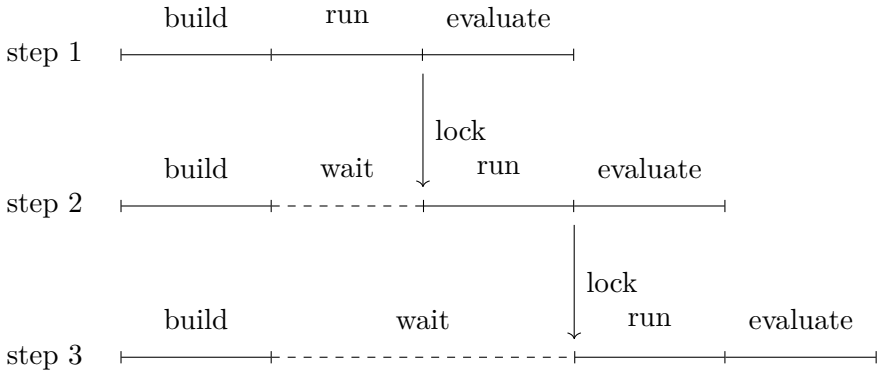
The *SRAMTimingSweep* class also provides functions to add hooks before the build and after the execution to make it possible to add operations before the test, which can be for example used to set or read other parameters of the system (e.g, supply voltage). This way this library is not restricted to the timings of the SRAMs and can be used for other sweeps as well.
To make the sweeps more efficient, *SRAMTimingSweep* runs as many tests in one playback program as possible and builds and evaluates these programs in parallel to maximize the usage of the hardware, see Figure 4.4a. The results from the test are stored in a shared memory array, to avoid passing big arrays between the processes. The implementation relies on the *multiprocessing* library of python. The hardware can only execute one playback program at a time, hence the steps are synchronized using a lock for access to the hardware, see Figure 4.4b. With these optimizations and the implementations in CPP for *RamTest*, I was able to speed up the sweeps, see Table 4.1.

SRAM	arrays/config	without	parallel	parallel + CPP	speedup factor
CADC	256	87 min	57 min	20 min	4.3
CapMem	100	–	–	117 min	–
digital neuron	10	40 min	36 min	3 min	8
analog neuron	10	216 min	42 min	5 min	42
synapse driver	500	433 min	260 min	43 min	10
SynRam	64	75 min	53 min	–	1.4

Table 4.1.: Speedup of the `read_delay` and `write_width` sweeps at `address_setup_time` 15 for the given amount of arrays per configuration. Missing times are due to too long to run times for the CapMem. For the SynRam an CPP implementation was already provided by *haldls* and is applied for all given times.



(a) One sweep spawns multiple `Step` subprocesses used for building, executing and evaluating the programs in parallel with hooks for adding further operations. The test results are stored in a shared memory array to avoid passing big arrays between the processes.



(b) As the hardware can only execute one playback program at a time, the `Step` processes are synchronized using a lock for access to the hardware.

Figure 4.4.: Sequence diagram of the *SRAMTimingSweep* class.

Chapter 5.

Results

5.1. SRAM Timing and Faults

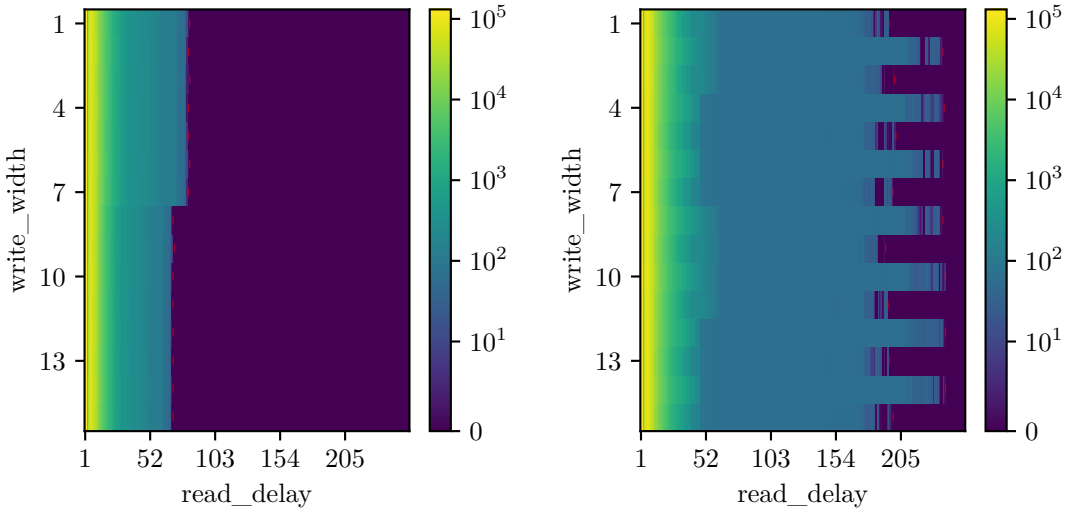
In this section the results of sweeping the SRAM timing configurations from Section 2.2.1 performed on the BSS-2 platform are described and analyzed.

5.1.1. CADC

For the first step of the analysis we consider the behavior of `read_delay` and `write_width` for fixed `address_setup_time`. We can fix `address_setup_time` at the maximum value, because in theory we don't expect interference from long setup times. For the CADC SRAM, it was possible to use the *ValueSweep* test pattern, as it performed equally well as the *PseudoRandom* test pattern and the mapping of the SRAM cells to small integers was already given.
In Figure 5.1, two examples of the sweep of the CADC SRAM are shown, with "good" and "bad" behavior. The first observation is that a substantial amount of possible configurations are faulty, but with increasing `read_delay` the amount of faults decreases until it reaches a critical `read_delay` setting after which the SRAMs is fault free. In Figure 5.1a this edge shows only a small decrease with increasing `write_width`. In Figure 5.1b on the other hand additionally to the previous behavior, there is an extended area of low faults ($\approx 10^2$), but much higher needed `read_delay` to be fault free and more variation along `write_width` in an alternating pattern. Furthermore, there is a strip of lower amount of faults stretching from `read_delay` 3 to 4, which in theory should not be possible, as the faults in general should on average decrease with increasing `read_delay`, but here we see a significant increase after `read_delay` 4 again. Further investigations showed that these faults are only dependent on the `read_delay` and did not alter the state of the cell, therefore we see incorrect read faults. For the slow singular channels in particular these faults returned "0" instead of "1" if another bit was set to "1", hence this is an instance of a coupled incorrect read fault, which also explains by the amount of 64 faults these channels returned constantly for the large areas of low faults.
From these observations we can conclude:

— The timings can be too fast for the CADC, but there are functional timings for almost all setups

— `read_delay` has the strongest impact on the amount of faults



(a) setup W61F3 north SRAM

(b) setup W61F0 south SRAM

Figure 5.1.: Number of faults found per timing configuration for the CADC SRAM with maximal `address_setup_time` for one "good" and "bad" CADC SRAM with a red line indicating the beginning of the fault free configurations. The colorbar is adjusted to the maximum of faults that can occur for the test.

— `write_width` shows some variation, in that for some SRAMs the edge moves to lower `read_delay`, but other show an alternating pattern with much more amplitude than this effect. Sometimes neither of the effects are visible. If the `read_delay` is high enough, the SRAM is fault free for all `write_width`.

— Approaching the fault free settings of the amount faults decrease. This is probably due to transistor parameter variation and increasing SNM.

— Sometimes there is a low, possibly singular, amount of cells that are significantly slower than the majority of the cells, which create faults for much longer timings.

— There is an anomaly around `read_delay` 3 to 4, where the amount of faults is unexpectedly lower than the trend would suggest.

To extend these observations to all tested systems, now the highest faulty `read_delay` for each `write_width` is considered, this parameter measures the minimal timing for which the SRAM is fault free for all slower settings and indicates the beginning of the "edge" of the faulty timings. This parameter is enough to understand the performance of an SRAM for the CADC as the other timings show no impact for high enough `read_delay`. Optimally, one wants to choose the configuration of the SRAM larger than this value to ensure the memory is fault free under varying conditions. To give an overview of the required timings for different SRAMs we consider the minimum of the highest faulty

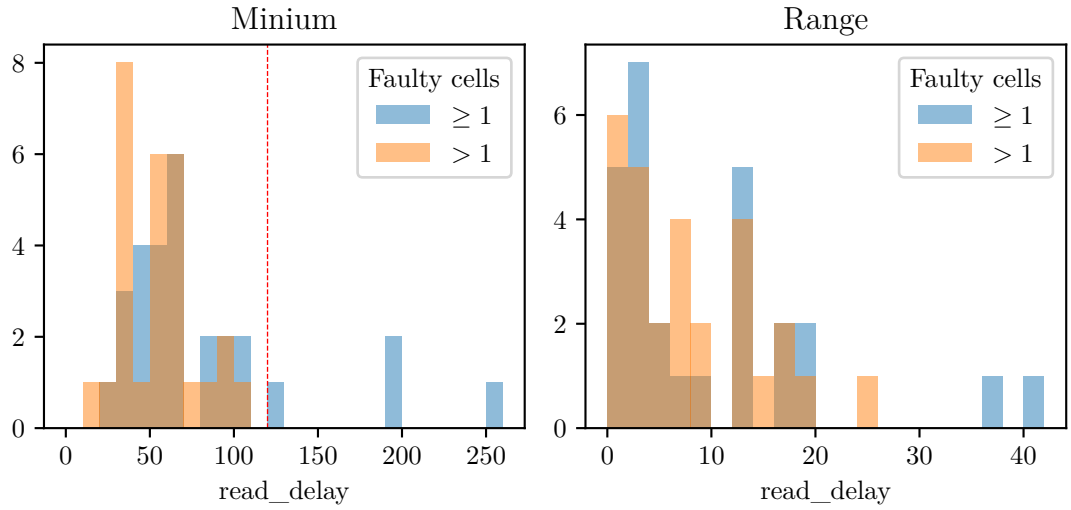


Figure 5.2.: Distribution over 28 CADC SRAMs of the the highest faulty `read_delay` settings along `write_width`. Per SRAM the minimum and range of the highest faulty `read_delay` settings are considered. The red line indicates the standard `read_delay` before this thesis.

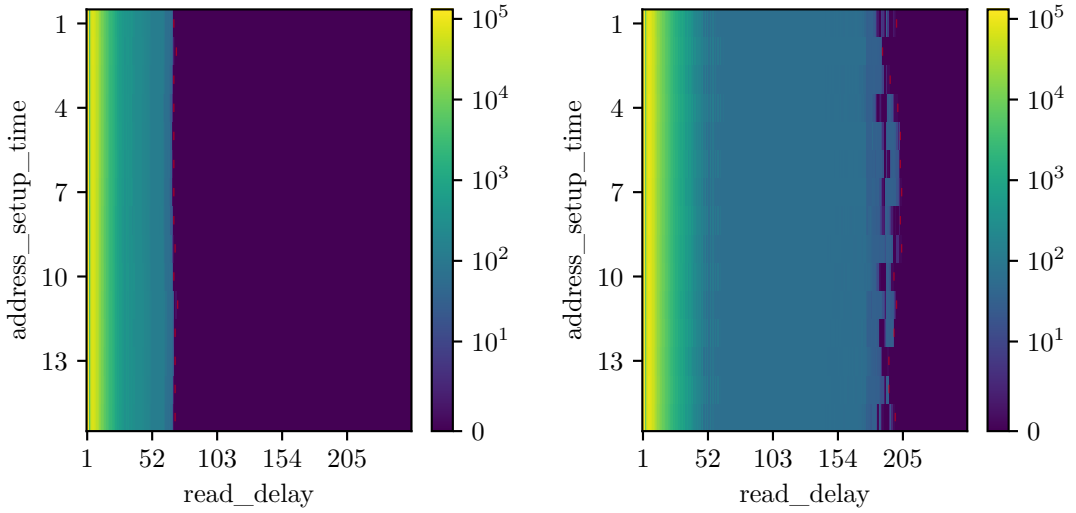
`read_delay` along `write_width` and the range of these highest faulty `read_delay` between the `write_width` timings. This shows how the SRAMs performs in total and how large the variation along `write_width` is. The second to last observation can be tested by ignoring the timings, where only one CADC channel returned faults. If true, this should remove the large areas of low faults and remove any significant deviation from the general distribution of the systems. This analysis ignores the behavior for too fast `read_delay` settings, but they are not relevant for the reliability of the SRAM.
Figure 5.2 shows the distributions of the considered metrics for the CADC SRAM over all tested setups. From this we can conclude:

— Most systems are fault free for `read_delay` higher than 120. But some systems are significantly slower.

— The variation along `write_width` is mostly contained below 20, which is relatively small with respect to the parameter space and typical cutoff. In the left histogram this variation would only make a difference of 2 bins.

— Slow singular cells limit the speed of the SRAM substantially and show greater variation along `write_width`.

— ignoring single faulty channels removes large deviations from the general distribution. Hence, these seem to stem only from singular slow cells.



(a) setup W61F3 north SRAM

(b) setup W61F0 south SRAM

Figure 5.3.: Number of faults found per timing configuration for the CADC SRAM with maximal `write_width` for one "good" and "bad" SRAM with a red line indicating the beginning of the fault free configurations. The colorbar is adjusted to the maximum of faults that can occur for the test.

These observations show that the CADC SRAM is sensitive to the configuration of `read_delay` and to some extent also `write_width`. Thus, these configurations have to be considered when commissioning new BSS-2 chips or at least be chosen as slow as possible per default. Especially so, as there are slow cells that need substantially longer `read_delay` than rest of the memory. During the tests there was even one system where for one cell no `read_delay` was long enough for any `write_width` (which can be fixed by choosing a higher voltage, as we will see in Section 5.2).
To analyze the impact of `address_setup_time` instead `write_width` is fixed at the maximum value and the same metrics are considered but along `address_setup_time`. Fixing `write_width` is permitted as the previous analysis showed only a small impact on the behavior of the SRAM. In Figure 5.3 the same SRAMs are shown, but with `write_width` fixed at the maximum value. The behavior is similar to the previous sweeps, but the impact of `address_setup_time` is even smaller. Again `address_setup_time` is not critical for the reliability of the SRAM, as choosing `read_delay` large enough will make the SRAM fault free for all `address_setup_time`. The slow singular cells are also visible in this plot and seem to have an similar impact, thus they also mainly depend on the `read_delay` timing. Lastly, the anomaly around `read_delay` 3 to 4 is also visible in this plot, indicating it is also only dependent on the `read_delay` timing. These observations prompt us to consider the highest faulty `read_delay` along `address_setup_time` with the same statistics as before.

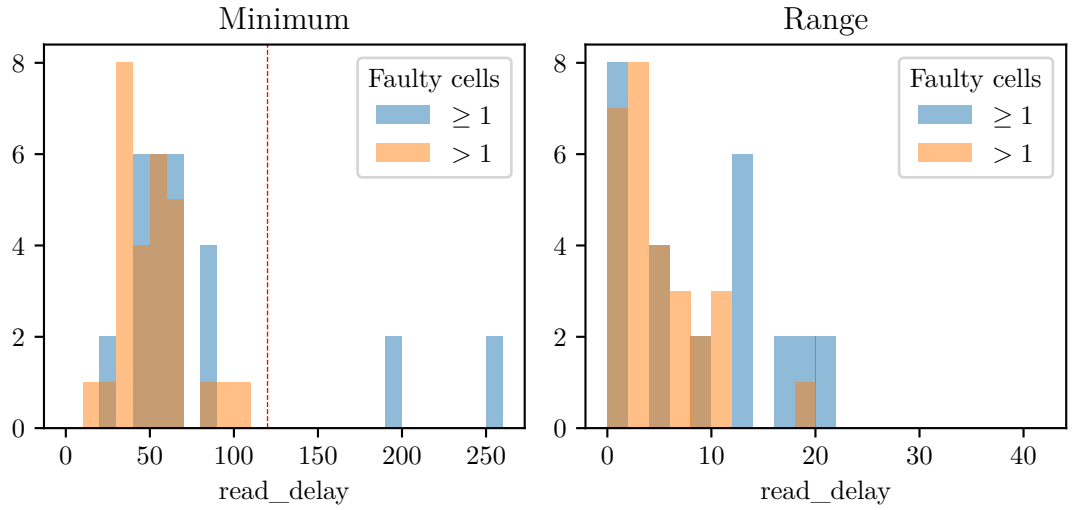
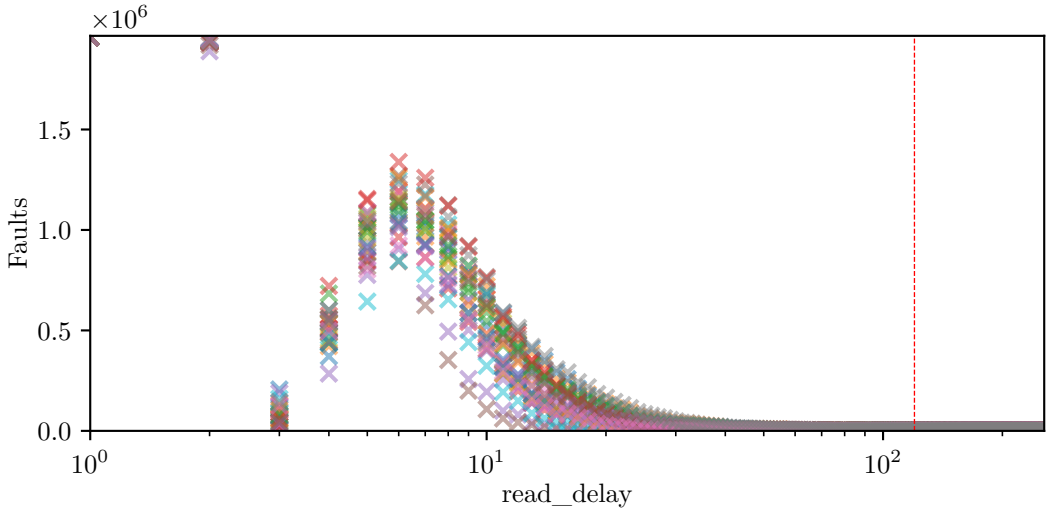
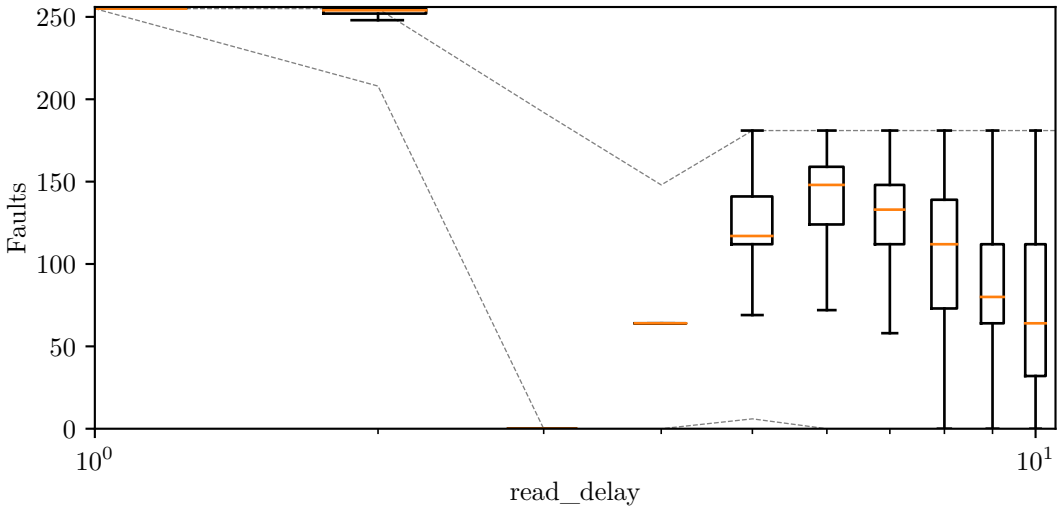


Figure 5.4.: Distribution over 28 CADC SRAM of the of the highest faulty `read_delay` settings along `address_setup_time`. Per SRAM the minimum and range of the highest faulty `read_delay` settings are considered. The red line indicates the standard `read_delay` before this thesis.

In Figure 5.4 one can see that along `address_setup_time` the highest faulty `read_delay` has an even smaller variation than along `write_width` (mostly only 1 bin in the left histogram). Again singular slow cells cause significantly slower behavior and more variation for some systems, of which we see more than along `write_width` (probably because of an alternating pattern being particularly high at 15), hence `write_width` has more impact for finding the correct configuration.
From comparing the distributions Figure 5.2 and Figure 5.4 we can see, that `address_setup_time` has less impact than `write_width` because the range of the highest `read_delay` is on average lower and the singular slow cells remain hat higher `read_delay` timings.
To further investigate the anomaly of lower faults at `read_delay` 3 we consider the behavior of the faults along only `read_delay` timing for the CADC SRAM. This is permitted because, as we have seen in the previous analysis, the other two timings are not critical for the reliability of the SRAM and only have a weak impact on the behavior. For this the amount of faults is summed over all `write_width` considered at maximal `address_setup_time`.
As can be seen in Figure 5.5a, all SRAMs show high a amount of faults for `read_delay` 1 and 2 and then unexpectedly drop to a very low amount, to then gradually increase again for higher `read_delay` until it starts to decrease again for high `read_delay`. We can see that the SRAMs are not fault free for `read_delay` 3 and 4, hence this effect does not matter for the correct configuration of the SRAM. The rate of decrease of the amount of faults for high `read_delay` is similar for all SRAMs but with different rates

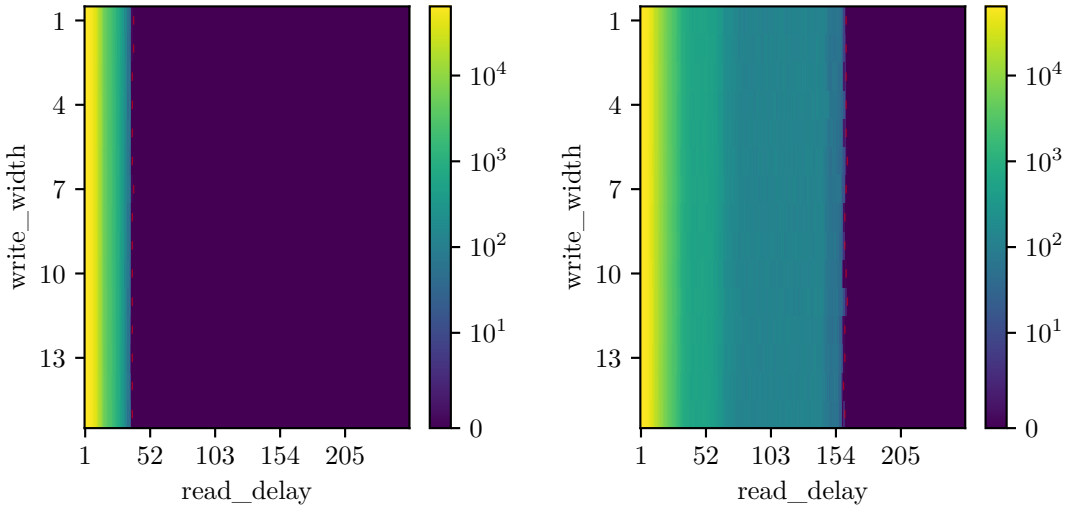


(a) Number of faults found per `read_delay` for the CADC SRAM summed over all `write_width` and at maximal `address_setup_time` for 28 SRAMs. The red line indicates the standard `read_delay` before this thesis.



(b) Distribution of the amount of faults along `read_delay` for 28 setups over all CADC channels with `write_width` and `address_setup_time` fixed at the maximum value. The dashed lines indicate the maximum and minimum faults observed. The plot is cutoff at `read_delay` 10, working configurations can still be found, but at higher settings not shown in the plot.

Figure 5.5.: Analysis of the faults along `read_delay` for the CADC SRAM.



(a) setup W66F0 north SRAM

(b) setup W63F0 north SRAM

Figure 5.6.: Number of faults found per timing configuration for the synapse driver configuration SRAM with maximal `address_setup_time` for one "good" and "bad" SRAM with a red line indicating the beginning of the fault free configurations. The colorbar is adjusted to the maximum number of faults that can occur for the test.

depending on the setup, until they all reach their highest faulty `read_delay`. To see if the increase of faults after the anomaly at `read_delay` 3 is systematic or just shows the distribution of the SRAM cells that are also fault free for `read_delay` 4 (as we only see the sum over the entire SRAM), we now consider the distribution of the faults along `read_delay` per CADC channel of all setups. In Figure 5.5b boxplots for each of the `read_delay` settings are shown. The boxplots show that the increase after `read_delay` 4 is in fact systematic as the quantiles are effectively identical to the median, thus the majority of the channels show similar behavior at his point.

5.1.2. Synapse Driver

For the synapse driver config SRAM the same evaluation as before is repeated. Here the *PseudoRandom* test pattern was used due to the mapping of the memory employed in software, which does not allow for efficient use of the *ValueSweep* pattern. In Figure 5.6 one particularly "good" and one "bad" behaving SRAM is shown. The result from the sweeps are similar to Figure 5.1 but with less variation along `write_width`. In Figure 5.6a the highest faulty `read_delay` is relatively low, while in Figure 5.6b, similarly to the CADC, there is an additional area of lower faults with a higher `read_delay` cutoff. Here we see no anomaly around `read_delay` 3 to 4, which makes the CADC unique in this regard.

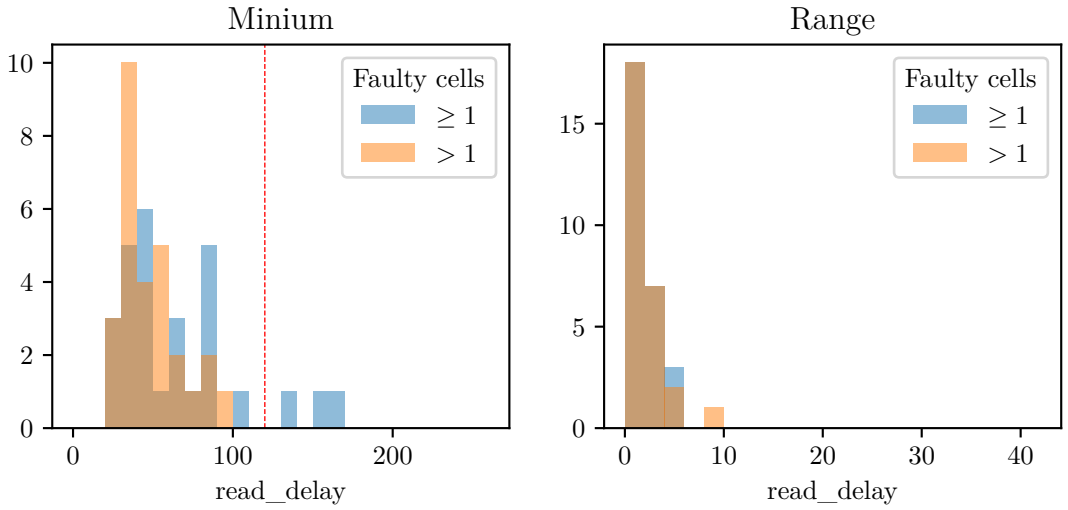


Figure 5.7.: Distribution over 28 synapse driver SRAMs of the of the highest faulty `read_delay` along `write_width` at maximal `address_setup_time`. Per SRAM the minimum and range of the highest faulty `read_delay` settings are considered. The red line indicates the standard `read_delay` before this thesis.

From the observations of the sweeps we can conclude:

— The timings can be too fast for the synapse driver configuration, but there are functional timings for all setups

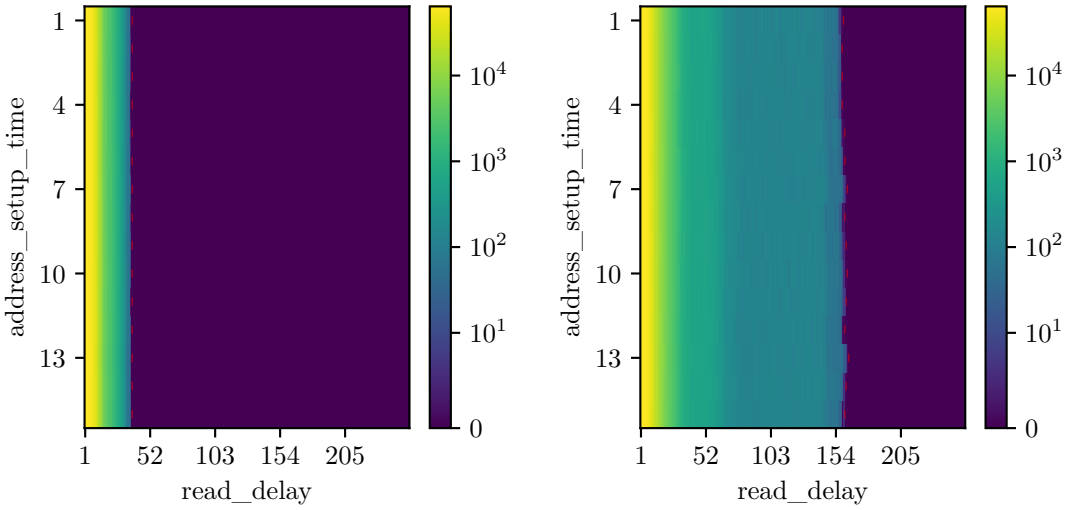
— The highest faulty `read_delay` is typically lower for setups without slow singular cells compared to the CADC

— The variation along `write_width` is relatively small compared to the CADC

— Slow singular cells are present, but seem to have less impact than for the CADC and don't return a constant amount of faults. Again we see a coupled incorrect read fault for these cells, but sometimes with two bits instead of one coupling bit set high for the fault to occur.

These observations indicate that the synapse driver configuration SRAM is slightly faster with respect to the timings as the `read_delay` cutoff is lower and the variation along `write_width` is smaller.
Considering the distributions of the highest faulty `read_delay` along `write_width` with and without singular faulty cells in Figure 5.7 we can conclude:

— `read_delay` can be too short, but the minimum for most systems is below 120. So far no setup has occurred where no timing is long enough for the synapse driver SRAM.



(a) setup W66F0 north SRAM

(b) setup W63F0 north SRAM

Figure 5.8.: Number of faults found per timing configuration for the synapse driver SRAM with maximal `write_width` with a red line indicating the begin of the fault free configurations. The colorbar is adjusted to the maximum of faults that can occur for the test.

— The variation along `write_width` is relatively small. Making up at most 1 bin difference in Figure 5.7.

— There are singular slow cells that need higher `read_delay` timings to work than the majority of the cells. The variation along `write_width` is not significantly affected by them.

For analysis of the `address_setup_time` again `write_width` is fixed at the maximum value, which is justified by the previous analysis. In Figure 5.8 we see the same SRAMs as before, but with `write_width` fixed at the maximum value. We see in fact a very similar behavior to the `write_width` sweeps, indicating that for the synapse driver both configurations are not critical for the reliability of the SRAM. The slow singular cells are also visible in this plot and seem to have a similar impact, thus they also mainly depend on the `read_delay` timing.
The histograms of the highest faulty `read_delay` along `address_setup_time` are shown in Figure 5.9. Here we see, that the variation along `address_setup_time` is relatively small (1 bin). Fixing `write_width` instead also causes more SRAMs to need `read_delay` above 120, hence `write_width` is more important for finding the correct configuration of the SRAM. The conclusion that `write_width` has more impact is less strong here, because the distributions are quite similar, but again we see that fixing `write_width` causes more setups to be broken by singular slow cells.

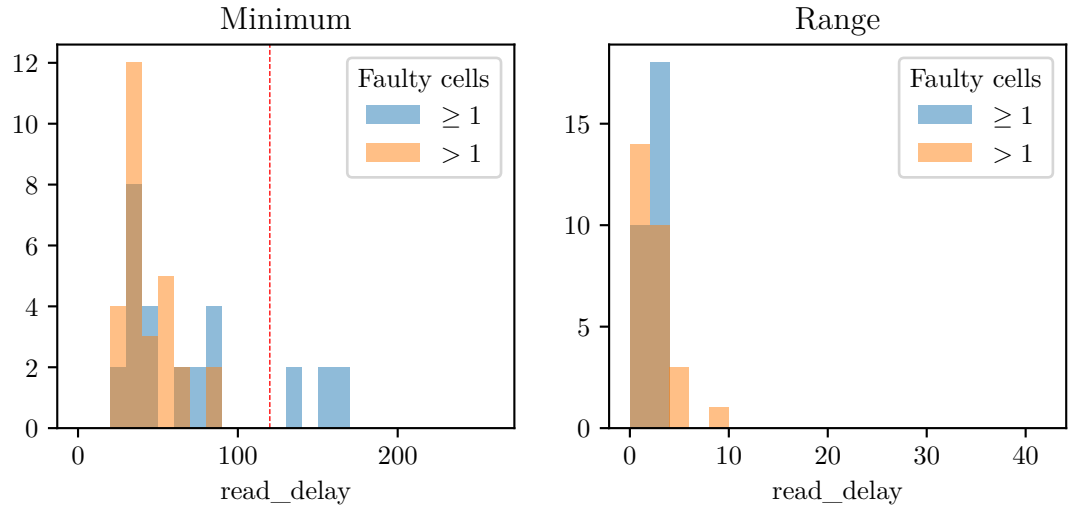


Figure 5.9.: Distribution over 28 synapse driver SRAMs of the the highest faulty `read_delay` along `address_setup_time`. Per SRAM the minimum and range of the highest faulty `read_delay` settings are considered. The red line indicates the standard `read_delay` before this thesis.

Last we consider the behavior of the faults along only `read_delay` for the synapse driver configuration SRAM, as we, like for the CADC, have seen that the other timings are not critical for the reliability of the SRAM.
In Figure 5.10 the amount of faults along `read_delay` summed over all `write_width` at maximum `address_setup_time` for the synapse driver SRAM is shown. Here we see exactly the behavior one would expect as the SRAMs starts at a high amount of faults for low `read_delay` and with increasing `read_delay` the faults monotonously decrease until the highest faulty `read_delay` is reached. This matches with the structure we see in Figure 5.6. Most importantly we do not see an anomaly around `read_delay` 3 to 4.

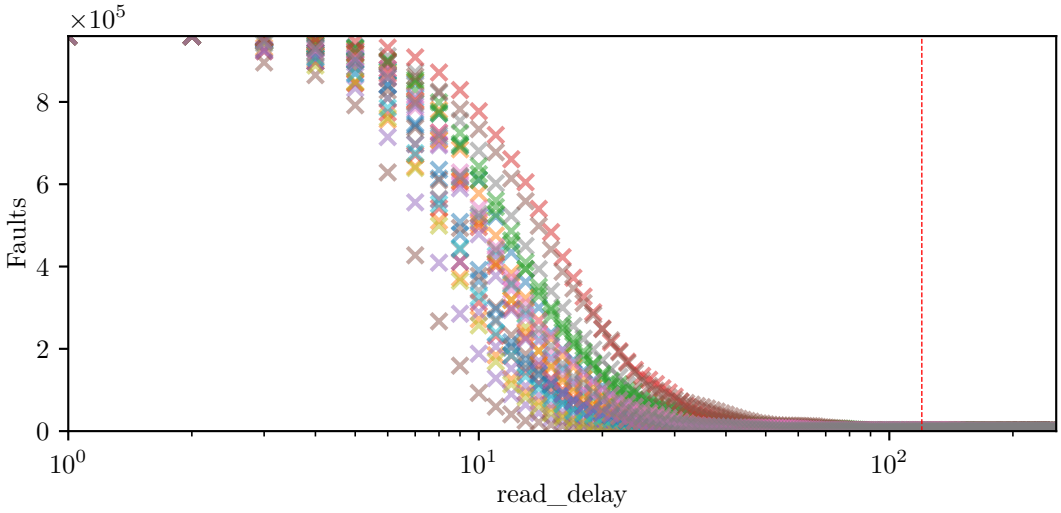
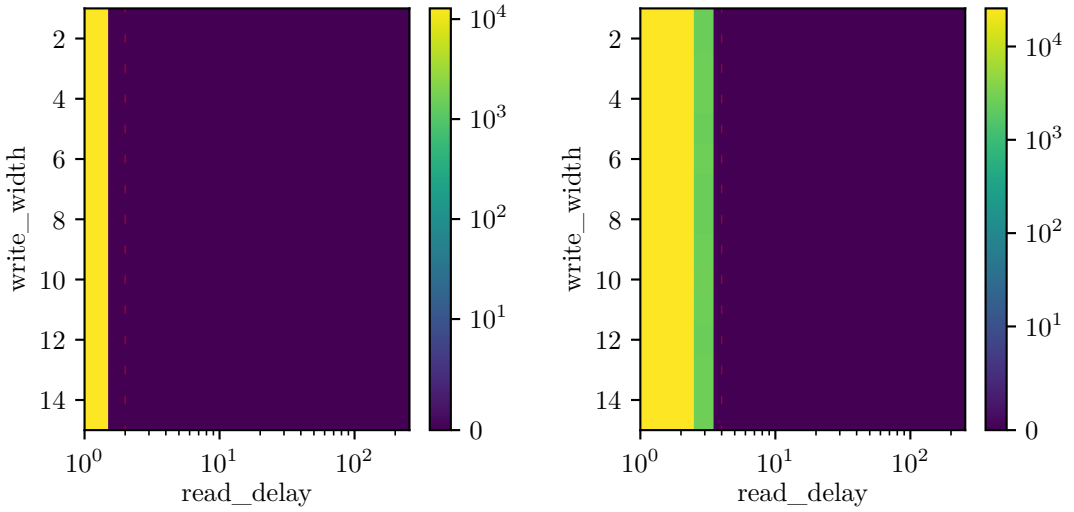


Figure 5.10.: Number of faults found per `read_delay` for the synapse driver configuration SRAM summed over all `write_width` and at maximal `address_setup_time` for 28 SRAMs. The red line indicates the standard `read_delay` before this thesis.

5.1.3. Analog neuron configuration

Continuing with the analog neuron configuration, we again first fix `address_setup_time` at the maximum value and consider the sum of faults over the entire SRAM. In Figure 5.11a we can see that the SRAM shows practically ideal behavior, as it is fault free for any timing with `read_delay` greater 1. As a matter of fact the sweep looks exactly identical for all setups. The `address_setup_time` sweeps look identical and are also the same for all setups, therfore the plots will be omitted here. We can not determine if the anomaly at low `read_delay` would occur here as we see already no faults for lower `read_delay`.
With both these observations we can again consider the same statistic as before, the highest faulty `read_delay` along `write_width` and `address_setup_time` with other respectively fixed at the maximal setting. Considering the distribution of the highest faulty `read_delay` over the setups in Figure 5.12a, all setups are equal in their behavior along `read_delay` and have no variation along `write_width`. The statistics along `address_setup_time` are identical as Figure 5.12b shows. With this we can conclude that the digital neuron configuration SRAM is functional for any `read_delay` timing greater 1, and is not sensitive to `write_width` nor `address_setup_time`.



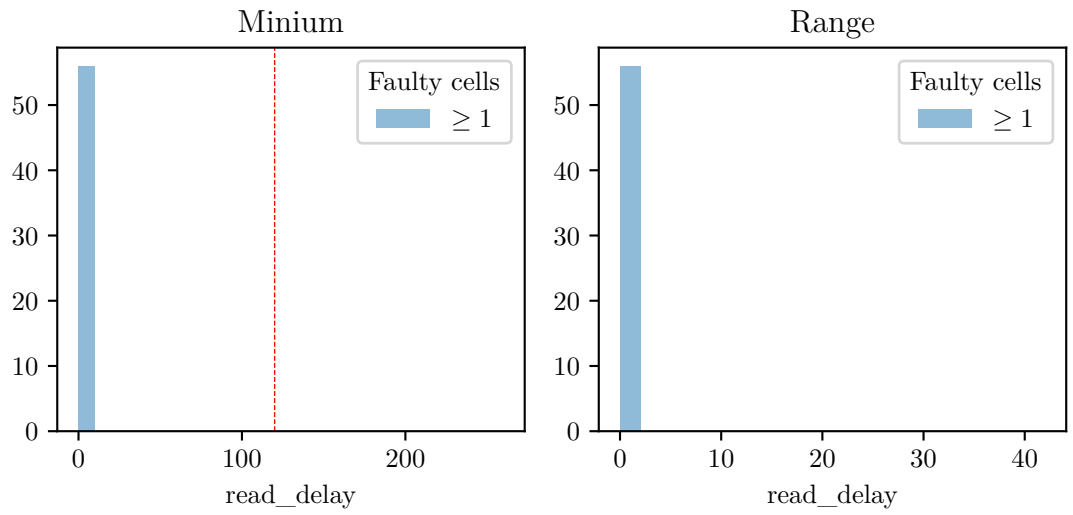
(a) analog neuron configuration block 0

(b) digital neuron configuration north

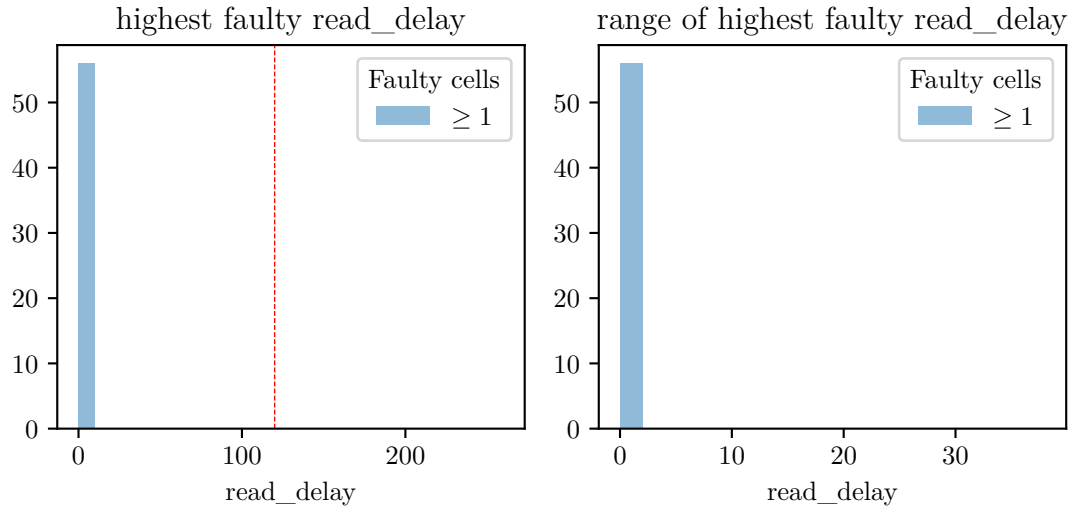
Figure 5.11.: Number of faults found per timing configuration for the neuron configuration SRAMs with maximal `address_setup_time` on setup W61F0 with a red line indicating the beginning of the fault free configurations. The colorbar is adjusted to the maximum number of faults that can occur for the test.

5.1.4. Digital neuron config

Repeating the analysis for the digital neuron config, in Figure 5.11b we see an example sweep with similar behavior to the analog neuron configuration, with the only difference being that the SRAM is only fault free for `read_delay` greater 3 instead of 1 and has a small transition region with a slightly reduced amount of faults. The transition region could also be of the same origin as the anomaly observed for the CADC, but we can not distinguish this as the SRAM transitions into fault free behavior for higher `read_delay` timings at this point. Again the sweep looks the same for all setups and the `address_setup_time` sweeps look identical.
Once again considering the statistic of the highest fault free `read_delay` along `write_width` and `address_setup_time` in an analogous analysis to the digital neuron config in Figure 5.13a and Figure 5.13b, we can see that again the metrics are equal for all systems and are fault free for `read_delay` greater 3 with no variation along `write_width` nor `address_setup_time`.

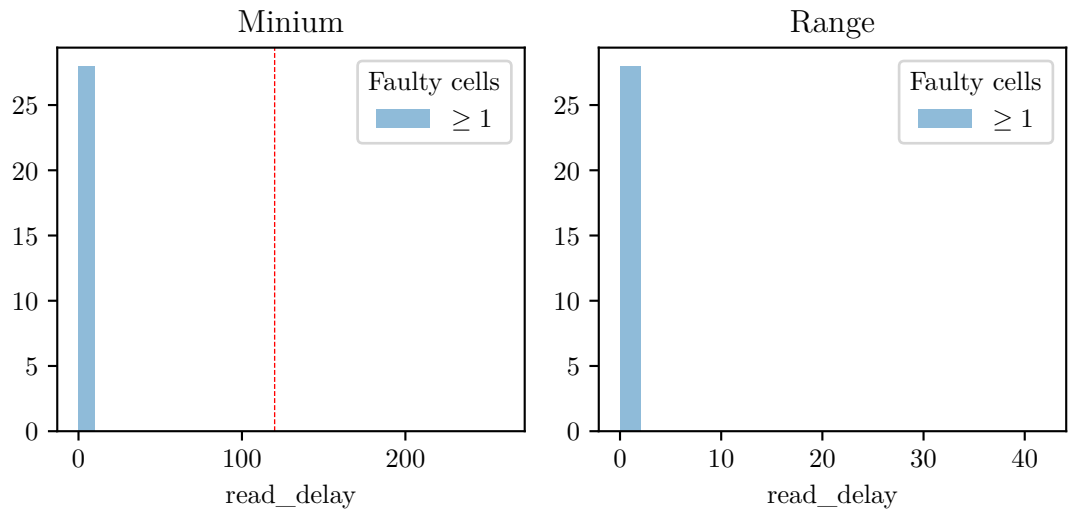


(a) `read_delay`

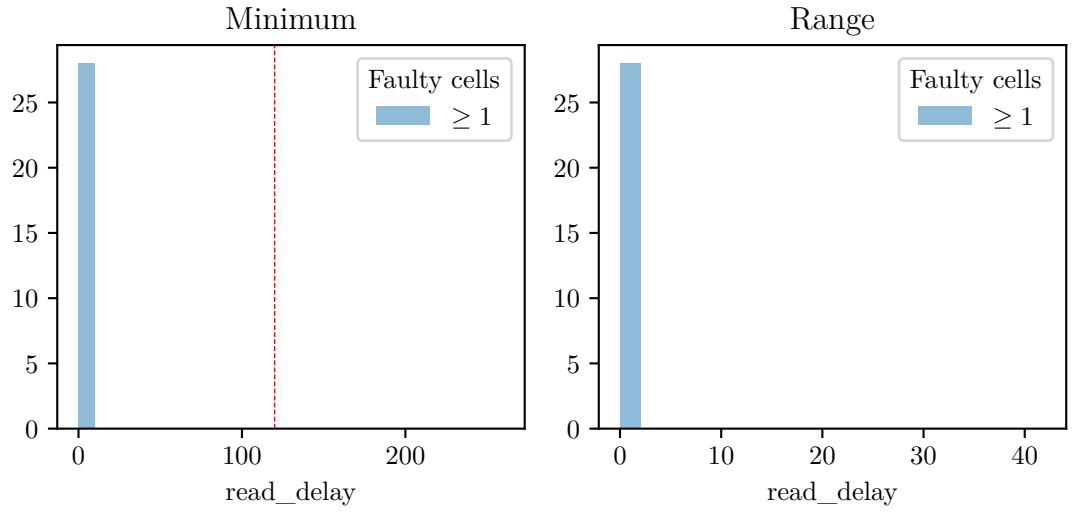


(b) `address_setup_time`

Figure 5.12.: Distribution over 56 analog neuron config SRAMs of the highest faulty `read_delay` along `write_width` and `address_setup_time`. Per SRAM the minimum and range of the highest faulty `read_delay` settings are considered. The red line indicates the standard `read_delay` before this thesis.



(a) `read_delay`



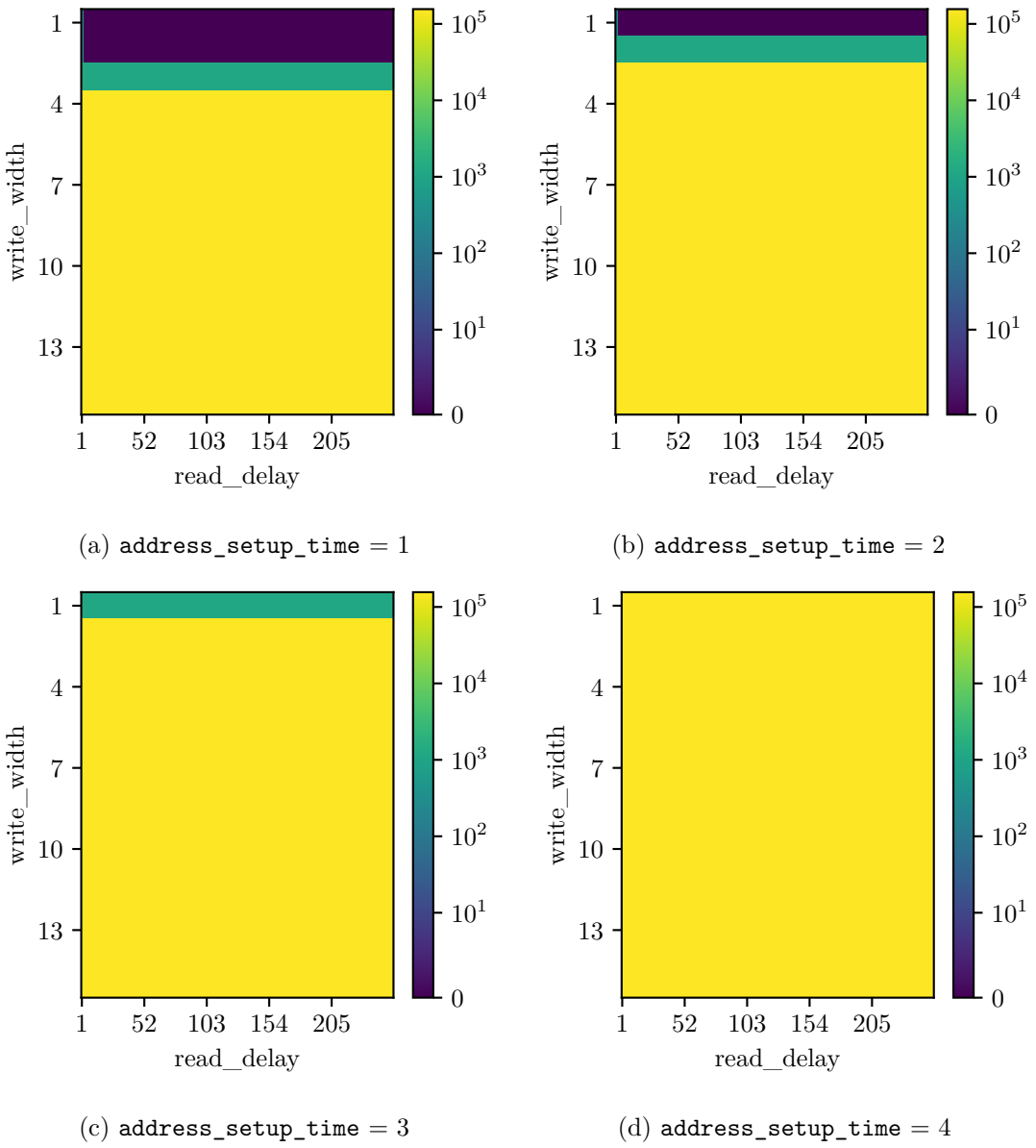
(b) `address_setup_time`

Figure 5.13.: Distribution over 28 digital neuron SRAMs of the highest faulty `read_delay` along `write_width` and `address_setup_time`. Per SRAM the minimum and range of the highest faulty `read_delay` settings are considered. The red line indicates the standard `read_delay` before this thesis.

5.1.5. CapMem

For the CapMem first results indicated that the configuration for CapMem SRAM behaves different from the other, as the sweeps showed no systematic structure in the parameter space. Further investigations showed, that a bug in the digital control logic of the CapMem SRAM caused the configuration to be gray coded before being written to the SRAM controller[1]. To account for this we had to apply reverse gray coding in software, which this thesis also implemented into the hardware container of the CapMem timing configuration, to fix the issue permanently. In this particular case the *PseudoRandom* pattern was used, because the *ValueSweep* pattern did not find all faults.
In Figure 5.14 sweeps with the applied reverse gray coding are shown for the lowest 4 `address_setup_time` configurations. For low `address_setup_time` and `read_delay` the SRAM shows a behavior similar to the neuron configurations, with faults only at very low `read_delay` timings, but with increasing `write_width` or `address_setup_time` the SRAM starts returning faults for all `read_delay` settings. In particular the combined value of `write_width` and `address_setup_time` must be smaller than 4 to show the behavior of the neuron configurations and for fault free configurations to exist. Again the sweeps are identical for all setups. Furthermore, if just the *ValueSweep* pattern was employed only one CapMem field showed this behavior, while using the *PseudoRandom* pattern almost all fields returned faults, indicating that there are different sequences of operations that can cause the SRAM to return faults, depending on the field.
This behavior seems to be caused by a bug in the digital control logic of the CapMem SRAM that causes the SRAM to return other values than the last written[2]. For example for one particular field this caused the SRAM to return a previously written "old" value instead of the most recent, if a read on any SRAM cell was done after writing both, see Listing 1. Other fields seem to show a similar behavior for more complex sequences. This bug was also reproduced in a digital simulation of the SRAM controller, hence it's certainly a bug in the digital control logic.

[1] see Issue 4047
[2] see Issue 4049



(a) `address_setup_time` = 1

(b) `address_setup_time` = 2

(c) `address_setup_time` = 3

(d) `address_setup_time` = 4

Figure 5.14.: Number of faults found per timing configuration for the CapMem SRAM with the 4 lowest `address_setup_time` on setup W61F0 SRAM block 0. The sweep looks identical for all 56 tested SRAMs. The colorbar is adjusted to the maximum number of faults that can occur for the test.

```
cell0 = CapMemCellOnDLS(5, 10) # arbitary
cell1 = CapMemCellOnDLS(1, 23)

builder = PlaybackProgramBuilder()

builder.write(cell1, x)

builder.write(cell1, y)

builder.read(cell0)
ticket = builder.read(cell1)

run(builder)

print(ticket.get()) # reads x instead of y
```

Listing 1: Pseudo code that produces one example of the bug in the digital control logic of the CapMem SRAM.

5.1.6. Summary

Summarizing the results of the sweeps we can conclude:

— `read_delay` has the greatest impact on most SRAMs on BSS-2

— In comparison `write_width` and `address_setup_time` have small impact, with that of the former being slightly larger

— CADC and Synapse Driver SRAM have a large area of too "slow" timings and thus need to be set sufficiently, which depends on the SRAM

— CADC and Synapse Driver SRAM sometimes have singular cells that are significantly slower than the rest of the SRAM, which need to be considered for the correct configuration

— the CADC shows an anomaly around `read_delay` 3 at which the amount of faults is significantly reduced. In contrast, the synapse driver SRAM show the expected behavior with increasing `read_delay`

— both digital and analog neuron configuration work for almost all settings and are much less critical with regard to the timing configuration

— The CapMem behaves similarly to the neuron configurations for low timing configurations but stops working entirely if `address_setup_time` and/or `write_width` become too large

Due to these observations it was decided to default to the most conservative timings in the software for all SRAM except for the CapMem. For the latter we decided to employ the hardware standard configuration `read_delay` = 8, `write_width` = 1 and `address_setup_time` = 1.

5.2. SRAM Hyperparameters

In this section the tests carried out for determining the impact of the hyperparameters supply voltage and temperature on the fault and timing behavior of the SRAMs are described.

5.2.1. Supply Voltage

First results showed that the behavior of the two neuron configuration SRAMs and the CapMem SRAM was not altered by the supply voltage. This could be explained by their already very fast timing behavior, such that we don't see the speed impact of the voltage changes in the considered range. Furthermore, the faulty settings of the CapMem are due to a bug in the digital logic which is presumably not influenced by the supply voltage. For this reason, only the CADC and synapse driver SRAMs are considered in the following.

CADC

In Figure 5.15, we can see a selection of the sweeps for different voltages, including the lowest and highest tested voltage. The obvious trend is that with rising voltage the SRAM becomes faster along `read_delay` which is shown by the left moving edge of the faulty area. Furthermore, a slow singular cell shows an alternating pattern along `write_width` with the "amplitude" decreasing with increasing voltage. Both of these observations support the fact, that the voltage acts as a scaling factor to the speed of the SRAM, which influences the timing mainly via the current I in Equation (2.1), but we do not go low enough with the voltages to see a complete failure of the SRAM. To better visualize the evolution of the speed of the SRAM the minimum of the highest faulty `read_delay` along `write_width` per voltage is considered, this again is the minimum `read_delay` that can be chosen for a stable configuration.
In Figure 5.16, we see the evolution of the minimum of highest faulty `read_delay` with the supply voltage for one setup. The data is fitted with an inverse square relationship, which is expected from the equation for the charge time Equation (2.1) and current Equation (2.2). We can see that depending on the SRAM we need a critical minimal voltage for fault free configurations to exist, after which the SRAM becomes faster with increasing voltage, with an approximate inverse square law, which deviates from the data at the higher voltages for one of the SRAM. The difference in the behavior of the hemispheres is due to a slow singular cell.
To understand the impact of the supply voltage on the functionality across the systems, the minimum supply voltage needed, such that the highest faulty `read_delay` is below

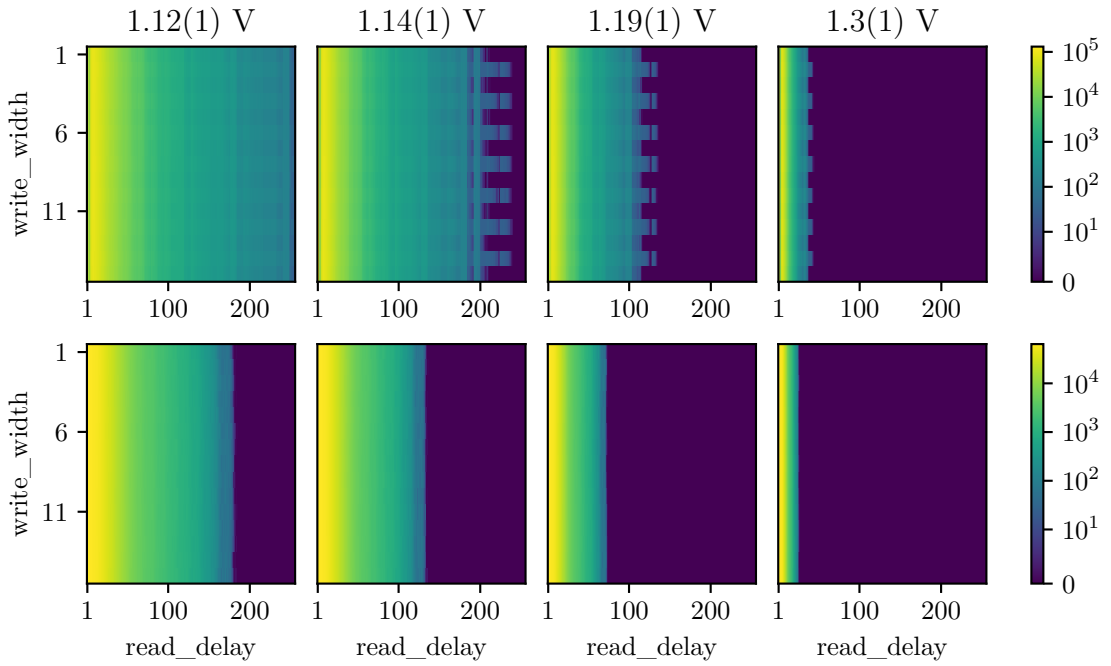


Figure 5.15.: Selection of CADC (top) and synapse driver (bottom) SRAM `read_delay` and `write_width` sweeps at maximal `address_setup_time` at different supply voltages. The colorbar is adjusted to the maximum number of faults that can occur for the test.

the maximum value, is considered. The distribution of these voltages is shown in Figure 5.17. All but one system have fault free configurations at the current supply voltage and most of them expect[3] 3 even under the nominal voltage of 1.2 V. Furthermore, looking at the statistics excluding singular faulty cells, we see that without them all systems have functional timing configurations below the nominal value. Hence, slow singular cells need also to be considered when setting the supply voltage, as they may scale stronger with the voltage than the majority of the cells.
In Figure 5.18 the faults per `read_delay` summed over all `write_width` and maximum `address_setup_time` are shown all for the measured voltages. It can be seen that for large `read_delay` the decay of faults becomes faster with increasing voltage, which is in line with the previous observations. Interestingly, we also see a transition of the `read_delay` anomaly from 3 to 2 with rising voltage, where the amount of faults at 3 increases while the amount of faults at 2 decreases and at `read_delay` 4 we see an inversion of the voltage behavior where the amount of faults increases with increasing voltage.

[3]see Issue 4048

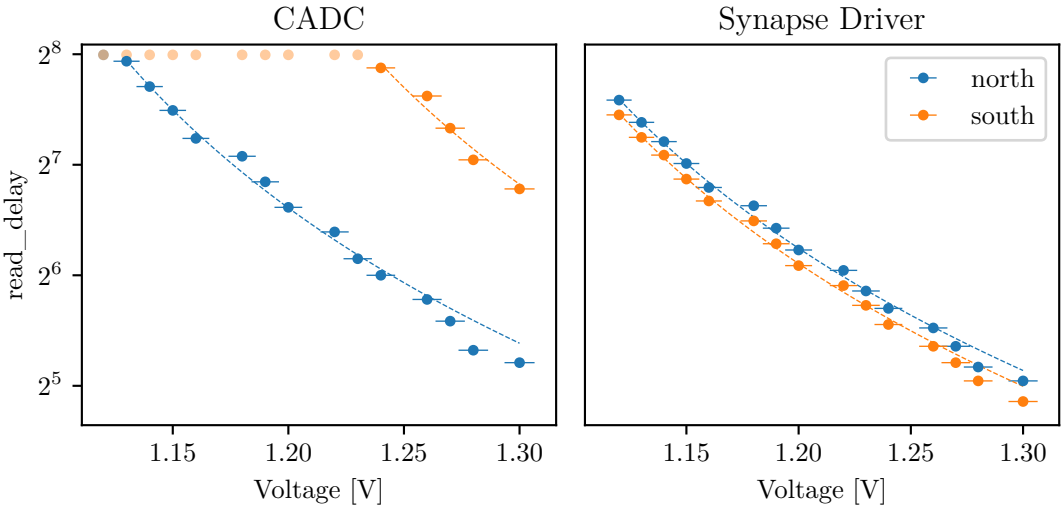
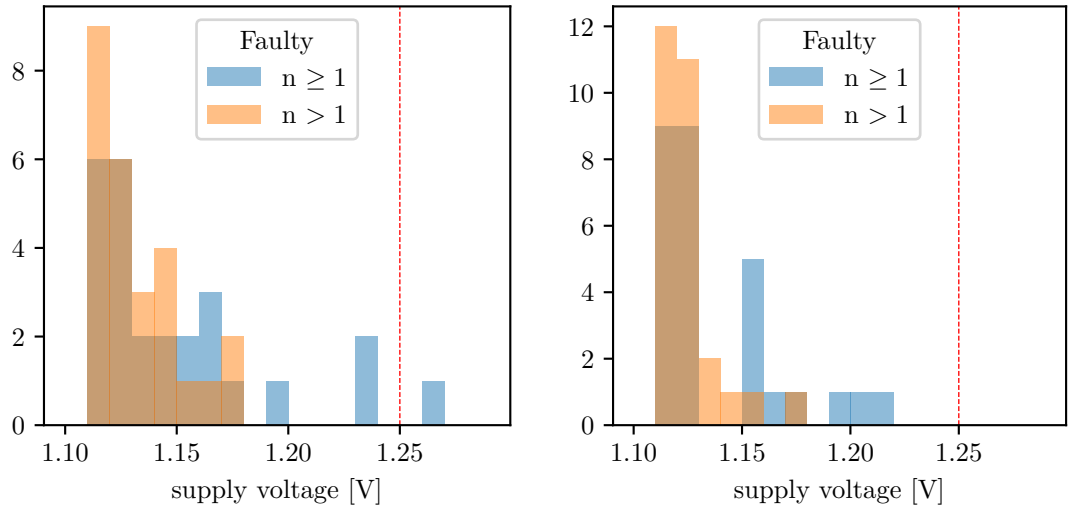


Figure 5.16.: Evolution of the minimum of the highest faulty `read_delay` along `read_delay` with supply voltage for the CADC and synapse driver SRAMs of setup W61F0 with fits of an inverse square relationship given by Equation (2.1) and Equation (2.2). Voltages that have no working configuration are greyed out. The error bars show the 0.5 % accuracy of the INA219. There can also be up to 5 mV equal offset of all voltages (Texas Instruments, 2015).

Synapse Driver

Figure 5.19 shows a selection of sweeps for different voltages for one synapse driver SRAM. Similar to the CADC the SRAM becomes quicker with increasing voltage and a decrease of variation along `write_width` can be observed, thus again we see a scaling effect of the voltage. Again we consider the minimal supply voltage per SRAM needed for a fault free configuration to exist, to investigate how the supply voltage impacts the reliability among the SRAMs. The distribution of the minimum supply voltage in Figure 5.17b shows that for the synapse driver SRAM all setups have a working state with the current supply voltage of 1.25 V and there are only 2 SRAMs that would not work with the nominal voltage of 1.2 V. Both of the setups that do not have a fault free configuration at nominal voltage have a singular slow cell, which is the reason for the higher supply voltage needed. Among the setups the slow singular cells are less of a problem for the synapse driver SRAM than for the CADC, but should also be considered when one wants to lower the supply voltage.
In Figure 5.16 we see the evolution of the minimum of the highest faulty `read_delay` along `write_width` with the supply voltage for one setup. The data is fitted with an inverse square relationship as before, which deviates from the data for the highest voltages. Furthermore, we see a very similar behavior for both hemispheres, which is



(a) CADC Offset

(b) synapse driver configuration

Figure 5.17.: Distribution of the minimum supply voltage for CADC and synapse driver SRAM. The red line indicates the standard supply voltage at the point of this thesis.

expected in absence of slow cells, and validates the the highest faulty `read_delay` as a metric for the speed of the SRAM. In conclusion, we see an approximate inverse square law for the speed of the SRAM with the supply voltage, but our model underestimates the speed of the SRAM at the highest voltages for both SRAMs. This could either be due to the model not being accurate for high voltages, or we underestimated the trend in general, which for example could be due to short channel effects such as channel length modulation (Dimitrijev, 2012).
Figure 5.19 shows the faults per `read_delay` over all `write_width` and maximum `address_setup_time` for the measured voltages. The behavior here as before is exactly what we would expect, as we see an increase in speed with increasing voltage and again no anomaly at low `read_delay`.

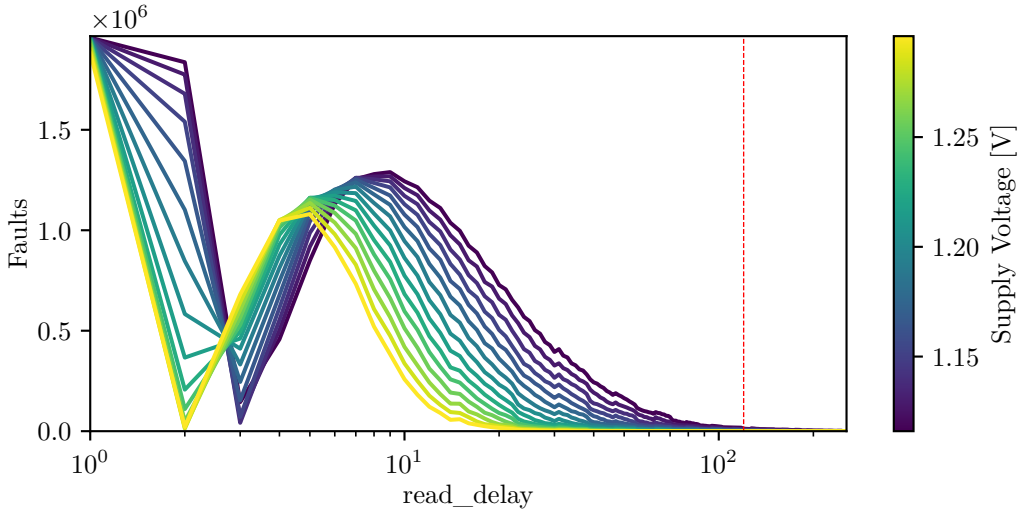


Figure 5.18.: Faults per `read_delay` for one CADC SRAM summed over all `write_width` and at maximal `address_setup_time` for the measured voltages. We see a transition of the `read_delay` anomaly from 3 to 2 with rising voltage. The red line indicates the standard `read_delay` before this thesis.

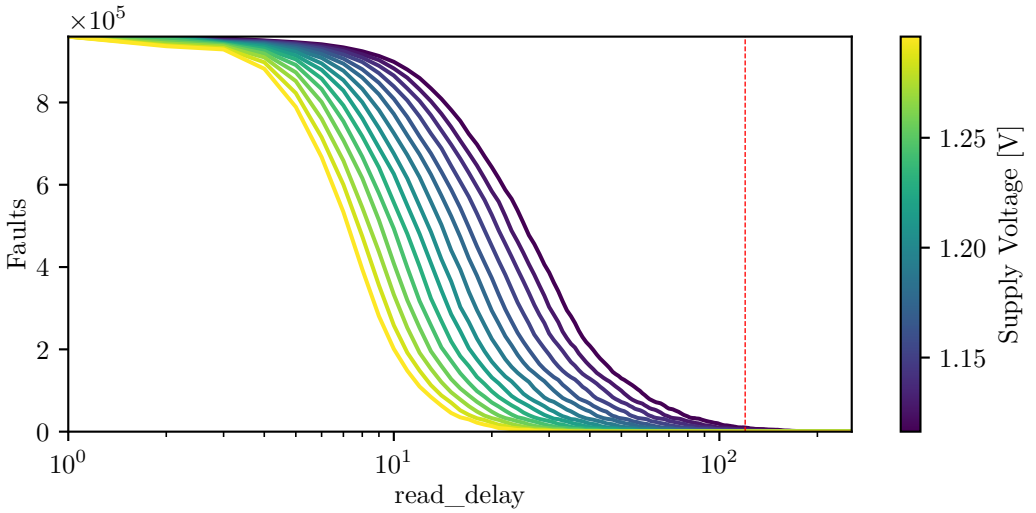


Figure 5.19.: Faults on synapse driver SRAM per `read_delay` summed over all `write_width` and at `address_setup_time` 15 for the measured voltages. The red line indicates the standard `read_delay` before this thesis.

5.2.2. Temperature

Last we investigated of the impact of the temperature on the fault and timing behavior of the SRAMs. Here only sweeps for the two CADC SRAMs on one chip were done, of which we see a selection for different temperatures in Figure 5.21. As one can see in Figure 5.20, the SRAM does in fact speed up with increasing temperature and similarly to the supply voltage reduces the variation along `write_width`. For once, we would expect the mobility and SNM to decrease with increasing temperature, which would necessitate slower timings. Furthermore, the saturation velocity also decreases with increasing temperature, which could also slow down the SRAM if it is a limiting factor for the current, which is the case for high voltages. On the contrary the threshold voltage of the transistors decreases with increasing temperature, which would increase the read current, allowing for faster timings (Wolpert and Ampadu, 2011). As we measure the fastest timing for the slowest cells, it may also be that leakage currents play a role, as they increase with temperature, which may allow for slow cells to work with shorter timings. Lastly, it could also be the case, that the SRAM does in fact slow down, but the SRAM controller/digital logic slows down more, such that the SRAM appears faster. Which effect dominates is not clear, but the results show that the speed of the SRAM increases with increasing temperature. The main conclusion is here, that setting the timing configuration of the SRAM above the highest faulty `read_delay` from the previous analysis should be sufficient to ensure the functionality of the SRAM under increasing temperature in the tested operating ranges. By the discussion above, we also have to pay attention to the temperature if the supply voltage was to be increased, as this could lead to an inversion of the observed behavior, because it weakens the impact of the threshold voltage and may increase the impact of velocity saturation (Wolpert and Ampadu, 2011).

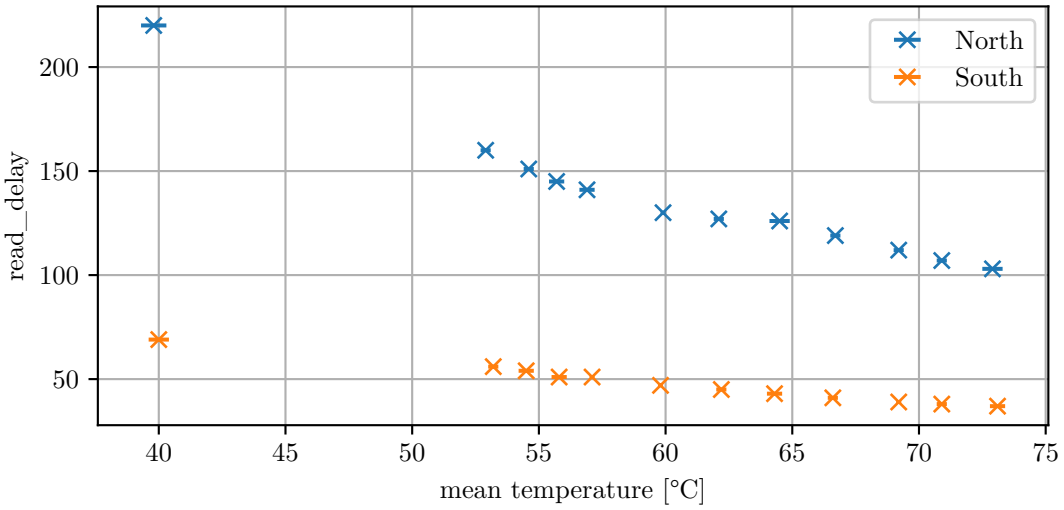


Figure 5.20.: Evolution of the minimum of the highest faulty `read_delay` along `write_width` against average temperature during the sweep for the CADC SRAM at maximal `address_setup_time`. The error bars show the observed range of the temperature during the sweep.

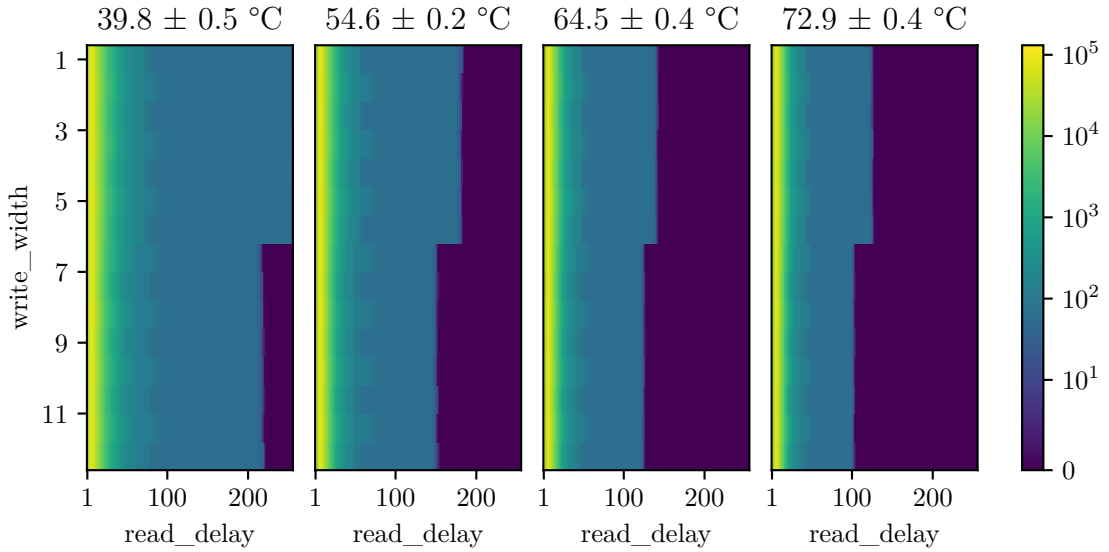


Figure 5.21.: Selection of the sweeps on the CADC SRAM for different voltages. The average temperature during the sweep T is given with 0.5 °C accuracy of the TMP112 sensor, see Texas Instruments (2024). The given deviations indicate the total range of the temperature during the sweep.

5.2.3. Summary

We have seen that both the supply voltage and temperature have an impact on the timing behavior of the SRAMs of the BSS-2 platform, both increasing the speed of the SRAM when they are increased. Considering the distributions of the minimum supply voltage (which is trivial for the not considered SRAM in the tested region), we can conclude that the current supply voltage is sufficient for almost all SRAMs, but should not be lowered to ensure the functionality of the CADC and synapse driver SRAMs. We have also seen that anomaly of the CADC SRAM at low `read_delay` also moves towards lower `read_delay` with increasing supply voltage, gradually increasing the faults at `read_delay` 3 and decreasing the faults at `read_delay` 2 with increasing supply voltage. In conclusion, setting the timing configuration of the SRAM above the highest faulty `read_delay` for all `write_width` should be sufficient to ensure the functionality of the SRAM with increasing temperature conditions. If supply voltage or temperature would be lowered, one has to pay particular attention to the slow singular cells, which may fail earlier than anticipated. Also increasing the supply voltage may lead to an inversion of the observed temperature behavior, which could cause faults if the temperature rises.

Chapter 6.

Discussion and Outlook

6.1. Summary and Discussion

The goal of this thesis was to develop a testing framework for the SRAM of the BSS-2 platform and use it to investigate the reliability and behavior of the SRAM with respect to the timing configuration and under varying supply voltage and temperature.
I have developed a testing library that allows treating the SRAM in a unified way and apply tests and patterns universally to all memories. This was achieved by abstracting the hardware specific details of the SRAMs into a common interface and providing a set of patterns and methods that can be applied to all memories using that interface. Furthermore, I also abstracted the creation of timing configuration sweeps employing the implemented tests. This allows for a quick implementation of tests and sweeps for new memories and made it able to quickly cover all SRAMs of the BSS-2 platform. Furthermore, the unified code paths reduce duplicate code and avoid errors due to differences in the implementation. We employed parallelization of the sweeps and partial implementation in CPP to speed up the testing process, which showed to be very effective. The timing configuration sweeps also offers the possibility to add further instructions using hooks to the sweep, which can be used to test the behavior of the SRAMs under different conditions, such as the supply voltage or temperature.
The testing framework was then used to investigate the reliability and behavior of the SRAMs with respect to the timing configuration. This was done by applying sweeps of the timing configuration and analyzing the occurrence of faults for different timings. The results showed that the CADC and synapse driver SRAMs are sensitive to the timing configuration and can fail if the `read_delay` configuration is set too low, the remaining timing configurations, `write_width` and `address_setup_time`, showed only a small impact on the behavior of the memory. Furthermore, the CADC and synapse driver SRAMs showed singular cells that are significantly slower and need higher `read_delay` timings to work, and showed more dependence on the `write_width` and `address_setup_time` for the CADC. Increasing supply voltage showed an increase of read speed and a decrease of variation along `write_width` and `address_setup_time`. One instance of a setup where under the current operating voltage no stable configuration was found, could be operated fault free at an increased supply voltage. The sweeps of the CADC SRAM under different temperatures also showed an increase of read speed with increasing temperature. Multiple effects that play a role in the speed of the SRAM with increasing temperature were discussed and it was noted that increasing supply voltage may lead to

an inversion of the observed temperature behavior. The CADC exhibited an anomaly at `read_delay` 3 where the amount of faults was significantly reduced in comparison to the overall trend. With increasing supply voltage the anomaly gradually transitioned to `read_delay` 2. The synapse driver memory did not show such an anomaly, which makes it unlikely that the anomaly is due to a bug in the SRAM controller. The analog and digital neuron configuration SRAMs showed to be much less sensitive to the timing configuration and work for all `read_delay` timings greater 1 or 3 respectively and are not sensitive to `write_width` nor `address_setup_time`. The CapMem SRAM showed a different behavior, it was found that the configuration was gray coded and had to be reverse gray coded to apply the correct configuration. Furthermore, the memory showed a bug in the digital control logic that caused the SRAM to return wrong values, if `write_width` and `address_setup_time` were set too high. Otherwise, it was found to be functional for any `read_delay` timing greater 1 if the other timings were set correctly. For these remaining SRAMs the read speed was not significantly influenced by the supply voltage. From these observations we concluded, by choosing long enough timings for the SRAMs, we can ensure the functionality of the memory and ensure that the SRAMs are reliable under varying supply voltage and temperature conditions in the tested ranges.

SRAM	**faulty setups before**	**faulty setups after**
CADC	4	1[1]
synapse driver	4	0
SynRam	3	0
neuron configurations	0	0
CapMem	0	0

Table 6.1.: Number of faulty setups before the internship and after the thesis by applying the suggested configurations.

In summary, I can conclude with this thesis, that with the actions taken the SRAM of the BSS-2 are now well understood and reliable. The impact of the supply voltage and temperature have been investigated and have showed to not be critical for the functionality of the memories in the current state of the BSS-2 platform. The developed testing framework can be used in the future to commission new chips and investigate occurring faults, and may even be used to do further investigations on the behavior of the SRAM under different conditions.

[1] the faulty setup would be fixed by increasing the supply voltage from 1.25 V to 1.27 V

6.2. Outlook

In conclusion, the tests and characterization of the memory of the BSS-2 platform have covered SRAMs and provided a good understanding of the behavior understand different conditions and basis for reliable operation of the memory on platform. However, there are possibilities to extend the application of the testing framework and open issues regarding the higher level software that depends on the timing configuration of the SRAM.

— The sweeps of the timing configurations could be included in the continuous integration pipeline to detect long-term changes in the behavior of the SRAM such as degradation of the SRAM cells. Using this faulty setups could be detected before they cause problems in the operation of the BSS-2 platform.

— One could, to better understand the impact of temperature, set up a temperature control system using a peltier element on the back of the chip carrier. With this variation of the temperatures below the room temperature could be investigated and also automatically collect data over all setups, to get statistical data on the impact of the temperature on the SRAM.

— Future hardware iterations could feature build-in self-test (BIST) for the SRAM to detect faults automatically and even could implement an automatic tuning of the timing configuration. This would make the platform more self-sufficient and reduce the need for manual testing. Furthermore, this could become more important if the hardware would mover to a smaller process node, where the SRAM is more prone to parametric faults (Sharma and Ravi, 2016).

— There have been issues regarding user-level software if the timings are set to the maximum value. In particular some calibration routines have been reported to fail, for at this moment unknown reasons. One likely cause is that the calibration routines assume some minimal speed of the SRAM, hence one next step would be to investigate this further and possibly adapt the calibration routines to be timing configuration agnostic.

Bibliography

Arandilla, Christiensen D.C., Anastacia B. Alvarez, and Christian Raymund K. Roque (2011). "Static Noise Margin of 6T SRAM Cell in 90-nm CMOS". In: *2011 UkSim 13th International Conference on Computer Modelling and Simulation*, pp. 534–539. DOI: 10.1109/UKSIM.2011.108.

Bergler, Nils (2024). *SRAM tests of synapse weight memory on neuromorphic hardware*. https://www.kip.uni-heidelberg.de/vision/publications/reports/report_nbergler.pdf.

Bosio, A. et al. (2012). "Advanced test methods for SRAMs". In: *2012 IEEE 30th VLSI Test Symposium (VTS)*, pp. 300–301. DOI: 10.1109/VTS.2012.6231070.

Cook, Alejandro et al. (2012). "Built-in self-diagnosis targeting arbitrary defects with partial pseudo-exhaustive test". In: *2012 13th Latin American Test Workshop (LATW)*, pp. 1–4. DOI: 10.1109/LATW.2012.6261229.

David, R., A. Fuentes, and B. Courtois (1989). "Random pattern testing versus deterministic testing of RAMs". In: *IEEE Transactions on Computers* 38.5, pp. 637–650. DOI: 10.1109/12.24267.

Dimitrijev, S. (2012). *Principles of Semiconductor Devices*. Oxford series in electrical and computer engineering. Oxford University Press. ISBN: 9780195388039.

Dounavi, Helen-Maria and Yiorgos Tsiatouhas (2023). "An aging monitoring scheme for SRAM decoders". In: *Integration* 88, pp. 108–115. ISSN: 0167-9260. DOI: https://doi.org/10.1016/j.vlsi.2022.09.009.

Electronic Visions (2024a). *Coordinate Systems for BrainScaleS architectures*. URL: https://github.com/electronicvisions/halco.

Electronic Visions (2024b). *Hardware Abstraction Layer (and STAteful encapsulation) for the HICANN-DLS Hardware*. URL: https://github.com/electronicvisions/haldls.

Harris, Charles R. et al. (Sept. 2020). "Array programming with NumPy". In: *Nature* 585.7825, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

Hunklinger, Siegfried and Christian Enss (2023). *Festkörperphysik*. Berlin, Boston: De Gruyter Oldenbourg. ISBN: 9783111027227. DOI: doi:10.1515/9783111027227. URL: https://doi.org/10.1515/9783111027227.

Müller, Eric et al. (2020). *Extending BrainScaleS OS for BrainScaleS-2*. arXiv: 2003.13750 [cs.NE]. URL: https://arxiv.org/abs/2003.13750.

Pavlov, Andrei and Manoj Sachdev (2008). *CMOS SRAM Circuit Design and Parametric Test in Nano-Scaled Technologies: Process-Aware SRAM Design and Test*. Springer Publishing Company, Incorporated. ISBN: 1402083629.

Pehle, C. et al. (Feb. 2022). "The BrainScaleS-2 Accelerated Neuromorphic System With Hybrid Plasticity". In: *Front. Neurosci.* 16.795876. DOI: 10.3389/fnins.2022.795876.

Rathi, Neetu et al. (2023). "A Review of Low-Power Static Random Access Memory (SRAM) Designs". In: *2023 IEEE Devices for Integrated Circuit (DevIC)*, pp. 455–459. DOI: 10.1109/DevIC57758.2023.10134887.

Sharma, Abhinav and V. Ravi (2016). "Built in self-test scheme for SRAM memories". In: *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1266–1270. DOI: 10.1109/ICACCI.2016.7732220.

Singh, Vijay, Sanjay Kumar Singh, and Raman Kapoor (2020). "Static Noise Margin Analysis of 6T SRAM". In: *2020 IEEE International Conference for Innovation in Technology (INOCON)*, pp. 1–4. DOI: 10.1109/INOCON50539.2020.9298431.

Srinu, M., E. Sreenivasa Rao, and P. Chandra Sekhar (2024). "Design of low power SRAM cells with increased read and write performance using Read - Write assist technique". In: *e-Prime - Advances in Electrical Engineering, Electronics and Energy* 7, p. 100381. ISSN: 2772-6711. DOI: https://doi.org/10.1016/j.prime.2023.100381.

Sze, S.M. and M.K. Lee (2012). *Semiconductor Devices: Physics and Technology*. Semiconductor Devices, Physics and Technology. Wiley. ISBN: 9780470537947.

Texas Instruments (2015). *INA219 Zerø-Drift, Bidirectional Current/Power Monitor With I2C Interface*. INA219. URL: https://www.ti.com/lit/ds/symlink/ina219.pdf.

Texas Instruments (2024). *TMP112x High-Accuracy, Low-Power, Digital Temperature Sensors With SMBus and Two-Wire Serial Interface in SOT563 and X2SON Package*. TMP112. URL: https://www.ti.com/lit/ds/symlink/tmp112.pdf.

Wolpert, David and Paul Ampadu (Jan. 2011). *Managing Temperature Effects in Nanoscale Adaptive Systems*. ISBN: 9781461407485. DOI: 10.1007/978-1-4614-0748-5.

Zu, Yazhou et al. (2016). "Ti-states: Processor power management in the temperature inversion region". In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–13. URL: https://ieeexplore.ieee.org/document/7783758.

Appendix A.

Appendix

A.1. Glossary

SRAM Static Random Access Memory

SynRam synaptic memory

SNN Spiking neural network

BSS-2 BrainScaleS-2

CADC column analog-to-digital converter

CapMem capacitive memory

SNM static noise margin

LDO low-dropout regulator

DAC digital-to-analog converter

jBOA just a bunch of ASICs

A.2. Software Versions

Name	Version/Git Hash
container	2024-04-17
bitfile	156, built on 2024-08-21
code-format	09f3a985a6f264359b10a6a129dd6dce7e55c9e8
halco	91d128d591d89ef25a45c07f3ec59277fc65a719
haldls	6c5881f7f27d980f611206779e1300f820485976
fisch	a67fc99215f038f09a33fd09ff85c0bb594f9f8c
rant	722edd57c9e42462a660db8a1febb0211ffad07c
ztl	b6745261d8bfdce44516d58d632c3c73834839d2
pywrap	5e2af30e9593882b471d3cd02df00b93f13ff479
lib-boost-patches	136c5b41cb046afe2c726aa4646928bf5190622e
libnux	fc3b137384596ea5adbd5d4ee1ddfc9761a2aabc
hate	35b3cb211cabbbc5c01036ae7878a73e338166c4
logger	73dadb3ce413c521845ef7d36f818073eee4fefa
hxcomm	95abf25670bd8cb7cc5b499cde56f653130cf20c
sctrltp	1d854f953f7e8c8ead44406a22bb80421ca3857c
visions-slurm	8f41ea4f5bd1573d8f4623e9ed698a29f30036a3
flange	28e729d59df3b4ff380f84351c40d4da3086bed8
lib-rcf	21fbcb0a7c30efed98278ee997754f28092b9736
bss-hw-params	b7be7827b51536804f0bda76f8ba4be693df23a8
hwdb	f7262189b0e55b686896a3dea952065c2f1a3789
bss2-devops	aaefdfdf11099ce21f651013de098ebc26b25056
calix	a706868c6ba285b1f8fd7cdef1a19d7328e02912

Table A.1.: Software versions used in this thesis

A.3. Funding Statement

The work carried out in this Bachelor Thesis used systems, which received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreements Nos. 720270, 785907 and 945539 (Human Brain Project, HBP) and Horizon Europe grant agreement No. 101147319 (EBRAINS 2.0)

A.4. jBOA heat chamber

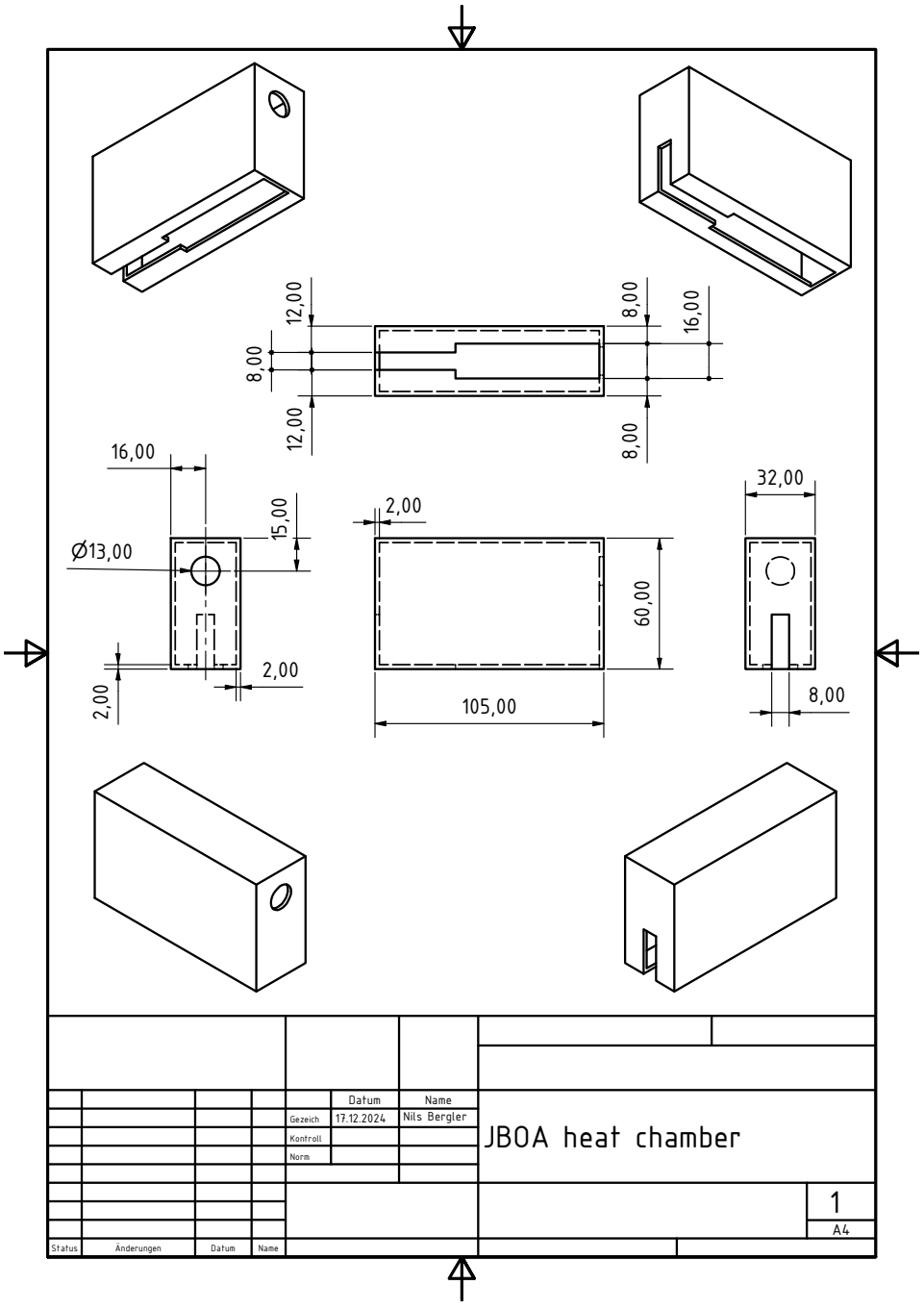


Figure A.1.: Schematic of the heat chamber for the jBOA system.

Acknowledgements

I would like to thank my supervisor Yannik for his guidance and support during the thesis and the internship, and the great feedback for me to improve my work. I thank Joscha for his expertise and help with the just a bunch of ASICs (jBOA) hardware. I also thank Arik and Florian for their excellent feedback on my bachelor thesis and keeping up with my weird sentences. Furthermore, I would also like to thank the rest of the Electronic Visions group for their support and the great working atmosphere, and for keeping everything running. Lastly, I would like to thank my girlfriend and family for her support and understanding during the thesis.
An honorable mention goes to the Electronic Visions espresso machine, which kept me awake during the long days, and is unfortunately broken at the time of writing.

To find fault is easy; to do better may be difficult.

(Plutarch)

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 29. 01. 2025,