In-Depth Analysis of DRAM Cells During Rowhammer

And a Novel Protection Method

Stefan Alexander Lung

Delft University of Technology

TU Delft

In-Depth Analysis of DRAM Cells During Rowhammer

And a Novel Protection Method

by

Stefan Alexander Lung

In partial fulfilment of the requirements for the degree of
Master of Science
In Embedded Systems

Student number:	4749731	
Thesis number:	Q&CE-CE-MS-2023-01	
Thesis duration:	September 12, 2022 – August 30, 2023	
Thesis committee:	Georgi Gaydadjiev,	TU Delft, supervisor
	Mottaqiallah Taouil,	TU Delft, supervisor
	Ranga Rao Venkatesha Prasad,	TU Delft, member

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Embedded Systems Programme
Computer Architecture Specialisation

Cover: Visualised DRAM data pattern used for testing the hardware implementation.

TU Delft

Abstract

Rowhammer is a security exploit used to cause bit errors DRAM chips. Newer DRAM technologies are becoming more vulnerable to rowhammer attacks, and existing protection methods are starting to reach their limits. This thesis provides methods for DRAM characterization by means of reverse engineering, an in-depth analysis of vulnerable cells during rowhammer and a novel protection method.

First the DRAM module is characterized by performing a retention test and a one-sided rowhammer attack throughout the memory. The resilience to rowhammer of vulnerable victim cells has been analysed by using various data patterns in the victim cell itself in combination with its direct and diagonal neighbours. The impact of the proposed protection method is measured by flipping the data in the attacker row during a simulated rowhammer attack. Various rowhammer sequences are investigated.

The results following the DRAM characterization show that only a one-sided rowhammer attack is possible. Rows that impact one another during rowhammer come in pairs of two, one attacker row impacts only one neighbouring row. The results of the in-depth analysis of cells during rowhammer shows evidence of negative horizontal impact coming from uncharged neighbouring cells in the same row and negative diagonal impact coming from charged cells neighbouring the attacker cell in the attacker row. The horizontal impact is mirrored in symmetrical cells whilst the diagonal impact is not. Results following the proposed protection method experiments show that using the protection method improves the overall resilience to rowhammer ranging from 50%, 65% to 100% depending on the hammer sequence.

Acknowledgements

The completion of this thesis would not have been possible without the support of others.

First, I would like to thank my thesis supervisors, Mottaqiallah Taouil and Georgi Gaydadjiev, both working at the Delft University of Technology. I am grateful for their academic insight, tips, guidance and constructive criticism.

Next I would like to thank my friends, whom helped me throughout the project by providing the occasional distractions.

And last but not least, I would like to thank my family and girlfriend for their understanding and emotional support.

Stefan Lung
Hoofddorp, August 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the topic of the thesis. A motivation is given, state of the art protection methods are discussed, the thesis contributions are provided and the thesis layout is presented.

1.1 Motivation

DRAM has become an essential component in modern computer systems. Ever since the first development of DRAM the strive has been to offer the processors more memory at a higher access rate. According to Moore's law every two years the number of transistors in IC's doubles. DRAM technology has also shown to follow this trend. Where every generation of DRAM development the bit density increases meaning a higher storage capacity. Increasing DRAM capacity has come at a cost and newer technology is being pushed to its limits. Modern DRAM devices have shown to suffer from the unwanted flipping of bits caused by specific access patterns.

One of the first papers discussing the topic of flipping bits in DRAM memory is [1]. Here it is mentioned that the scaling down of DRAM technology has made it more difficult to prevent unwanted electrical interaction between cells. Where DRAM cell implementations have become smaller and smaller, negative effects have started to become more prevalent.

This unwanted flipping of bits in DRAM memory has not gone unnoticed among those with certain curiosities or even malicious intent. This is where the term rowhammer comes to play. rowhammer is a security exploit where an attacker frequently accesses a row in DRAM with the intention to flip bits in neighbouring rows. Other than soft errors that are caused by environmental interference, bit flips caused by rowhammer are due to electrical interference between components and these are intentional. In 2015

Google's Project Zero [2] showed that is is possible to gain kernel privileges by means of a rowhammer attack. It was shown that a successful bit flip could occur in laptops dating as far back as 2010.

The ease at which it takes to flip bits in DRAM devices is increasing for newer generations of DRAM. The results in [3] show that newer DRAM devices are becoming increasingly vulnerable to rowhammer and require significantly fewer row accesses to cause bit flips in neighbouring rows.

Protection methods have been developed to mitigate the effects of rowhammer. These methods range from manufacturing DRAM with higher quality standards, adjusting DRAM refresh period parameters, refreshing only victim rows and even the detection of malicious access patterns. Many of these protection methods are being pushed to their limits as they have to account for the decrease in resilience of modern DRAM devices.

The motivation behind this research is due to two main point. These are, that newer DRAM technology is becoming increasingly vulnerable to rowhammer and existing protection methods cannot keep up with this trend. The goal is to investigate the effects of a newly proposed protection method, that operates by flipping the attacking row data throughout the rowhammer attack in order to prevent the failure of bits in the victim row. This investigation will also entail the effects that local DRAM cells have on each other during rowhammer without and with implementing the proposed protection method.

1.2 State of the art

In the present day there are numerous protection methods specifically targeting rowhammer. A number of these protection methods have been investigated and are discussed in [3]. Figure 1.1 shows the different types of protection methods. A clear distinction is made between hardware and software implementable protection methods. This thesis will solely focus on a hardware implementable protection method. The software half in the figure is to indicate that such protection methods exist.

The probabilistic protection methods are based on the concept of refreshing the victim row with probability. After a row has been accessed the PARA [9] refreshes the neighbouring rows with a certain probability. PRoHIT [10] uses a table to keep track of rows

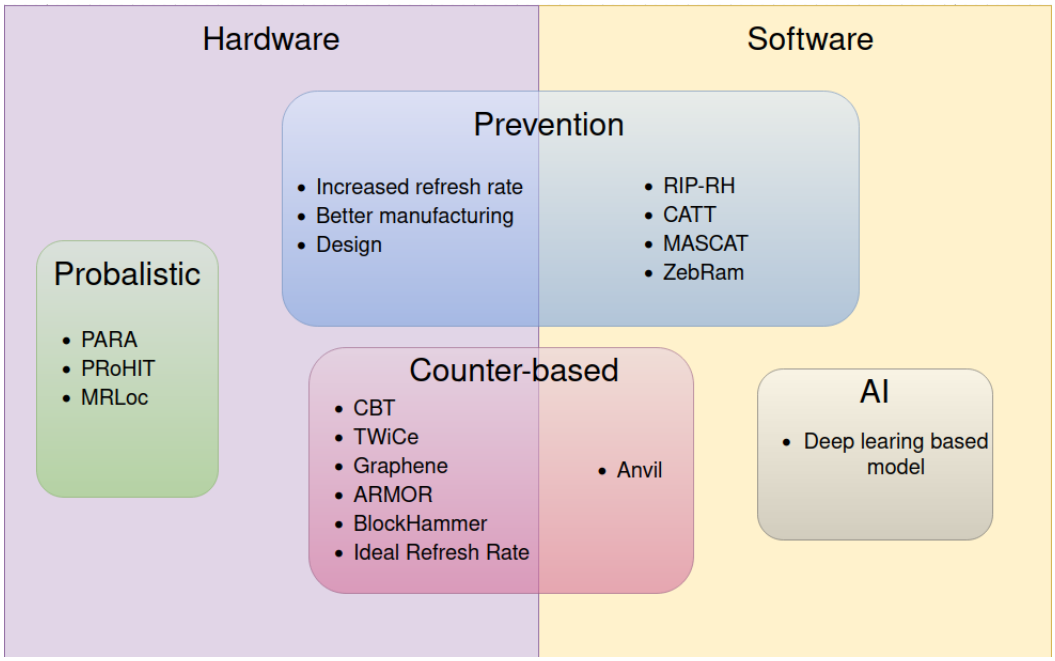


Figure 1.1: Protection method types

being accessed frequently, and refreshes the neighbouring rows with certain probability. MRLoc [11] also keeps track of rows that are being accessed at a high rate and places the neighbouring victim rows in a queue to be refreshed with certain probability. Row refreshing restores the cell charge levels to their original state and if performed in time will prevent bit errors. New DRAM technology is becoming increasingly vulnerable to rowhammer causing the probabilistic methods to increase their probability to refresh neighbouring victim rows. This in turn means that the DRAM will be refreshing its rows more frequently, resulting in a higher energy consumption and overhead.

Counter based methods are designed to keep track of attacker rows and to take active measures. Ideal-refresh-rate [6], CBT [12], TWiCe [13], Graphene [14], ARMOR [15] and BlockHammer [16], use different data structures and schemes to keep track of the attacker rows they all refresh the victim rows after reaching a certain threshold. Meaning if an attacker row is accessed with a high frequency it will stand out and the neighbouring rows will be refreshed.

The prevention based protection methods do not keep track of possible attacker rows and no probabilistic refreshes are performed. Rather, it comes down to design and manu-

facturing of the DRAM chip and/or the controller. Increased refresh rate [6] determines the refresh rate depending on the lowest hammercount. Meaning before an attacker row can even reach the number of activation-precharge cycles needed to flip a bit in the victim row the whole memory bank is refreshed. Other preventive measures are to manufacture the DRAM chip with other materials or with higher quality standards. In [18] a proposal is made to perform hydrogen annealing as an extra step in the production process. Here it is suggested that the cell resilience to rowhammering increases due to this step. Reducing the wordline voltage as a protection method is discusses in [19]. This might decrease the effects of capacitive cross talk and electron drift. Though decreasing the wordline voltage will require a longer activation time and in turn lowers the possible rowhammer rate.

1.3 Contributions

In this thesis several topics will be investigated regarding rowhammer. A protection method is proposed and tested under certain circumstances. Before the protection method is investigated a set of preceding experiments is executed in order to understand the DRAM characteristics and the cell level impact of rowhammer. The main contributions of this thesis are:

- A protection method based on the flipping of cell charge in the attacker rows during rowhammer.
- Methods for characterizing specific DRAM devices by means of reverse engineering.
- Understanding the effect of neighbouring cell charge on victim cell resilience to rowhammer.
- A performance indication of the proposed protection method.

1.4 Thesis layout

This thesis is structured out as follows. Chapter 2 introduces the basic concept of DRAM and Rowhammer. Following this chapter 3 will propose the Row Flipper protection method and will discuss its expected results against rowhammer. Chapter 4 describes the methodology and design of the experimental setup. A few steps are proposed in order to provide a pathway. Following this the experiment list is given and a design of the experimental setup is discussed. Chapter 5 explains the experimental infrastructure

implementation and the results of the experiments are shown. Nearing the end, Chapter 6 discusses the results and compares them to the findings in literature, some new findings are discussed and the impact of the Row Flipper protection method is compared to state of the art. Finally, chapter 6 starts with a summary of the thesis, followed by the final conclusion and ending the thesis is a list of suggestions for future work.

Chapter 2

Background

This chapter discusses the basic concepts of DRAM and rowhammer. First DRAM is explained and how it manages to store and retrieve data. Following this is the concept of rowhammer and how it manages to flip bits in DRAM memory.

2.1 DRAM

At the hart of this thesis lies DRAM, which stands for Dynamic random-access-memory. DRAM is often implemented between the processor and a larger storage medium such as a hard drive. The reason why DRAM is an abbreviation with the word dynamic is because this memory has to be refreshed within a certain time period to ensure data integrity. Other than SRAM which stands for static random-access-memory which does not require the refreshing of data after startup. On the other hand SRAM is faster and uses less power, however it is more expensive compared to DRAM which is generally cheaper due to a higher bit density per area.

2.1.1 Cells

DRAM is composed of memory cells. Each cell holds a single bit of information by means of capacitance. Capacitors lose charge over time due to leakage and that is why DRAM needs to be refreshed periodically. A cell in DRAM represented as having one capacitor and one transistor. As described in [8] aside from this 1-transistor 1-capacitor there also exist other designs such as the 3-transistor 1-capacitor design. Modern DRAM use this 1-transistor 1-capacitor concept. Figure 2.1 below shows the 1-transisor 1-capacitor design. Here the *wordline* can open the *transistor* and charge can flow from the *capacitor* to the *bitline* or vica versa. Other sources use the word *digitline* in place of *bitline* and

rowline instead of *wordline*.

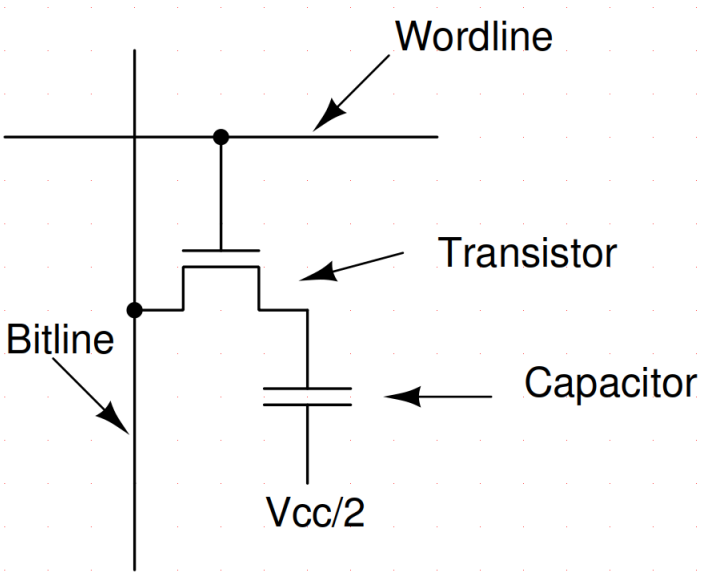


Figure 2.1: A DRAM 1-transistor 1-capacitor cell.

The bitline itself has a capacitive nature. Before accessing the capacitor by opening the transistor the bitline is charged to a voltage of $V_{cc}/2$. If this capacitor has a positive potential compared to the bitline it is considered to store a bit of value 1 and in the case it has a negative potential it is considered to store the value 0. As mentioned earlier when opening the transistor charge will flow from the capacitor to the bitline or vice versa. This will result in a change of voltage on the bitlines. Based on figure 1.26 in [8] Figure 2.2 shows the voltages of the bitline and cell capacitor before opening the transistor. Here the bitline is resembled as a capacitor due to its capacitive nature.

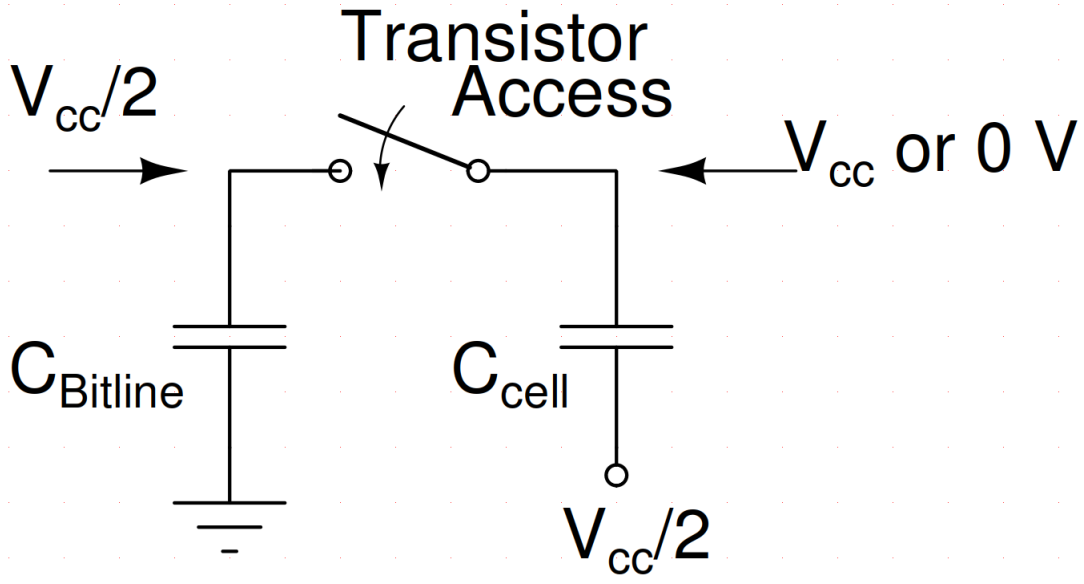


Figure 2.2: Schematic illustrating charge sharing between cell and bitline.

Sense amplifiers are used to measure the voltage difference on the bitlines. Two bitlines are shared by one sense amplifier and the sense amplifier measures the difference in voltage between the pair of bitlines. Before accessing any memory cells the bitlines both are charged to $V_{cc}/2$, and in this case the sense amplifier will measure a difference of 0 V between the bitline pair. Once a cell is accessed (wordline activated, bitline drained or charged) only one bitline in the pair will have a changing voltage. The voltage between the two bitlines will change positively or negatively and the sense amplifier will then measure the contents of the cell. Figure 2.3 shows the sense amplifier ($S0$) with the bitline pair ($B0$, $B0^*$). In the DRAM model only one wordline ($WL0$, $WL1$, $WL2$, $WL3$) can be enabled at a time.

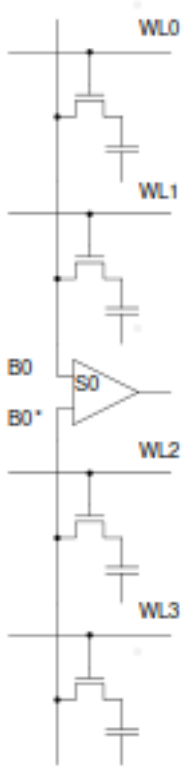


Figure 2.3: Bitline pair with sense amplifier.

When the sense amplifiers measure the bitline voltage difference rising above $V_{cc}/2+V_{th}$ a logical 1 is measured and conversely when the sense amplifiers measure the voltage dropping under $V_{cc}/2 - V_{th}$ a logical 0 is measured. Here V_{th} is a threshold voltage.

2.1.2 Arrays and Banks

Multiple cells are placed in an array. As seen Figure 2.4 the DRAM cells are placed in an array form. Multiple transistors can be enabled by a single wordline in a row, thus the term rowline is also used. The bitlines are shared by multiple cells in a column. Also seen are the sense amplifiers these are placed between the two arrays and are connected to two bitlines each. When a row is activated (wordline) the sense amplifiers will measure a voltage difference between the two bitlines and thus determine if a logical 0 or 1 was stored in the cell.

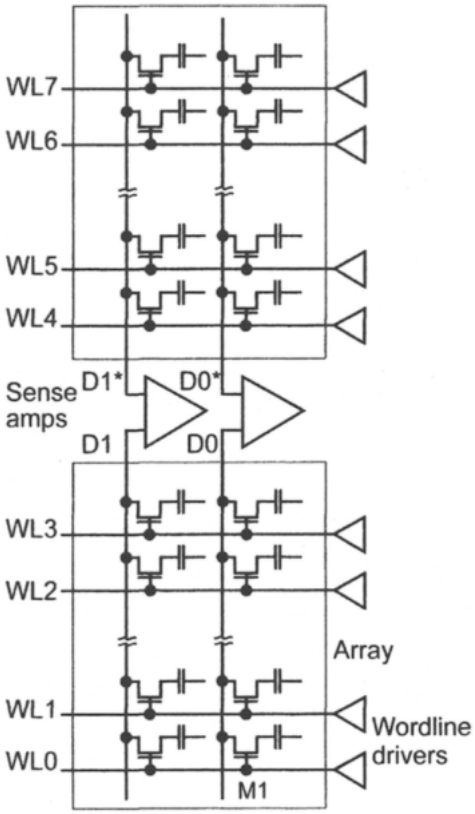


Figure 2.4: Open DRAM array with sense amplifiers [8].

The arrays can be implemented in different ways. [8] discusses the open and folded array architecture. Figure 2.4 shows an open array architecture and Figure 2.5 is a folded array. The more modern DRAM implementations use the folded array architecture.

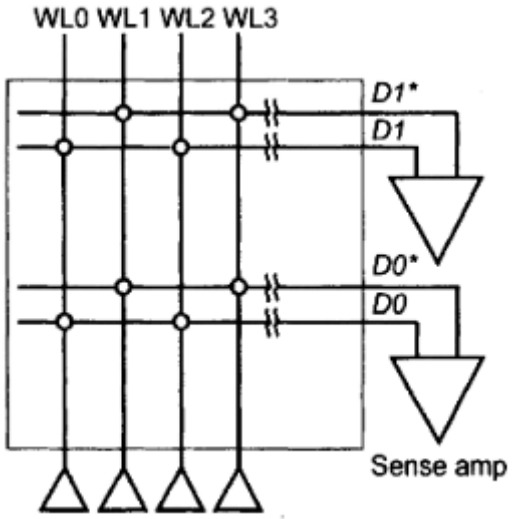


Figure 2.5: Folded DRAM architecture [8].

Multiple sub arrays form a larger memory array. These memory arrays reside in a bank. DRAM chips have multiple banks within them. These banks are separate from one another. It is possible to operate multiple banks at a time. It is prohibited to activate two wordlines (rows) in a single bank though it is possible to activate multiple wordlines in separate banks. During DRAM operation it is common practice to switch between banks.

2.1.3 Control Logic

The DRAM arrays can be seen as a two-dimensional space. To access the information in a specific location one needs two coordinates. These coordinates are the row number and column number. Control logic is required to grant access to these specific locations. The row control logic will enable and disable the wordlines whilst the column control logic buffers the sense amplifier values and provides these to the output depending on the selected column. In Figure 2.6 the basic architecture of the control logic is shown. Here the a single location in the memory array is being accessed by asserting the row address onto the row decoder and then selecting the desired column in the column mutex.

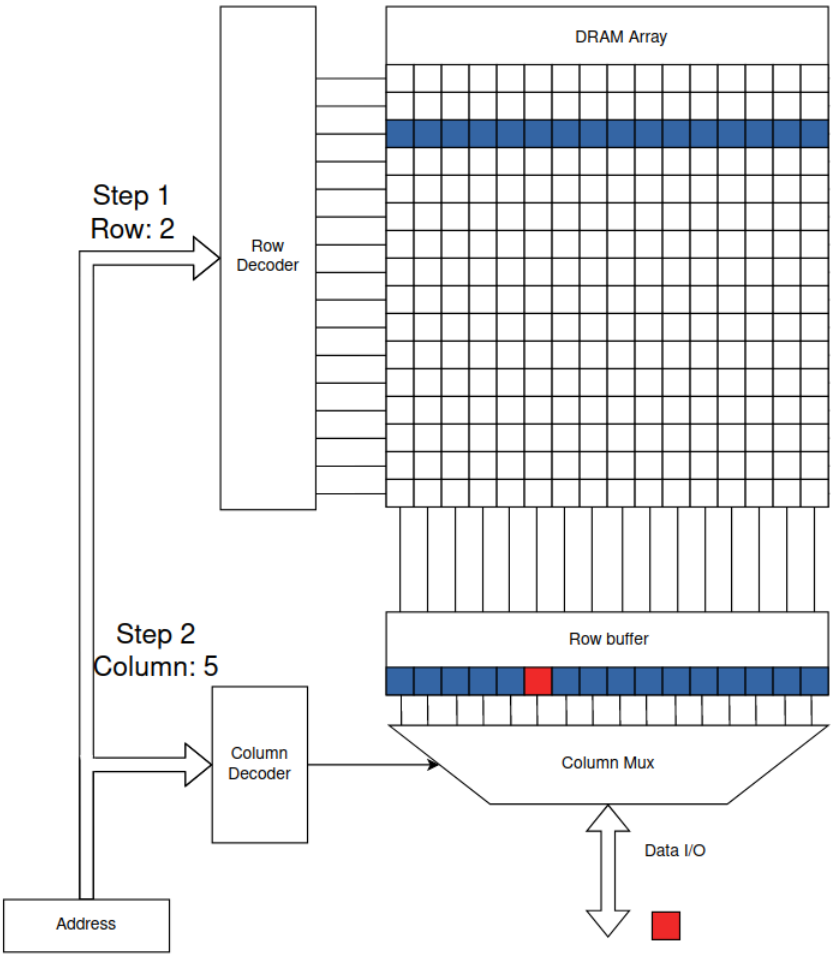


Figure 2.6: Basic control architecture accessing a single location.

2.1.4 Operations

The sole purpose of DRAM is to store and retrieve information. This can be seen as writing and reading from DRAM. A C program that accesses and changes contents in a data array will probably have load word *lw* and save word *sw* instructions in its assembly code. To execute these instructions the DRAM controller will have to perform numerous memory commands in sequence. The basic commands are:

Activate is used to drive the cell contents in the rows to the row buffer and thus requires the row address. During this operation the desired wordline is enabled and the

bitline voltages starts to change. The sense amplifiers measure the voltage difference and thus the cell contents. This operation is destructive to the cell contents in the row and only one row can be active at a time in a single bank.

Read is used to select the column in the row buffer. The row buffer essentially holds the values measured by the sense amplifiers. The selected column contents are forwarded to the output.

Write is similar to read. The column selects the location where the new data should be inserted. The cell contents are not directly changed during this operation and instead the row buffer contents are altered at the specific column location.

Precharge is used to close the DRAM row. This is essentially the reverse of the activate command. Here the bit values in the row buffer are driven back to the cells. Then the wordline is closed and the bitline voltages are restored to $V_{cc}/2$.

Refresh is used to keep the the memory cell charges in the memory bank fresh. Memory cells lose their charge over time and and at some point will fall to a degree where the sense amplifiers cannot measure the correct value. The refresh command ensures that the cell charges are kept to a proper level. It is commen practice for DRAM to refresh each bank every 64 ms. Refreshing essentially activates and precharges each row in the bank consecutively.

The flow of these DRAM commands is represented in Figure 2.7 where the basic DRAM commands are shown as transition between states. Some states transition without an issued DRAM command such as *precharge* going to *idle* and *activating* going to the *bank active* state. After *reading* or *writing* the DRAM will go into the *bank active* state.

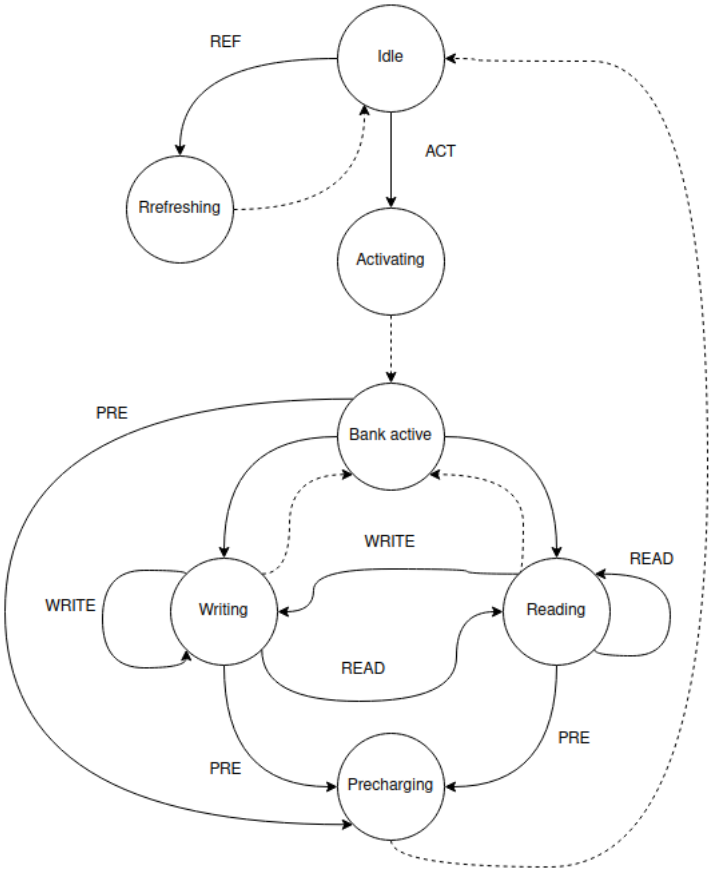


Figure 2.7: State machine basic DRAM commands

It is important to note that it is possible to execute the activate and precharge commands in sequence. Though, in practice this will not happen during the execution of a C program as this will consist of read and/or write commands in between the activate and precharge commands.

2.2 Rowhammer

As already stated in the introduction, Rowhammer is a security exploit that takes advantage of a circuit level DRAM vulnerability where repeatedly accessing (i.e., hammering) a DRAM row can cause bit flips in physically nearby rows. This allows an attacker to manipulate data in the DRAM memory without having direct access to it.

The basic concept of rowhammer relies on having an attacker row and a victim row. These two rows are physical adjacent to one another. By repeatedly accessing the attacker row bit flips will occur in the victim row. The victim row is not accessed by the control logic during the rowhammer attack. The role of the row buffer is to store a whole row for the control logic to select the desired column. If an attacker only accesses a single row, the attacker row, the rowhammer attack will be unsuccessful. The reason for this is that the contents of the attacker row are stored in the row buffer and therefor a rowhammer attack is executed by alternating rows. There are mainly two types of rowhammer attacks, a one-sided and two-sided attack. During a one-sided rowhammer attack only one row is the attacker and multiple neighbouring rows can be the victim. During this attack an attacker row and a random row must be selected in order to overwrite the contents in the row buffer. The two-sided rowhammer attack uses two attacker rows that are neighbours to a victim row and by using two attacker rows in sequence the row buffer will be rewritten during every row access. Figure 2.8 shows the difference between a one-sided and two-sided rowhammer attack. Here the attacker row cells are shown in black and the victim row cells are shown in orange.

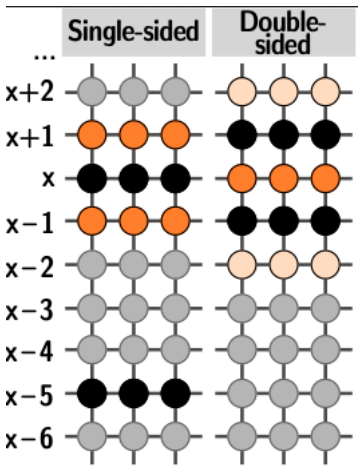


Figure 2.8: One-sides vs Two-sided Rowhammer [4]

One of the first papers to discuss the phenomenon of rowhammer is [1]. The source discusses the possibility to corrupt nearby DRAM rows by frequently accessing rows. They managed to successfully flip bits on an Intel and AMD system and revealed that multiple DRAM modules from different manufacturers where vulnerable to this problem.

Assembly code was used to for executing DRAM memory access patterns. Here it was revealed that two rows must be accessed to induce bit flips. The access patterns that successfully induces bit flips are due to frequently opening and closing of a row and not due to the frequency of reading and writing operations. This means that the DRAM commands activate and precharge are responsible for causing bit flips while read and write don't seem to have much influence. This is understandable seeing that only activate and precharge directly influence the contents of the DRAM cells (destruction and restoration) whilst the read and write commands are limited to interacting with the row buffer.

2.2.1 Data Pattern

Rowhammering with different data patterns show varying results. In [1] and [3] 4 types of data patterns have been used in a rowhammer experiment with different DRAM modules from varying manufacturers. The data patterns that were investigated are Solid, Row stripe, Columnstripe and Checkered. These patterns were also inverted and tested. In [3] it is shown that the number of occurring bit flips varies across data pattern for different types of DRAM modules and manufacturers. As seen in Figure 2.9 some data patterns induce more bit flips for different types of DRAM than others.

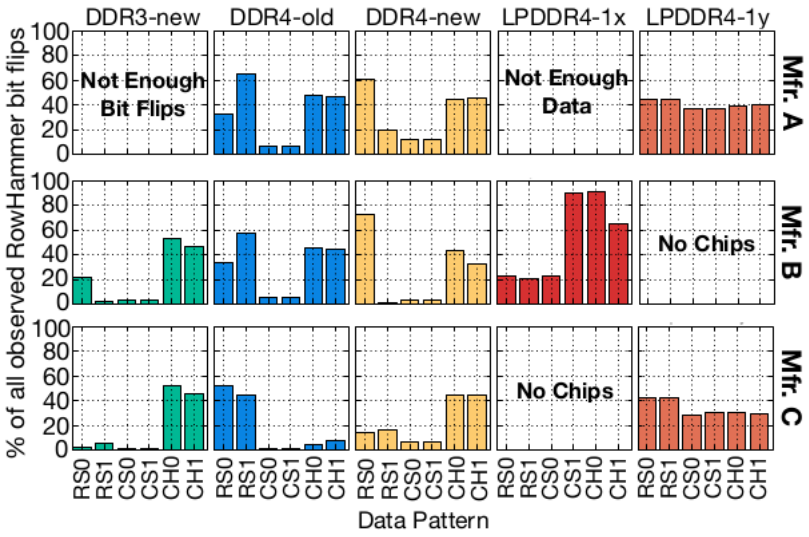


Figure 2.9: Bit flip coverage vs data patterns with different types of DRAM and 3 types of manufacturers [3]

A visual representation of data patterns is shown in Figure 2.10. These data patterns are loaded into a memory bank before performing a rowhammer attack. The solid 0 and solid 1 are shown explicitly whilst the other three data patterns Checkerboard, Columnstripe and Rowstripe are shown in a single form. The latter three can also be inverted to cover the other bits in the memory array

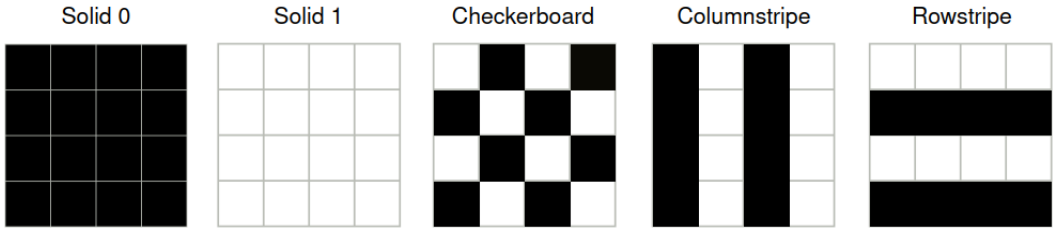


Figure 2.10: Data pattern types. With black uncharged cells and white charged cells

2.2.2 Failure Mechanism

Bit flips are shown to be one way. Meaning that a bit cannot flip twice during a rowhammer attack. The [5] provides an insight into bit flips at the cell level. A logical bit flip might occur from $0 \rightarrow 1$ or from $1 \rightarrow 0$ though according to the paper [5] vulnerable cells can only lose charge during rowhammering. The reason that cells flip from $0 \rightarrow 1$ is that these are anti-cells. These cells logically represent the opposite of their cell charge. During the research the victim and attacker cell were tested with various cell charge combinations. A rowhammer attack was performed with the victim and attacker cell both being charged, both being uncharged and finally having an opposite charge. It was shown that most successful flips occur when the attacker cell is uncharged and the victim cell is charged. There were also successful flips when the attacker and victim cell were both charged. No bit flips were found when the attacker and victim were both uncharged and finally no flips were found when the attacker was charged while the victim was uncharged. The reason provided was that during an activate-precharge cycle electrons travel from the attacker to the victim cell and this only seems to occur with a charged victim cell. A similar topic is covered in [17] where a trap-based failure mechanism is discuss. Here electrons are trapped in the attacker row transistors after disabling the wordline. These electrons become de-trapped after time and travel to the victim cell capacitor. This shows that with each rowhammer the potential in the victim capacitor drops. As seen in Figure 2.11 the path of the electron is shown to travel to the victim capacitor.

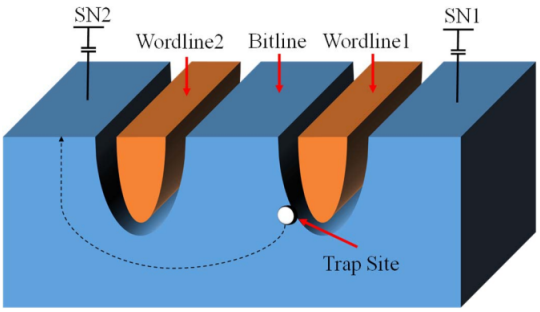


Figure 2.11: Electron transport from trap site to victim capacitor [17]

A cell's retention time does not predict if a cell is vulnerable to rowhammer. In [5] it is shown that there are cells that have a low retention time while having a higher resilience to rowhammering and vice versa. Figure 2.12 shows 5 cells taken out from two groups. In group A there are cells that have a high resilience to rowhammering while in group B the cells have a high retention time.

Type	Cell	t_{RET}	$N_{APH\text{-}FTH}$	Description
	Cell A-1	**3.1 s**	50,500 K	
	Cell A-2	**3.1 s**	54,500 K	
A	Cell A-3	**3.3 s**	43,500 K	The worst t_{RET}
	Cell A-4	**3.5 s**	24,500 K	
	Cell A-5	**3.5 s**	55,000 K	
	Cell B-1	over 20 s	**197 K**	
	Cell B-2	over 20 s	**197 K**	
B	Cell B-3	over 20 s	**233 K**	The worst $N_{APH\text{-}FTH}$
	Cell B-4	over 20 s	**235 K**	
	Cell B-5	over 20 s	**238 K**	

Figure 2.12: Two groups showing cell retention time and Hammer resilience [5]

Wordline-to-wordline cross-talk is another possible reason for cells to lose their charge during rowhammer as the neighbouring wordlines are somewhat capacitively coupled. Under certain scenarios toggling a wordline might cause the voltage on neighbouring wordlines to rise ever so slightly allowing the transistors to drain the capacitors. [7] goes into great depth explaining this cross-talk. The takeaway from this paper is that the newer generations of DRAM suffer most from this failure mechanism.

Temperature is another factor that plays a role during rowhammer. In [6] it is explained that a higher temperature increases the bit-error-rate during rowhammer. The following Figure 2.13 shows the trend of the bit errors during rowhammer for an increased temperature. At least 3 of the 4 DRAM devices coming from different manufacturers suffer from an increasing bit-error-rate (BER) for an increasing temperature.

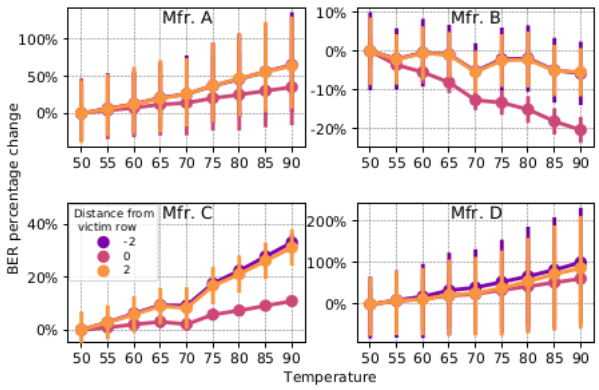


Figure 2.13: Impact of temperature on BER (Bit-error-rate) for 4 manufacturers [6]

In the same paper [6] the impact of an increased attacker row activation time is shown to increase the vulnerability of the victim row to rowhammering. As shown in Figure 2.14 the increase of the attacker row activation time results in a higher bit-error-rate in the victim rows. What is interesting to note [6] also mentions that increasing the precharge time decreases the vulnerability of the victim cells.

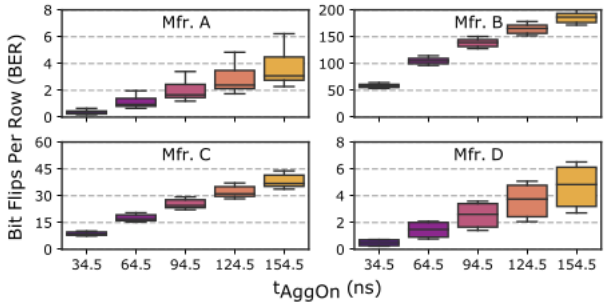


Figure 2.14: Impact increased attacker row activation time for 4 manufacturers [6]

Two types of failure mechanisms seem to cause these negative effects during rowham-

mer. Namely the injection and capture of electrons during a repeated activation-precharge cycle and the capacitive disturbance of wordline-to-wordline cross-talk. Furthermore it is demonstrated that an increasing temperature and an increasing row activation time both have an impact on the vulnerability of a cell during rowhammer.

Chapter 3

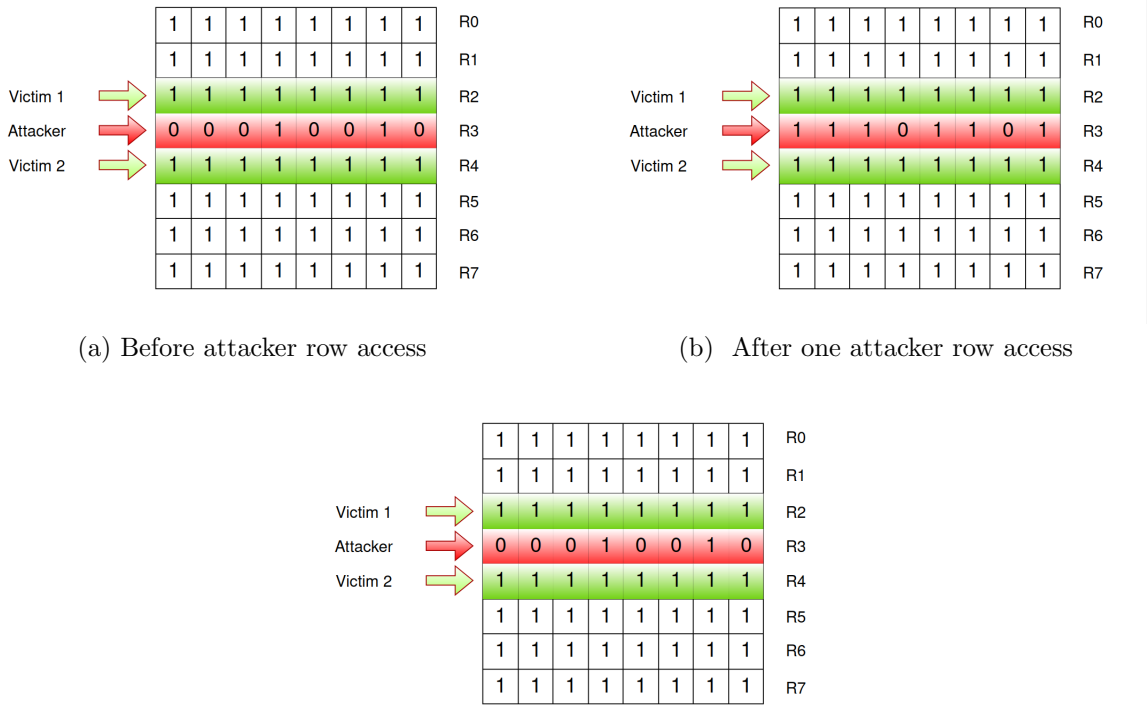
Proposed Protection Method

In this chapter a protection method is proposed, where the goal is to prevent the negative effects caused during rowhammer attacks by means of flipping the data in the attacker row. First the protection method is discussed. After this three design options are discussed for implementing the Row Flipper protection method.

3.1 Row Flipper Protection Method

Strong evidence suggests the success to flip a bit in a DRAM row during rowhammer is due to specific cell contents of the victim and attacker rows. The chance to flip a charged victim cell is higher when an attacking cell is uncharged. If a victim cell is uncharged there is no possibility to flip it regardless of the attacking cell's charge. Unfortunately one cannot alter the victim row's content preemptively before an attack, as one cannot predict the targeted victim row easily and assume the attacking row's content throughout such an attack. The easier option is to alter the attacker row's data in such a way that rowhammering becomes unfeasible.

During a single-sided rowhammer there are two victim rows and one attacking row. The attacking row will be accessed throughout the attack and will thus need to be flipped by the protection method to prevent bit errors in the victim rows. To illustrate this Figure 3.1 shows the transition of the attacker row during a single sided rowhammer attack. Here the initial values of the attacking row as well as the victim rows are shown in Figure 3.1a. Once the rowhammer attack has begun and the attacker row has been accessed the method will flip the attacker row contents as seen in Figure 3.1b. The attacker row is flipped again after the second access as seen in Figure 3.1c. This will continue throughout the rowhammer attack.



(a) Before attacker row access

(b) After one attacker row access

(c) After two attacker row accesses

Figure 3.1: Attacker row content flipping during single-sided rowhammer

In the case of a two-sided rowhammer attack there are three victim rows adjacent to the two attacking rows. There is one row specifically that will receive an extra hard hammering as it sits in between the two attacking rows. As mentioned earlier to perform this two-sided rowhammer the attacking rows must be accessed in alternating order to ensure the row buffer is overwritten. Figure 3.2 shows the flipping of data in the attacking rows throughout the rowhammer attack. The initial content is shown in Figure 3.2a, here the attacking rows are spaced with one victim row in between and two victim rows on the outside. After the first row has been accessed, *Attacker 1*, its contents are flipped by the method, as shown in Figure 3.2b. The following access is performed on *Attacker 2* and its contents are in turn flipped by the method as shown in Figure 3.2c.

(a) Before attacker row accesses

(b) After attacker 1 row access

(c) After attacker 2 row accesses

Figure 3.2: Attacker row content flipping during two-sided rowhammer

3.1.1 Expected Results

The aim of the Row Flipper protection method is to maintain the victim cell's voltage level above the threshold voltage. To give an example Figure 3.3 shows the victim cell losing charge during a rowhammer attack with and without the Row Flipper protection method. Also shown is the victim cell losing charge over time under normal circumstances. The aim is to ensure that the cell's voltage remains above the threshold voltage, *V_th*, for the sense amplifier to measure the correct cell contents within the DRAM's refresh period. The example shows the refresh period of 64 ms. When the victim is attacked without the Row Flipper protection method its charge drops below the threshold within the 64 ms refresh interval. In this case the sense amplifier will measure an incorrect value and thus the victim cell bit has been flipped. When the attacking row data is flipped throughout the rowhammer attack the victim cell voltage will drop at a lower rate and thus will keep its charge above *V_th* within the 64 ms refresh interval.

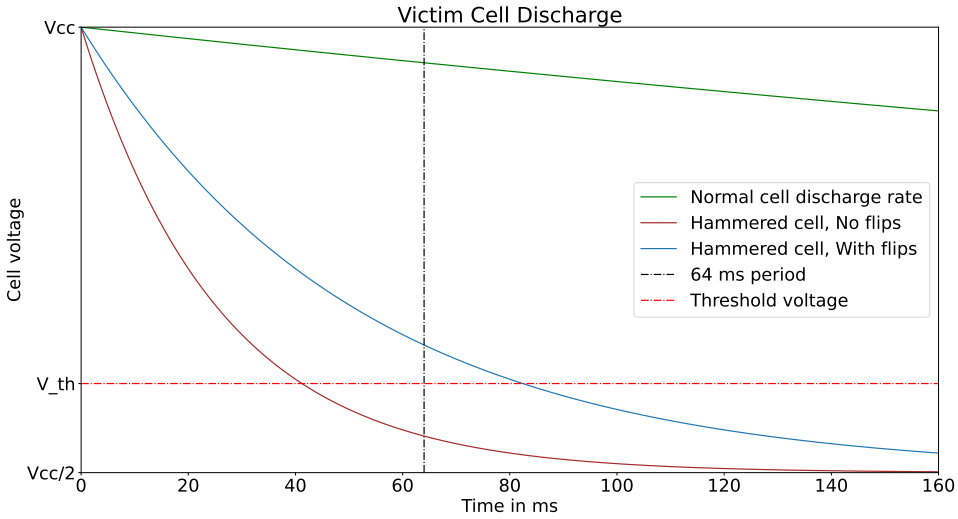


Figure 3.3: Victim cell discharge under different circumstances

Certain victim cells may be extremely vulnerable to rowhammer attacks and will fail regardless. Even though the cell's charge will drop under *V_th* within the 64 ms the protection method will still make it harder for the attacking row during rowhammer. The attacker will have to increase their efforts by increasing the number of hammers performed on the victim row to induce a successful bit flip in the victim row. This will decrease the overall potential of inflicting bit errors in the memory. The Row Flipper protection method could also be used in combination with other protection methods to protect the memory, such as throttling of attacker row accesses or an increased refresh rate (lowering refresh period time).

3.2 Exploring Row Flipper Design Implementations

The basic idea of the protection method has been proposed with the expected results. In this section the implementation design criteria will be discussed. A software implementation concerning the CPU, memory controller implementation and on chip implementation are explored.

3.2.1 Software Protection Method to run on CPU

The kernel is part of the operating system and sits between the user space applications and hardware components. It has access to the memory controller *MC* as seen in Figure 3.4. The user processes run within the operating system and the kernel ensures that memory operations are processed, and that each user process has access to its data within memory. Furthermore it ensures that no other user process can access and alter another processes data.

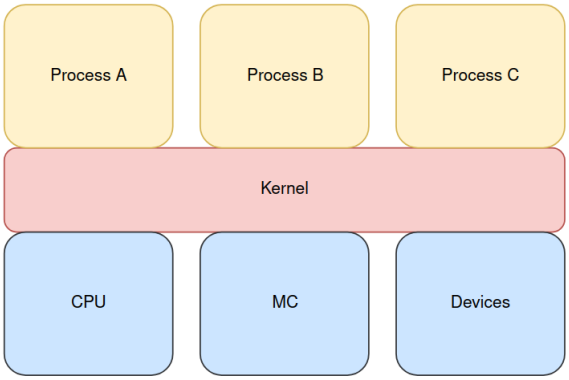


Figure 3.4: Kernel with processes and hardware

The software version of the Row Flipper protection method must be implemented in the kernel. When a malicious user process is performing a rowhammer attack it is expected to send memory requests very frequently. The malicious process only has to request access to a single byte in memory as the DRAM module is designed to activate an entire row providing access to the single byte. Not knowing the exact structure of the DRAM module the kernel stands on very limited ground in order to protect the victim cells. The row size of the DRAM module is usually unknown to the kernel as the operating system is created to run on multiple different devices possibly operating different DRAM components from different manufacturers. To explain the issue in Figure 3.5 the attacker is accessing a specific byte in the DRAM. Here the kernel is limited to the address of the byte and is not aware of the total number of bytes in a DRAM row, so the kernel does not know which other victim locations are being affected. As the row width is not known the attacker row data cannot be flipped entirely by the kernel. This, together with many other issues discourages the protection method to be implemented as a software solution.

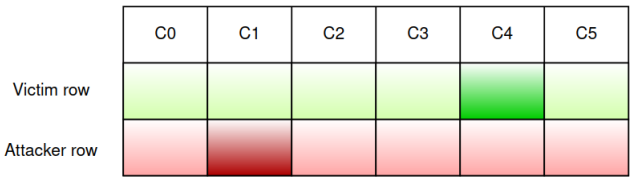
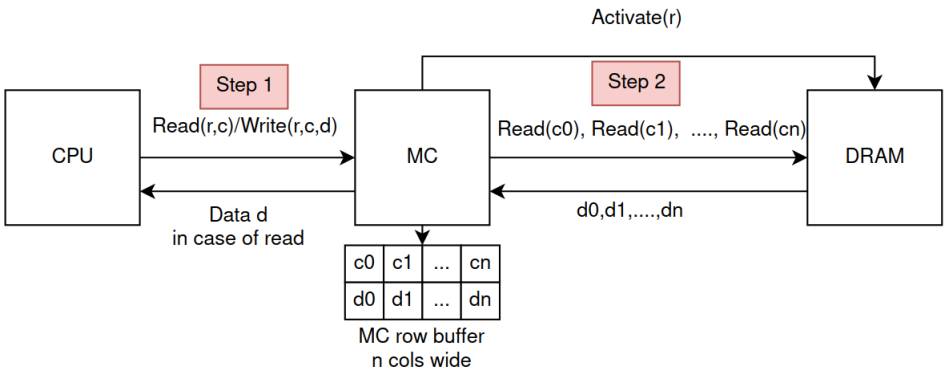


Figure 3.5: Attacker accessing byte in C1, Victim affected in C4

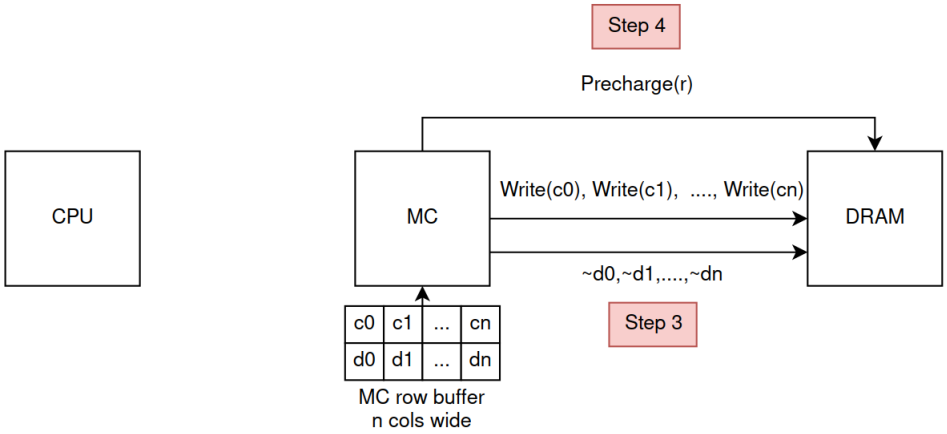
3.2.2 Memory Controller Modifications

The memory controller interacts with the DRAM module and sends DRAM commands to read and write data. The memory controller being closer to the DRAM module has a better understanding of the exact workings of the DRAM module. It is developed for a specific type of DRAM (such as DDR3, DDR4, etc), knows the DRAM storage size, the timing delays of certain command and so forth. The exact internal layout of the DRAM module however is shielded from the memory controller. The memory controller could perform specific tests on the DRAM device to get an understanding of its internal memory array structure. For example a test to get the exact row size and a test to discover true- and anti-cell locations.

There is however a limitation. The memory controller cannot flip an entire row in DRAM instantly. This is because the memory controller is limited to performing specific DRAM commands. For the memory controller to flip the attacker row data during a rowhammer attack it will have to first read the entire attacker row, save it in a buffer and then write the flipped data back into the attacking row. Figure 3.6 shows the memory controller receiving a read or write operation from the CPU and the steps it takes to perform the flipping of an attacker row. In Figure 3.6a at *step 1* the memory controller receives a read or write command from the *CPU*. Regardless of which of the two commands the *MC* receives it will have to read the entire DRAM row. Continuing, in *step 2* the *MC* activates the row and the columns *c0* to *cn* are read with n being the last column. The data is buffered in a register accessible by the *MC*. If the *CPU* had sent a read request in *step 1* it will receive the requested data, and if a write command has been issued the data will be stored in the buffer. The data flipping of the attacker row is performed in Figure 3.6b. Here the request of the *CPU* has been fulfilled and thus the row can be closed. Before this *step 3* shows that the *MC* sends write commands to the DRAM with the original data being flipped. Finally in *step 4* the row is closed by a precharge command.



(a) Activate and read whole row



(b) Write flipped contents back and precharge

Figure 3.6: Flipping attacker row contents in memory controller

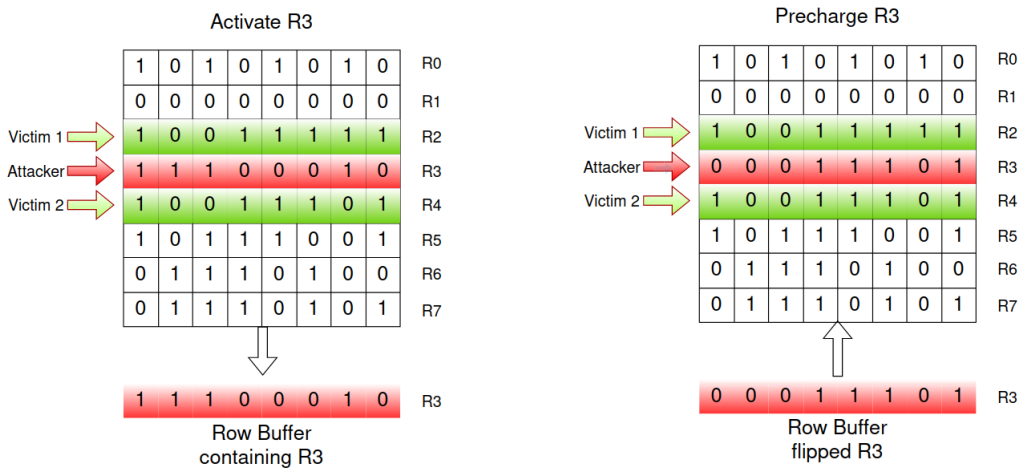
This implementation does of course come with its drawbacks. A row will have to be read entirely and have its contents flipped and written back. This will have a great impact on the throughput of the device. Even if the CPU requests reading the data of an entire DRAM row the overhead increase will be doubled as writing the flipped content back to the DRAM row will require roughly the same amount of time as reading the entire DRAM row (assuming a DRAM read operation takes the same amount of time as a write operation). Aside from the extra overhead a choice must be made between keeping track of attacking rows or keeping track of the flipped rows. Meaning that the protection method can be implemented in two ways. The first is to identify and keep

track of attacking rows, in this case only the attacking rows will be flipped. The second option is to flip all rows being access regardless whether the memory requests are driven by malicious or non-malicious CPU processes. To ensure proper functioning for all user processes the memory controller must keep track of the flip status of all DRAM rows. Modern DRAM modules have 100,000 plus rows so extra internal memory must be added to the memory controller.

3.2.3 Modified DRAM Chip Design

An option is to implement the Row Flipper protection method in the DRAM chip. The DRAM row can be flipped in one instance, something that is not possible for a software and memory controller design. This will be done by flipping the content in the row after the row has been accessed. Then, once the row is being closed by the precharge command the following happens. Instead of restoring the cells' content to their original value, they are logically inverted and the flipped content is written into the cells. To clarify an example is given. As seen in Figure 3.7a, the attacker row *R3* has the potential to flip bits in the neighbouring rows *R2* and *R4*. The attacker row is being activated and its contents are stored in the row buffer. After this the row buffer contents are flipped when the attacker row is precharged and the row will store the logical opposite of its original contents as seen in Figure 3.7b.

Compared to the previous design options the DRAM module is fairly limited in terms of flexibility. Where the other components could have the option to implement a mechanism to detect malicious memory access patterns adding this functionality to a DRAM chip would not be feasible. The feasible option is to keep track of the flip status of each row within the DRAM memory to ensure data correctness. Here a table keeps track of the row's flip status. The table value is a single bit per row that is 0 to indicate the row is not flipped and 1 to indicate that the row has been flipped. When reading a column from the data buffer the bits may need to be flipped in order to provide the correct output value, this will depend on the table's value corresponding to the row. An example of this design is shown in Figure 3.8. The *Flip Table* is shown next to the *DRAM Array*. The wordlines are connected from the *Row Decoder* through the *DRAM Array* to the *Flip Table*. The *Row Flipper* component sits in between the *DRAM Array* and the *Row buffer*. This *Row Flipper* component essentially flips every bit going to, or coming from the row buffer depending on the *Flip Register* value.



(a) Activating attacker row R3

(b) Flipping row buffer and precharging row R3

Figure 3.7: Flipping attacker row contents in DRAM array

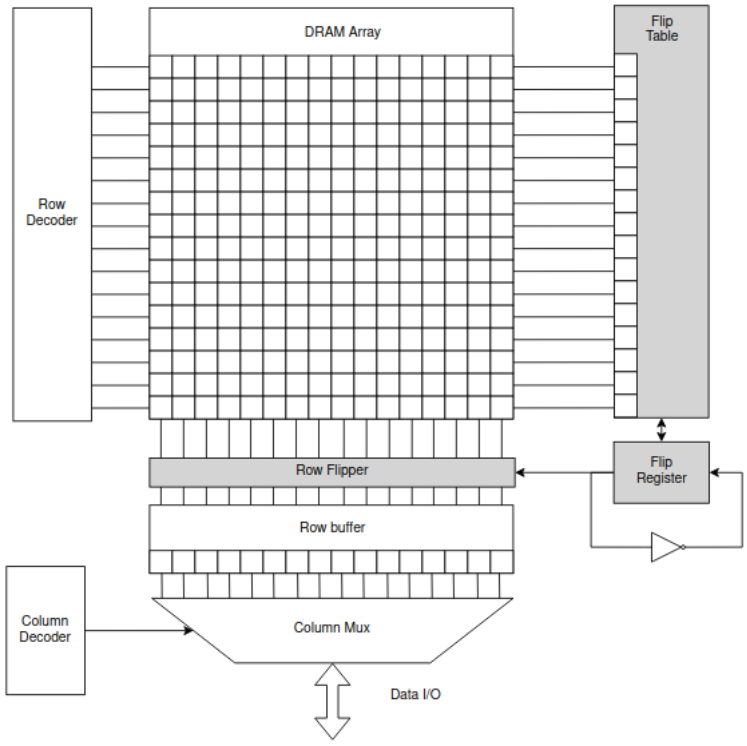


Figure 3.8: DRAM with flipping component and flip table

The Row Flipper protection method will work as follows. When a row is activated the corresponding *Flip Table* value is written into the *Flip Register* register. The value in this register determines if the row buffer contents must be flipped or not. When a read operation is performed the correct value is set to *Data Out*. When a precharge is issued the *Flip Register* content is negated. This in turn determines whether the content in the row buffer must be flipped. The *Flip register* value is stored in the corresponding *Flip Table* index associated with the row, and the row buffer content (flipped or not) is inserted in the memory array.

Some drawbacks are to be expected. For example, there will be extra overhead in the form of area and performance. The flip table will require area, this could be area that would otherwise be used for the main goal of the DRAM device (to store memory). Adding hardware to flip the row content will increase the row access latency.

Chapter 4

Methodology & Design

This chapter will discuss the methodology and design. The methodology is the road-map for investigating the thesis topic and explains the steps taken to evaluate the results. In this section the experiments are discussed together with the metrics in order to measure the results. After this the experimental infrastructure design is given. here the design requirements are listed together with the hardware and software designs. The list of experiments together with the design can be used as basis for other studies to follow.

4.1 Approach

To reach the goal of this thesis some essential steps need to be taken to understand the problems evolving from rowhammer. There are several objectives that are essential for reaching the conclusion. To illustrate this Figure 4.1 shows the steps that need to be taken to provide a final conclusion. As seen some do not directly impact the conclusion, they should rather be seen as parts of the puzzle. The following subsections will discuss these objectives in detail.

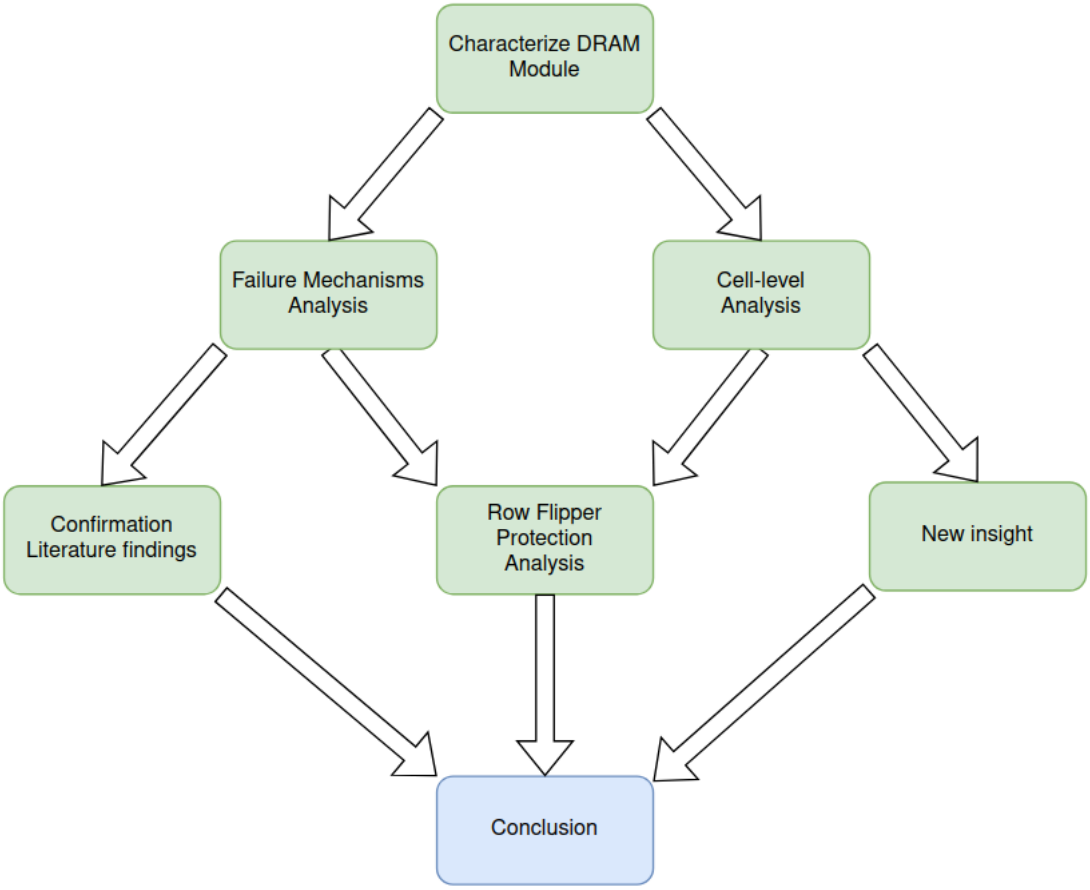


Figure 4.1: Steps leading to the conclusion

4.1.1 Characterize DRAM Module

Characterizing the DRAM module is the first step that needs to be taken as it is important to understand what one is working with to be able to draw the correct conclusions. During this step the DRAM module to be used is tested under normal working conditions to make sure that the experimental setup functions. Thereafter, the physical layout of the DRAM module is investigated. Here the DRAM datasheet together with the reverse engineering of the DRAM module will provide the information required to proceed with the experiments.

4.1.2 Failure Mechanism Analysis

The literature discussed in the background has shone a light on the failure mechanisms that are thought to cause the flipping of bits during rowhammer. Two of these will be investigated, wordline-to-wordline cross-talk and electron de-trapping. The results will provide an insight into the effects of the failure mechanisms that are at play during rowhammer for the specific DRAM module used throughout the experimentation. This will be achieved by performing rowhammer attacks with predetermined parameters.

4.1.3 Confirmation Literature Findings

After the failure mechanisms have been analysed the results will be compared to those found in literature as it is important to understand if the DRAM module used during the experiments is suitable to perform further tests on. If the results differ to what has been found in literature this should be taken into consideration.

4.1.4 Cell-level Analysis

In this step experiments specifically investigating the victim and attacker cells are executed. This might result in new findings. Other than previous steps that analyse the effects of rowhammer on a broader scale (for example row and column) this step will analyse the effect on very specific cells during rowhammer. The aim will be to find the minimum hammer resilience, to alter direct neighbouring cells of the attacker/victim and to find any relation between them.

4.1.5 New Insights

The outcome of the cell-level analysis may reveal some new results not discussed before. These might be usable for future work on the topic of DRAM and rowhammer. In the case that there are new findings these will be discussed and proposed for future work.

4.1.6 Row Flipper Protection Analysis

The general idea of the Row Flipper protection method has been discussed. In this step the Row Flipper protection method will be simulated in experiments, and the results analyzed to obtain an indication of the effect of using this protection method.

4.2 List of Experiments

The experiments are categorized in three sets. Experiment set 1 is used to characterize the DRAM memory module. Experiment set 2 investigates the cell level impact of rowhammer together with the cells neighbouring the victim and attacker cell. These will also play an important role. The final set 3 will investigates the proposed protection method implemented in a memory controller.

4.2.1 Experiment set 1: Reverse Engineering

This set of experiments provides an understanding of the physical implementation of the DRAM module. Without knowing the physical characteristics it is impossible to draw correct conclusions. The experiments will provide a detailed insight into the following:

- The location of true- and anti-cells
- The exact row-size
- Rows neighbouring each other
- Columns neighbouring each other
- Hammerable cells within refresh period

Each experiment is created to provide information of a specific aspect of the DRAM module. These results will be used throughout the other two experiment sets, namely experiment set 2: cell-to-cell level experiments and experiment set 3: protection method experiments

4.2.1.1 1a) True/anti-cell rows

True- and anti-cells are logically opposite to one another. When a true-cell has a charged value it holds a logical 1 and discharged it holds the value 0. Anti-cells are the opposite, charged is a logical 0 and discharged a logical 1. It is known that capacitors lose their charge over time and that is why DRAM needs to be refreshed periodically (usually within 64 ms). To find the true/anti-cells a solid data pattern will be loaded into the memory bank, then the DRAM will be left idle for a fixed time with auto-refresh disabled. After this the whole memory bank will be read and all the logical cell values will be compared to the initial data pattern. The pseudo-code of this experiment is shown below in algorithm 1.

Algorithm 1 Find true/anti-cells

```
TrueCellRowSet ← {}
AntiCellRowSet ← {}
row ← #rows
DataPattern ← 0xff                          ▷ Put solid data pattern all 1's
WriteWholeBank(DataPattern)                 ▷ Function to write to whole bank
Sleep(300)                                  ▷ Wait 5 minutes
MemoryArray ← ReadWholeBank()
while row ≠ 0 do
    NrErrBits ← CountErr(DataPattern, MemoryArray(row))
    if NrErrBits ! = 0 then
        AddToSet(TrueCellRowSet, row)
    else
        AddToSet(AntiCellRowSet, row)
    end if
    row ← row − 1
end while
```

This experiment can also be run with the solid 0 data pattern to find the anti-cells, these will flip from $0 \rightarrow 1$ after reaching their retention time.

4.2.1.2 1b) Size of Rows

To verify the size of the rows in the DRAM bank an experiment is performed based on the previous experiment. A column stripe data pattern is inserted into the bank (with this data pattern there is no need to keep track of true/anti-cell row locations as both row types will have charged cells in them) and only one row is set as active, auto-refresh is disabled and the rest of the bank is left to discharge over a time period. When reading the whole memory bank and comparing it to the initial data pattern only the chosen row should show 0 bit errors. An example of the experiment is shown in algorithm 2.

Algorithm 2 Size of rows

```
ErrFreeRowSet ← {}
row ← #rows
DataPattern ← 0x55                          ▷ Column stripe data pattern
WriteWholeBank(DataPattern)                 ▷ Function to write to whole bank
Activate(ChosenRow)
Sleep(300)                                  ▷ Wait 5 minutes
Precharge(ChosenRow)
MemoryArray ← ReadWholeBank()
while row ≠ 0 do
    BitErrRow ← CountErrRow(DataPattern, MemoryArray(row))
    if BitErrRow == 0 then
        AddToSet(ErrFreeRowSet(row))
    end if
end while
```

This experiment assumes that there will be at least one charged cell in every single row to flip during the time period.

4.2.1.3 1c) Finding Neighbouring Rows

In this experiment the neighbouring rows will be found by performing a rowhammer attack. A one-sided rowhammer attack will be performed for each row. Two rows must be chosen to perform a rowhammer attack. The rows chosen are the row address and the negated row address, assuming that these are not neighbours and are spaced far apart. After each round the row address is incremented by one. For example, if the row addresses input ranges from 0 to 511 the first round will hammer the rows 0 and 511, the next round 1 and 510 and so on. It is understood that the rowhammer attack creates most bit errors in the victim rows when the attacker has discharged cells and the victim has charged cells, so the attacker row will be loaded with a negated data pattern. To run this experiment independently from 1a this experiment should be run with two data patterns (namely, solid 0 and solid 1) and only the attacking rows should have an opposite data pattern. The neighbouring rows will be determined by their bit error rate being higher than 0. The code in algorithm 3 shows the basics of the experiment. The result will be a set of rows with their neighbouring rows.

Algorithm 3 Neighbouring rows Rowhammer

```
DataPatternSet ← {0x00, 0xFF}
RowNeigbourSet ← {}
for DataPattern in DataPatternSet do
    row ← 0
    while row < #rows/2 do
        WriteWholeBank(DataPattern)
        WriteRow(row, ¬DataPattern)
        WriteRow(#rows − row, ¬DataPattern)
        HammerCount ← #MaxHammerCount        ▷ Set for maximum interference
        while HammerCount > 0 do
            Activate(row)
            Precharge(row)
            Activate(#rows − row)
            Precharge(#rows − row)
            Hammercount ← Hammercount − 1
        end while
        MemoryArray ← ReadWholeBank()
        FirstErrorRowList ← ListFirstHalfErrorRows(MemoryArray)
        SecondErrorRowList ← ListSecondtHalfErrorRows(MemoryArray)
        AddToSet(RowNeigbourSet((row, FirstErrorRowList))
        AddToSet(RowNeigbourSet((#rowsrow, SecondErrorRowList))
        row ← row + 1
    end while
end for
```

At the end of this experiment all rows impacting each other should be known. The direct neighbours should have the highest bit-error-rate though second neighbours will also be logged in this experiment.

4.2.1.4 1d) Finding Neighbouring Columns

The victim and attacker row location will be required for this experiment. Here the attacker will perform a rowhammer attack with a walking 1/0 in its row. The victim row will be loaded with a solid 1 or solid 0 data pattern (depending on the type of cells in the row) and the attacking row will be loaded with the same pattern except for one column cell that is used as an attacker cell. At the end of the rowhammer attack the victim row will be analysed for bit flips. The next round will reset the victim and attacker

rows to the set data pattern and have the next attacker column bit in the attacker row inverted. The victim row could show 3 results after being attacked, namely:

1. No bit flips could occur during the rowhammer attack.
2. The column bit in the attacker row affects the exact same column in the victim row.
3. A bit flip occurs in a different column location.

The following algorithm 4 is the basis for the walking 0 in the attacking row. This algorithm can also be used with the *Solid 0* data pattern to perform a rowhammer attack with a walking 1 in the attacker row.

Algorithm 4 Rowhammer with Walking 0

```
DataPattern ← 0xff                                  ▷ Example with Solid 1
AttackerColumn ← 0
while AttackerColumn < ColumnSize do
    WriteDataToRow(VictimRow, DataPattern)          ▷ Write to entire row
    WriteDataToRow(AttackerRow, DataPattern)
    NegateBit(AttackerRow, AttackerColumn) ▷ Negate the attacker bit in the row

    Rowhammer(VictimRow)                            ▷ Perform rowhammer on victim

    VictimColumn ← AnalyseRow(VictimRow, DataPattern)

    LogData(VictimColumn, AttackerColumn)   ▷ Log columns for further analysis
    AttackerColumn ← AttackerColumn + 1             ▷ Increment attacker column
end while
```

4.2.1.5 1e) Hammerable Victim Cells within Refresh Period

This experiment provides a set of vulnerable cells in the memory bank that have the potential to flip within the refresh period. Here the refresh period determines the maximum hammercount during the rowhammer attack. This experiment is very similar to Section 4.2.1.3. However, that experiment ignores the refresh period and this experiment will set the hammercount to a value that allows for the maximum theoretical $activate \rightarrow precharge$ hammer cycles within the refresh period time.

4.2.2 Experiment set 2: Cell Level experiments

Once the reverse engineering has been performed this set of experiments will explore the effects of rowhammering on a deeper level. The goal of these experiments is to understand the interaction between attacker cells and victim cells. This will be explored on a physical local level, meaning cells that neighbour one another directly.

These experiments will investigate the following:

- Verifying victim cell discharge during rowhammer
- Exploring victim cell and attacker cell combinations
- Exploring victim data pattern impact on rowhammer
- Exploring attacker data pattern impact on rowhammer
- Finding evidence of diagonal cell impact
- Finding evidence for horizontal cell impact
- Victim- and attacker cell symmetry
- The Hammer resilience of cells

Two points require more explanation. Other than just looking at the impact of the cell charges between the victim cell and attacker cell during rowhammer a slightly broader investigation will be undertaken. The victim cell has two neighbouring cells in the same row (the same goes for the attacker cell). These neighbouring cells will also be taken into the equation to see if their charge also impacts the victim cell's hammer resilience.

4.2.2.1 2a) Cell to Cell Data pattern

This experiment will cover all points except the symmetry. Here an attacker cell group and a victim cell group are defined. These cell groups consist out of the two neighbouring cells, and the cell of interest in the middle. To illustrate this Figure 4.2 shows the victim and attacker cell groups. These, are physical local neighbours of one another. Here V-Cell stands for victim cell (subsequently A-Cell stands for attacker cell). V-LN stands for victim-left neighbour and V-RN for victim right neighbour.

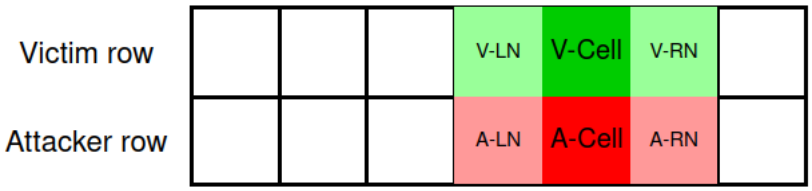


Figure 4.2: Cell groups in victim and attacker row

Each group consisting of 3 cells will be tested with 8 different data patterns $\{000, 001, 010, 100, 101, 110, 011, 111\}$. A total of 64 attacker and victim group combinations will be tested in this experiment. The data patterns will be loaded into the groups and the victim row will be hammered by the attacker. A binary search will find the minimal hammercount of the V-Cell. Each round will be rerun numerous times in order to account for disturbances. In Algorithm 5 the experiment is shown with the binary search incorporated, this algorithm will be used during the following experiments listed in set 2 and set 3.

Algorithm 5 Cell data patter rowhammer

```
DataPatternSet ← {000, 001, 010, 100, 110, 101, 011, 111}
VictimCellGroup ← Vrow, Vcol
AttackerCellGroup ← Arow, Acol
for VictimPattern in DataPatternSet do
    for AttackerPattern in DataPatternSet do
        WriteDataPattern(VictimCellGroup, VictimPattern)
        WriteDataPattern(AttackerCellGroup, AttackerPattern)
        Min ← 0
        Max ← MaxHammercount
        while Min ≤ Max do
            HammerCount ← ⌊((Min + Max)/2)⌋
            while HammerCount > 0 do
                Hammer(Attacker(0))
                if CellFlipped(VictimCell) == True then
                    Max ← Hammercount − 1
                else
                    Min ← Hammercount + 1
                end if
            end while
        end while
        LogResults(VictimPattern, AttackerPattern, Hammercount)
    end for
end for
```

The results of this experiments will shine a light on multiple points. By running this experiment for true- and anti-cell groups one can verify whether the victim cells lose charge during rowhammer and for which attacker group data pattern. Secondly diagonal cell interaction can be measured. For example understanding if the value of A-RN has impact on the V-Cell. Perhaps neighbouring cells can impact resilience of the V-Cell.

4.2.2.2 2b) Victim and Attacker Cell Symmetry

This experiment will be performed using the same algorithm as mentioned before with the 8x8 data patterns and a specific cell set is required. Here, the attacker and victim role will be reversed in that the attacker becomes the victim and the victim becomes the attacker. The two neighbouring column cells have shown to impact one another

when having the attacker role. The minimum hammercount of the neighbours will be compared to one another.

4.2.3 Experiment set 3: Row Flipper Protection Method Experiments

Basic principles of the proposed Row Rlipper protection method will be tested by means of executing specific DRAM commands. The rowhammer sequence will be adjusted to account for the flipping of data in the memory. This means that the rowhammer command sequence will be expanded with write and read commands, and that the impact of these commands must be investigated without flipping the data in the row. The experiment will be executed on a cell level and performed similarly to that in set 2 (8x8 data pattern, victim group, attacker group and binary search). Instead of executing the protection method for a whole DRAM row it will be limited to the flipping of the attacker cell group. The experiment will be performed with the following rowhammer command sequences:

- $activate \rightarrow read \rightarrow precharge$
- $activate \rightarrow write \rightarrow precharge$
- $activate \rightarrow read \rightarrow write \rightarrow precharge$
- $activate \rightarrow write \rightarrow read \rightarrow precharge$

The write command allows for the flipping of the attacker cell contents. The Row Flipper protection method can only be implemented with rowhammer sequences containing a write.

Algorithm 6 shows the rowhammer sequence with a read and write command. Here the write command writes the data pattern to the attacker group, after this the data pattern is negated and the next round the attackers cell contents are flipped.

The experiment will investigate the impact of the rowhammer command sequences with and without flipping (except for the read only sequence). Meaning 6 + 1 rowhammer sequences will be tested on a limited set of victim cell groups with the 8x8 data patterns.

Algorithm 6 Rowhammer sequence with data flip

```
while HammerCount > 0 do
    activate(Row)
    read(Col)
    write(Col, DataPattern)                ▷ This writes to the column
    precharge(Row)
    DataPattern ← ¬DataPattern             ▷ This flips the data
    activate(DummyRow)
    read(DummyCol)
    write(DummyCol, 0)
    precharge(DummyRow)
    HammerCount ← HammerCount − 1
end while
```

4.3 Metrics

The metrics that will be used as measure are bit-error-rate and hammercount. these are used to evaluate the results from the experiments. This helps understand if there is a positive impact when implementing the Row Flipper protection method or if it is insignificant compared to the baseline results.

4.3.1 Bit-error-rate

The bit error rate determines the number of flipped bits. For instance during reverse engineering the bit error rate per row will help find the true and anti-cell locations in the memory. When searching for the row neighbours the bit error rate will be high in the rows adjacent to the attacking row. To calculate the bit error rate the memory data after an experiment is compared to the initially inserted data pattern. Then the flipped bits are located and counted. The bit-error-rate can be calculated for rows,columns, banks, arrays, etc.

4.3.2 Hammercount

The hammercount is a measure that identifies the rowhammer resilience of a memory cell. When a cell has a high hammercount it is considered to be resilient against rowhammering. Of course this will also depend on the data pattern of the victim and attacker, though by running a rowhammer attack using ideal data patterns the minimum ham-

mercount will be the determining factor of the victim cell resilience. The way this is measured is by running a rowhammer attack, the hammercount is predetermined in the experiment and after reading the memory, if the victim cell has been flipped it could be concluded that for the hammercount the victim cell will flip. The minimum hammercount is the most important metric, this will define the worst case scenario. Experiment set 2 shows an algorithm that searches for the minimum hammercount by using a binary search method.

4.4 Experimental Infrastructure Design

Infrastructure will be composed from hardware and software components. The hardware being an FPGA development board and the software being instructions assembled from experiment algorithms. Data will be evaluated by a PC that is connected to the FPGA development board via Ethernet.

First the design requirements of the infrastructure are discussed. This gives a clear view of the necessary capabilities the setup must have. Following this the three components of the experimental setup is discussed.

4.4.1 Infrastructure Design Requirements

The experiments require certain capabilities. These requirements are leading in the design of the experimental infrastructure. The main requirements are listed below:

- Interface with SDRAM (e.g DDR3)
- Data transfer to and from a PC
- The execution of DDR commands in a tight sequence

DDR SDRAM commonly is used on user devices such as PC's, laptops and mobile phones. The first paper [1] discussing the flipping of bits in memory used a DDR3 SDRAM module. Later research also presented rowhammer being performed on DDR4.

The goal will be to repeat experiments with changing variables to understand the effects of rowhammer. This will result in a large amount of data that needs to be processed. A PC is responsible for this data processing as a PC can store large amounts of data and process this data easily by means of software.

If a simple experiment board such as a raspberry pi or even a light PC is used one will not have the full control over the exact DRAM commands being executed. The operating system might intervene, the processor can only perform read and writes to specific memory locations and one does not have control over the exact timing of the memory commands. The experimental setup thus requires a very specific type of memory controller where the user can send a sequence of atomic DDR commands and have the confidence that these are executed without any unnecessary delays in between.

4.4.2 Specialized Memory Controller

A specialized memory controller must be developed to execute exact DRAM commands. Here the experimental infrastructure will consist out of three parts. The PC, an FPGA development board and of course the DRAM module. The development board will be designed to operate as a memory controller. The PC will have the responsibility of sending instructions to the FPGA development board. The FPGA board will in turn execute the instructions and send the results to the PC. Figure 4.3 shows the interactions between the three main components. The PC interacts with the DDR3 components via the FPGA.

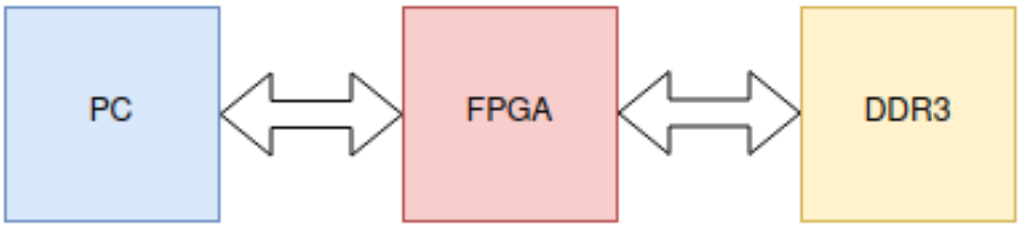


Figure 4.3: Experiment environment with the FPGA acting as memory controller

4.4.2.1 FPGA design

The FPGA will be responsible for the following, memory data transfer to the PC, execution of instructions provided by the PC, interfacing with the DDR3 components. Seen in Figure 4.4 are the FPGA components and their connections with one another.

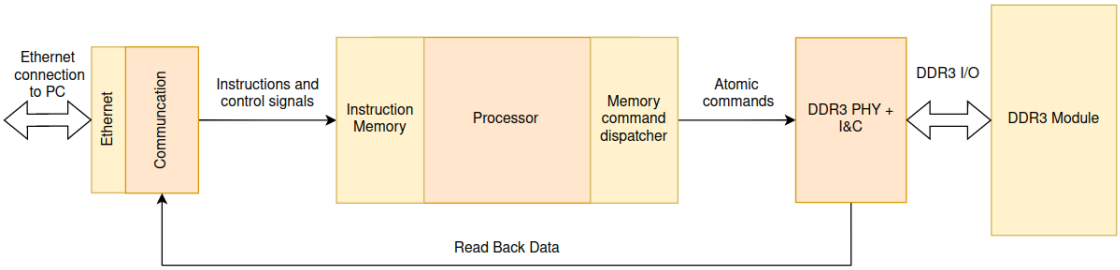


Figure 4.4: FPGA components

From left to right, the communication component interacts with the Ethernet PHY and has a very basic communication protocol to send and receive commands from the PC. The component receives memory data from the DDR3 PHY and sends it to the PC. The instruction memory is loaded directly from the communication component and the processor will receive a reset signal. The processor is responsible for fetching and executing instructions and loading atomic DDR3 commands into the dispatcher. The processor does this by fetching instructions from the instruction memory, executing the instructions and in the case of memory specific DDR3 atomic instructions, it provides the required data in the proper format for the dispatcher. The dispatcher receives these atomic commands with the required data such as, opcode, row address, column address and data to write. After this the DDR3 command is released in a timely fashion. All DDR3 memory commands have a certain time requirement before a next DDR3 command can be issues. These timing requirements are dependant on the type of DDR3 module and the operating clock frequency of the module. The DDR3 PHY works in tandem with the DDR3 module and is aware of the timing delays required for data transfer between itself and the DDR3 module. During startup the DDR3 PHY starts with memory initialization and calibration.

4.4.2.2 PC Design

The PC will assemble the experiments and send the instructions to the FPGA development board. Once the FPGA development board is finished executing the experiments the data is gathered and analysed by the PC. Experiments are run multiple times with varying parameters (such as varying the data pattern for an experiment). That is why the PC has the ability to adjust the experiment, reassemble it into instructions and to send the new instructions to the FPGA development board. Figure 4.5 shows the components that will be implemented on the PC. Here the experiment list provides the

specific experiments in a template format. This allows certain parameters to be adjusted easily when rerunning the experiment. The assembler takes the experiment and assembles it into machine code for the FPGA development board. The communication protocol sends and receives data to and from the development board. The PC will send machine code and receive raw memory data from the development board. This raw data is the actual memory content of the DRAM module. This data is analysed and depending on certain conditions the parameters in the experiment will be adjusted accordingly. The data is also logged and will be used later on to create statistics.

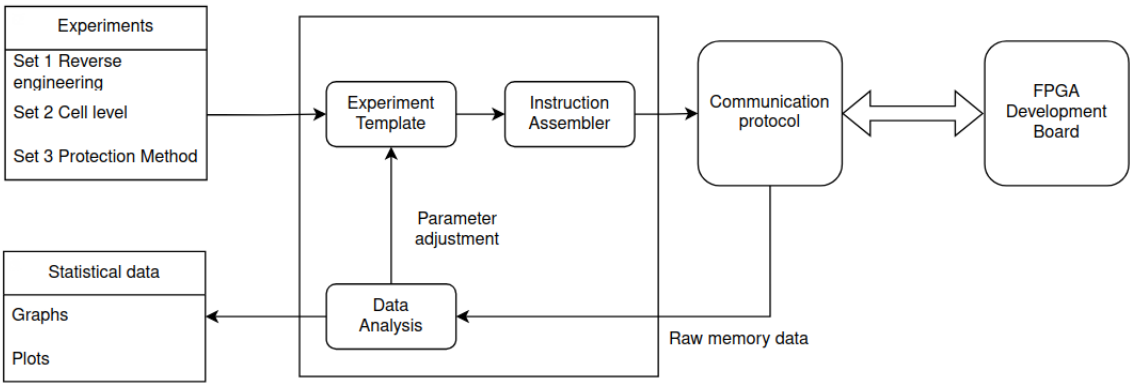


Figure 4.5: PC components

Chapter 5

Infrastructure Implementation & Experimental results

This chapter will discuss the experimental infrastructure implementation and experimental results. The experimental infrastructure is implemented following the design requirements. An FPGA development board with DDR3 modules has been chosen to implement a specialized memory controller. The results of each experiment is discussed in the other half of this chapter. Here the specifics of executing the experiments will be discussed in depth. The results will be shown in graphs and figures and an explanation is given to clarify the findings.

5.1 Infrastructure Implementation

An FPGA development board is the preferred solution for creating a specialized DDR memory controller. Supplemental components are implemented to accommodate the execution of experiments and to ensure ease of use. The chosen development board is the Digilent Genesys 2 [20] as seen in Figure 5.1.

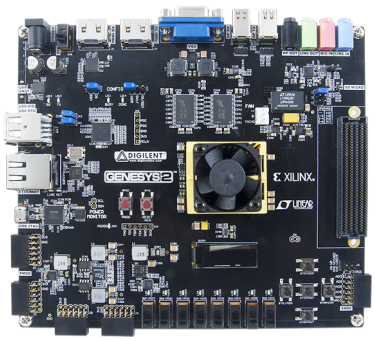


Figure 5.1: Genesys 2 Kintex-7 FPGA development board[20]

Listed below are the most important features of the Genesys 2 development board. These are:

- 1 GiB of DDR3 memory composed from 2x512 MiB chips.
- 10/100/1000 Ethernet PHY
- Xilinx Kintex-7™ FPGA (XC7K325T-2FFG900C)

The Genesys 2 includes 2 Micron MT41J256M16HA-107 DDR3 memory components. One of these components will be used as the main test subject during experimentation.

5.1.1 FPGA Implementation

The FPGA component implementation is discussed in this section. First the the specifications of the DDR3 module are explained, following this the DDR3 PHY implementation will be discussed, after this the fucntion of the DDR3 command dispatcher will be shown in depth, then the processor implementation is given and finally the communication module together with the communication protocol is presented.

5.1.1.1 DDR3 Module

The DDR3 module is the main test subject. The memory on the Genesys 2 development board is composed of 2 Micron MT41J256M16HA-107 DDR3 [21] modules. The functional diagram of the DDR3 DRAM module is shown in Figure 5.2. Here one can see that the device consists of 8 memory banks containing exactly 64 MiB each. The data input/output width of a single module is 16 bits.

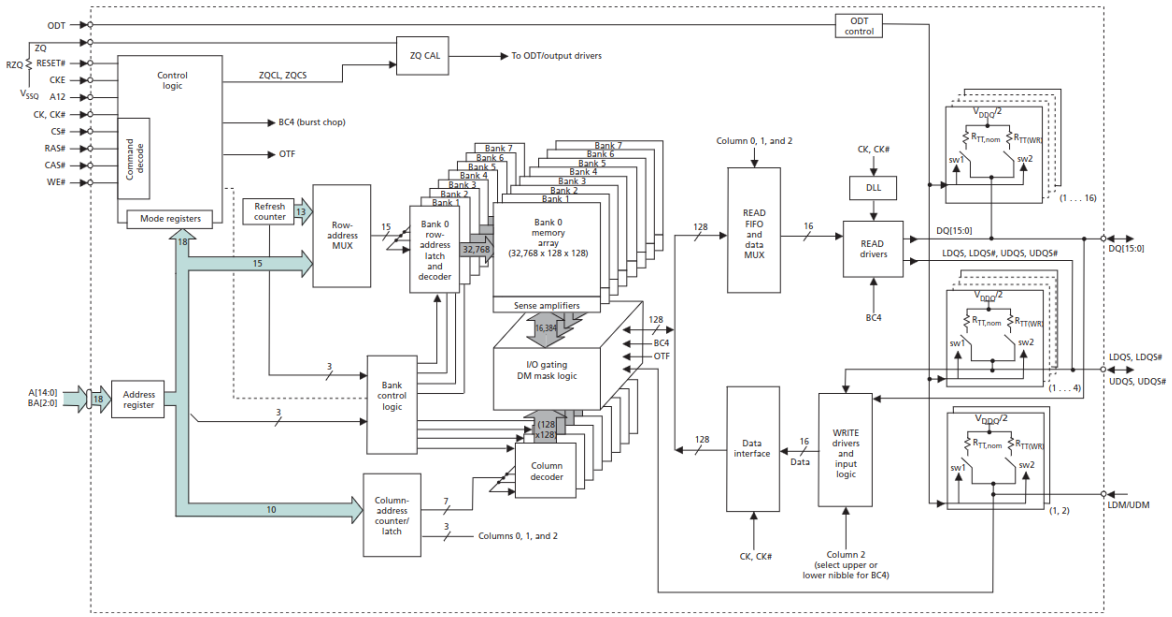


Figure 5.2: Functional diagram of the DDR3 modules [21]

Input and Output The DDR3 PHY is connected to the DDR3 modules. During experimentation not all input- and outputs of the DDR3 chip need to dealt with directly. The user is mostly concerned with the bank, row, column and data. Other input- and outputs such as the clock, DQ, LDQS, UDQS, LDM/UDM and ODT are all handled by the DDR3 PHY with its initialization and calibration functionality. Table 5.1 lists the input- and outputs of a single DDR3 module and describes their basic functionality.

Table 5.1: Inputs and outputs of DDR3 chip

Symbol	Name	Description
Reset#	Reset	This resets the DDR3 chip
CK,CK#	Clock	This is the differential clock input. All input signals are sampled on the positive edge of CK and the negative edge of CK#
CKE	Clock Enable	Enables/disables the clock signal. Used during configuration and power down.
CS#	Chip select	This enables all input signals to the chip. When disabled the input signals are ignored.
RAS# CAS# WE#	Row address strobe Column address strobe Write Enable	These three inputs together with CS# determine the DDR3 command.
A[15:0]	Address input	Here the row or column address are provided to the DDR3 chip as input in combination with the CS#,RAS#,CAS# and WE#.
BA[2:0]	Bank address input	This is the bank address input.
DQ[15:0]	Data input/output	This is the data connection to the DDR3 chip. This is a bi-directional connection to the DDR3 PHY.
LDQS, LDQS# UDQS, UDQS#	Upper/lower byte data strobe	During a read operation the data strobe is set as an output and during a write operation set as an input. The data strobe is aligned with the data on the DQ I/O.
LDM, UDM	Lower/upper byte input mask	Is used to mask part of the I/O data on the DQ.
ODT	On-die termination	Input used for termination resistance to the DDR3 SDRAM

Commands The basics of DRAM have been explained briefly in chapter 2. The most important DDR3 commands have been listed below in Table 5.2. These will be the most used DDR3 commands during experimentation. There are other commands that are

used for initialization and calibration though these will be issued by the DDR3 PHY after startup and the user will not have to concern themselves with these directly.

Table 5.2: DDR3 commands

Fuction	Control input				Requires
	CS#	RAS#	CAS#	WE#	
Refresh	L	L	L	H	Valid BA and A
Activate	L	L	H	H	Valid BA and row address on A
Precharge	L	L	H	L	Valid BA and A
Read	L	H	L	H	Valid BA and column address on A
Write	L	H	L	L	Valid BA and column address on A
No Operation	X	H	H	H	

All the commands, except the *No Operation* command must have a valid signal on the *BA* and *A* inputs. When performing an *Activate* or *Precharge* command for example the row address needs to be provided along with the bank address. The two other commands that perform an operation on the column enable the *CAS#* input and require the column address on the *A* input.

5.1.1.2 DDR3 PHY

The DDR3 PHY is the component that sends the command signals to the chip and ensures timely data transfer. Xilinx provides memory interface generation support known as MIG [22]. MIG generates specific memory interfaces depending on the type of DRAM. This interface provides simplicity for other components to access the DDR3 module and limits them to the use of read and write interactions. The memory controller within the interface generates the sequence of DDR3 commands for the DDR3 PHY shielding other components from certain complexities. MIG is used to generate an interface specifically for the DDR3 modules on the Genesys 2. After this the DDR3 memory controller within the interface has been removed leaving the PHY together with initialization and cali-

bration components. As seen in Figure 5.3 the components created by MIG. Encircled in red is the memory controller that has been removed from the interface leaving only the necessary components.

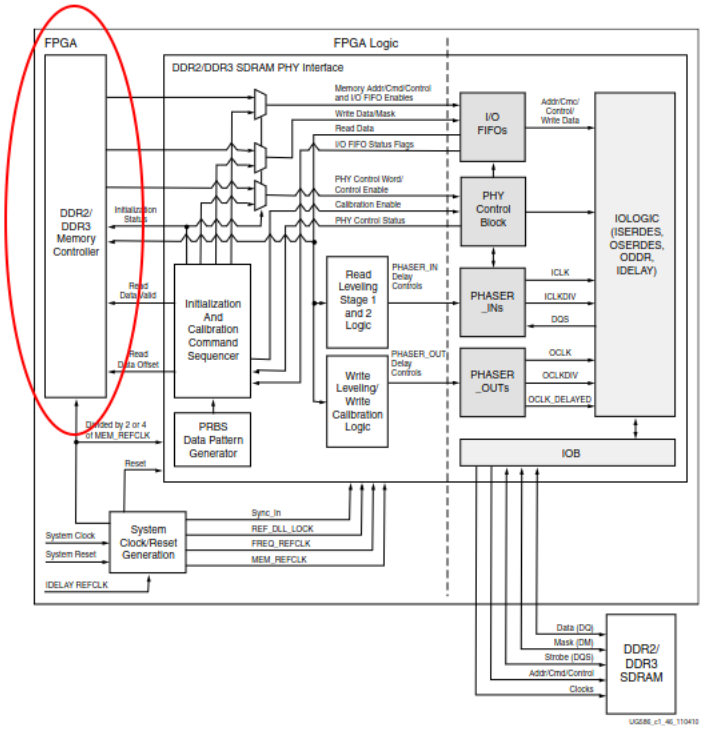


Figure 5.3: Memory interface with Memory controller, PHY and other components [22]

The guide [23] provides the required information needed to operate the PHY directly. It explains that the PHY has two implementations options, the 4:1 and 2:1 implementations. These allow the PHY to operate on a different timescale compared to the DDR3 memory. The 4:1 implementation operates with a clock period four times greater than the DDR3 memory and the 2:1 design twice as great. The 4:1 will have 4 slots to issue a DDR3 command whilst the 2:1 only 2. The 4:1 design is chosen for the FPGA implementation. This allows for the processor and dispatcher to operate on a lower frequency compared to the 2:1 design for a fixed DDR3 clock rate.

To explain the functionality of the time slots an example is provided. It is known that the command signals *CS#, RAS#, CAS#, WE#* together with the *A[15:0] and BA[2:0]* inputs determine the DDR3 commands. The PHY has 4 slots for each of these

inputs, *CS#[3:0], RAS#[3:0], CAS#[3:0], WE#[3:0], A[63:0] and BA[11:0]* (width divisible by 4). For the sake of simplicity these command signals will be simplified as a single input command *CMD[3:0]*. Figure 5.4 shows an example on how to use the slots to issue commands. When issuing an activate (ACT) command on slot 1 the DDR3 module will receive the command after 4 + 1 CK. Following this the PHY receives two consecutive *READ* commands both issued on slot 3. These will be issued after 4 + 3 CK. The read commands are 4 CK apart. Finally, the precharge (PRE) command is issued on slot 1 and is issued 4+1 CK to the DDR3 module after being received by the PHY. So by using the command slots in the PHY the DDR3 module receives commands within a resolution of 1 CK

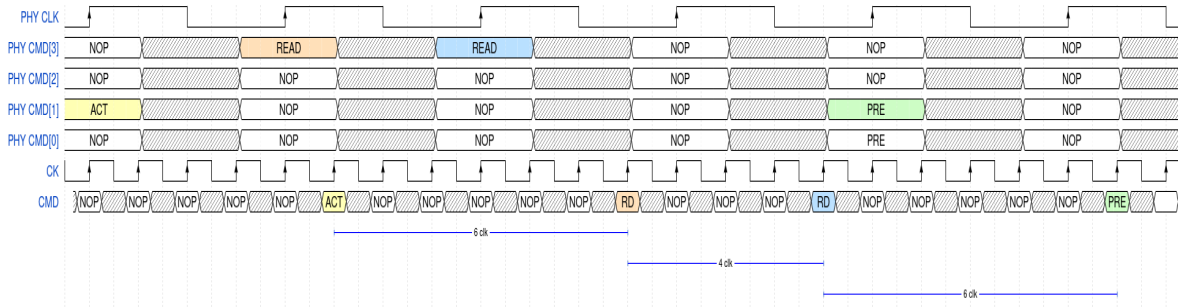


Figure 5.4: DDR3 PHY slot timing

5.1.2 DDR3 Command Dispatcher

The dispatcher uses the PHY command slots to issue DDR3 commands in a timely fashion. This component is aware of the timing requirements for each command. These requirements are listed in the memory datasheet [21]. The 4:1 PHY implementation has been chosen meaning the dispatcher will have 4 slots to issue the DDR3 commands. The dispatcher operates on the same clock frequency as the PHY and thus can issue a command every 4 memory clock cycles. The dispatching of a command is shown in Figure 5.5. Starting from idle, the dispatcher issues the command if the counter *CC_counter* is below 4 and a command is available. After issuing the command the counter *CC_counter* is set to the specific clock delay *Command CC Delay* for the DDR3 module to execute the command properly. When there is no command available or the counter's value is too high, the dispatcher will issue a NOP to the PHY and the counter is lowered by 4. The counter cannot reach a value below 0 and is used to determines the PHY slot on which the new command should be issued.

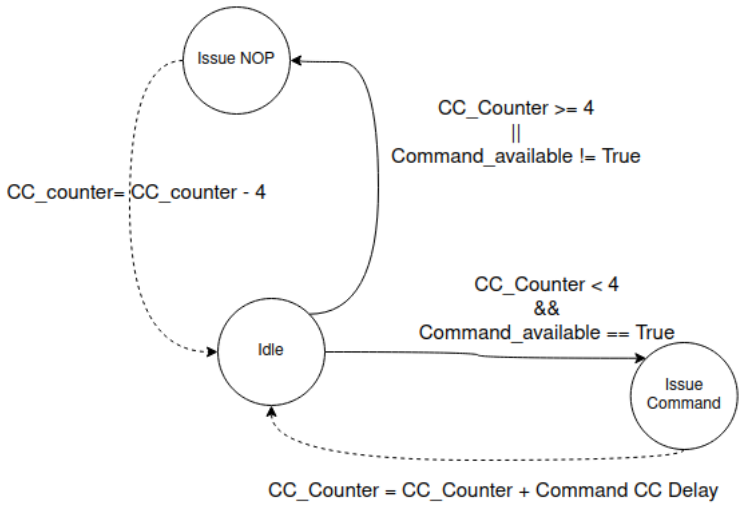


Figure 5.5: State machine of DDR3 command dispatcher

The dispatcher is connected to the processor via an instruction FIFO. The processor inserts the atomic DDR3 instructions into the FIFO, and the dispatcher takes these instruction from the FIFO. Table 5.3 shows the format of the instructions. The dispatcher determines the type of instruction by its opcode. To keep things simple, the instruction is split into its segments and each segment is inserted into the PHY as a separate input signal, while taking the slot into account. These DDR3 instructions are 38 bits wide and composed of the DDR3 command opcode, bank address, row/column address and the data width equal in size to the DDR3 module data width.

Table 5.3: DDR3 atomic instruction format

Segment:	Opcode [37:34]	Bank Address [33:31]	Address [30:15]	Data [15:0]
Composition:	CS# RAS# CAS# WE#	BA	A	Reg Data

5.1.3 Processor

The processor provides functionality to run complex experiments on the FPGA. Without the processor each experiment would have to be assembled into a long list of DDR3 commands and sent to the FPGA development board. This would not be practical as the FPGA has limited storage capacity. Aside from this many of the listed experiments contain loops and conditions in their algorithms. This means that certain parts of the experiment code will be run numerous times. That is why a processor is an essential component. The processor will execute processor specific instructions and pre-process the DDR3 commands to eventually insert them into the FIFO connected to the dispatcher.

The processor instruction set is based the Little Computer 3 (LC-3) instruction set. The reason for this is because this instruction set is fairly barebones but powerful. This will keep processor design time within bounds. The aim is to run the processor on a higher clock frequency compared to the DDR3 command dispatcher and a complex processor design might not allow for this.

Only the essentials of the LC-3 instruction set will be implemented. These are *add*, *and*, *not*, *br* (branch). Together with these one extra arithmetic instruction is added, namely *sll* (shift logic left). Without this, shifting will require adding a register to itself numerous times as $reg * 2 == reg + reg$. The instruction set for the processor has been implemented while respecting the LC-3 instruction format. There are two types of instructions, The processor instructions and the DDR3 atomic command instructions. These instructions are listed in Table 5.4. The width of the processor instructions is 30 bits. The standard LC-3 4 bit wide opcode has been extend by 1 bit. This extra bit distinguishes the processor specific instructions from the instructions used for the DDR3 command dispatcher. The 3 bit *DR and SR* fields allow for 8 data registers. The *NZP* field allows for branching. *N* bit is set for branching on a negative value,*Z* is set for zero and *P* for positive. The 1 bit *Set imm* field is used to indicate an immediate instruction, the difference will be that the processor takes the value in the immediate field rather than from *SR2*. An extra instruction, not previously mentioned, has been added. This is the *Sbr* standing for set bank register. This instruction will load a 3 bit value into a specific register used by the processor for assembling DDR3 dispatcher instructions. The DDR3 specific instructions are similar in format to the processor specific instruction format. This allows for the same data path to be taken in the processor. The *DR/NZP* field is not used for the DDR3 instructions. The *SR* field is used for selecting the register holding the address (row for activate/precharge and column for write/read) value. The

Write instruction uses the *SR2* field to point to the register holding the data to be written to the memory location. The immediate field in *NOP* is sent to the dispatcher to issue the immediate number of *NOP* instructions and *REF* is used to set the refresh period.

Table 5.4: Processor instruction set

Segment:	OPCODE 5-bit	DR/NZP 3-bit	SR 3-bit	Set imm 1-bit	SR2/IMM 8-bit
Processor specific instructions					
Add	0 0 0 0 1	DR	SR1	0	0 0 SR2 0 0 0
Add imm	0 0 0 0 1	DR	SR1	1	IMM 8-bit
And	0 0 1 0 1	DR	SR1	0	0 0 SR2 0 0 0
And imm	0 0 1 0 1	DR	SR1	1	IMM 8-bit
Not	0 1 0 0 1	DR	SR1	1	1 1 1 1 1 1 1 1
Sll	0 0 0 1 0	DR	SR1	0	0 0 SR2 0 0 0
Sll imm	0 0 0 1 0	DR	SR1	1	IMM 8-bit
Br	0 0 0 0 0	NZP	Offset 12-bit		
Sbr	0 1 1 1 1	000	000	1	0 0 0 0 0 Imm 3-bit
DDR3 specific dispatcher instruction					
ACT	1 0 0 1 1	000	SR1	0	0 0 0 0 0 0 0 0
PRE	1 0 0 1 0	000	SR1	0	0 0 0 0 0 0 0 0
WRITE	1 0 1 0 0	000	SR1	0	0 0 SR2 0 0 0
READ	1 0 1 0 1	000	SR1	0	0 0 0 0 0 0 0 0
NOP	1 0 0 1 0	000	000	1	IMM 8-bit
REF	1 0 0 0 1	000	000	1	IMM 8-bit

The processor assembles the DDR3 dispatcher commands from the instruction opcode and register values containing bank address, row/column address and data to be written. The format shown earlier in Table 5.3. As mentioned before the dispatcher receives instructions from the processor via a FIFO. This is shown in Figure 5.6. Here it is seen that the processor writes to the FIFO and in the case that the FIFO is full the processor is halted. The DDR3 command dispatcher reads from the FIFO if it is not empty. The FIFO allows for the two components to operate at different clock frequencies, allowing the processor to operate at a higher clock frequency ensuring that the DDR3 command dispatcher will have DDR3 instructions available at all times during the execution of an experiment.

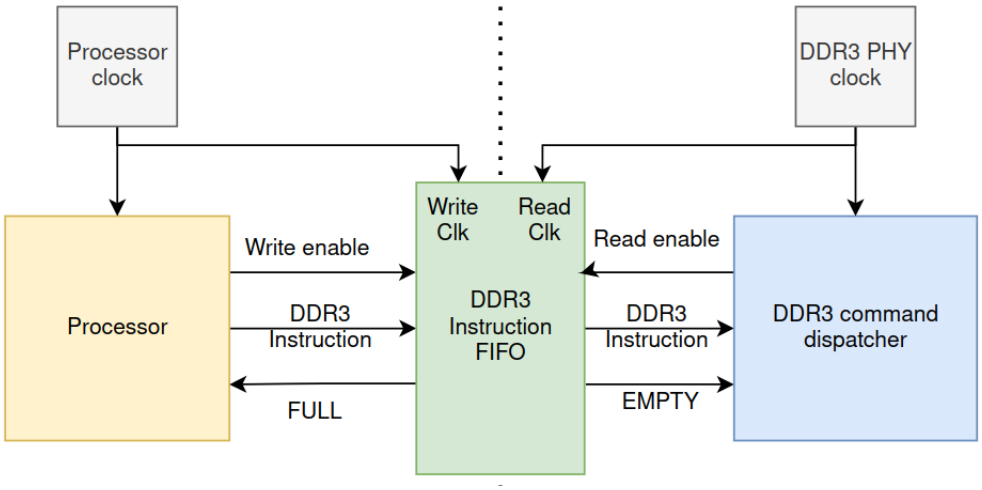


Figure 5.6: FIFO in between processor and dispatcher

5.1.4 Communication and Control

This component is responsible for receiving, transferring and loading instructions into the processor instruction memory. To start this component is connected to an ethernet PHY. The Genesys 2 features a RGMII interface. The communication component receives ethernet packages and deconstructs these into destination, source, and payload segments. These segments are forwarded to the communication handler. The handler functionality is shown in Figure 5.7. The system starts in the idle state and sends a broadcast message every second. This broadcast message is for the PC to see that the FPGA is online. The PC sends a message to the FPGA and the communication handler will check the payload and determine the validity. In case the package content is valid an ACK is sent and if invalid a NACK is sent. Once a connection has been established memory data can be sent to the PC. When the DDR3 modules memory is being read the DDR3 PHY will fill the data buffer. Once this data buffer has reached a certain threshold the data form the memory will be packaged and sent to the PC.

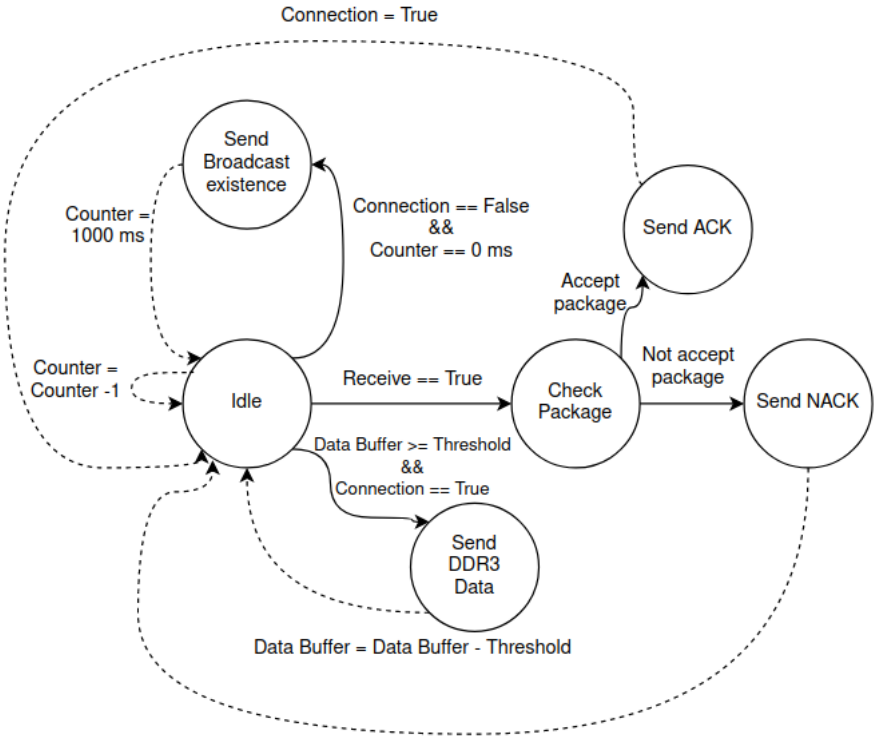


Figure 5.7: Communication handler

The PC and FPGA communication packages are listed in Table 5.5. The PC initiates the loading of an experiment by sending a *Start program load* package to the FPGA. The FPGA ensures the processor is halted and that the instruction memory is loaded with the data from the *Program Instruction* data segment. Once the FPGA receives a *End program load* instruction the processor is reset starts executing the instructions.

An Ethernet frame ends with a frame check sequence. This check uses a 32 bit CRC. If at any point during communication an Ethernet package becomes corrupt the Ethernet controller will throw the package away due to failing the frame check. The PC knows the number of DDR3 read commands in the experiment instructions. The PC will check the number of bytes in the received memory data packages to ensure no Ethernet frame loss has occurred during operation.

Table 5.5: PC and FPGA communication packages

Command:	start 1-byte	Command 1-byte	Data size (N) 1-byte	Data N-bytes
PC packages				
Start program load	0x01	0x02	0x00	-
End program load	0x01	0x03	0x00	-
Program Instruction	0x01	0x04	0x03	20-Bit instruction in 3-Bytes
FPGA packages				
ACK	0x01	0x06	0x00	-
NACK	0x01	0x15	0x00	-
Memory Data	0x01	0x05	#Bytes in buffer	Memory Data

5.1.5 Experiment Infrastructure Summary

Summarizing, the experiment infrastructure is composed out of a PC and a Genesys 2 FPGA development board with DDR3 modules. The experimental infrastructure interfaces with one DDR3 module. The experiments are assembled on the PC. The PC communicates with the Genesys 2 and transfers the experiment instruction code to the FPGA via an Ethernet connection. The FPGA handles the incoming data packets and inserts the experiment instructions into the processor instruction memory. The experiment code is composed of processor specific instructions and DDR3 instructions. These DDR3 instructions are assembled by the processor and are inserted in a FIFO for the DDR3 command dispatcher. The dispatcher takes commands from the FIFO and dispatches these in a timely fashion to the DDR3 PHY. The PHY interacts with the memory on a physical level and initializes the memory module and calibrates itself during startup. The data read from the DDR3 modules is sent to the PC.

The PC specifications are listed below:

- PC with Ethernet connection running Ubuntu 20.04
- Communication and assembly programmed in: C and Python

The FPGA specifications:

- Processor operation frequency: 200 Mhz.
- DDR3 PHY operation frequency: 100 Mhz.
- Communication operation frequency: 125 Mhz.

The DDR3 module specifications:

- Operating frequency: 400 Mhz.
- Data transfer speed: 800 MB/s.

5.2 Experiment Results

This section discusses the results from the experiments. First the results of experiment set 1 are discussed, following this are the results from experiment set 2 and finally the results are shown from experiment set 3.

5.2.1 Experiment set 1: Reverse engineering

These results will provide information on the specific DDR3 module test subject. The results found in this section are essential to the other two experiment sets. Here the true- and anti-cell locations are described, the neighbouring rows are given, the size of the rows are verified, the column structure is discovered and finally a set of vulnerable cells is composed.

5.2.1.1 True- and anti-cell locations

The true- and anti-cell locations have been found. This is done by first loading a solid data pattern into the whole memory bank, after this the memory is left idle for 5 minutes with self refresh disabled. During this time the charged memory cells lose their charge. Once the 5 minutes have expired the whole memory bank is read out and compared to the initial data pattern.

The solid 1 data pattern reveals the true-cells in the memory bank. As seen in Figure 5.8 the bit errors are shown per row for the solid 1 data pattern. The x-axis shows the row index number. This is essentially the row address. This figure shows that in certain rows a high number of bit flips occur when a solid 1 data pattern is loaded while

others are unaffected. This is because the solid 1 data pattern charges the true-cells. During the time period these cells lose their charge and thus their bit value flips.

Figure 5.8: True-cell locations with solid 1 data pattern

The solid 0 data pattern reveals the anti-cells in the memory bank. This is the opposite of what has been shown in Figure 5.8. The bit flips that occur while using solid 0 are shown in Figure 5.9. These rows showed no bit flips with a solid 1 data pattern. But when a solid 0 data pattern is loaded bit flips do occur. The solid 0 data pattern charges the anti-cells. These cells lose their charge over time.

Figure 5.9: Anti-cell locations with solid 0 data pattern

Solid 0 has a comparable error rate to solid 1. The two data pattern errors have been combined in a single plot. Figure 5.10 shows the two data pattern errors per row for the given retention time. It is visible that there is an alternating pattern between the error locations. Where the solid 1 data pattern shows errors the solid 0 data pattern shows none, and vice versa.

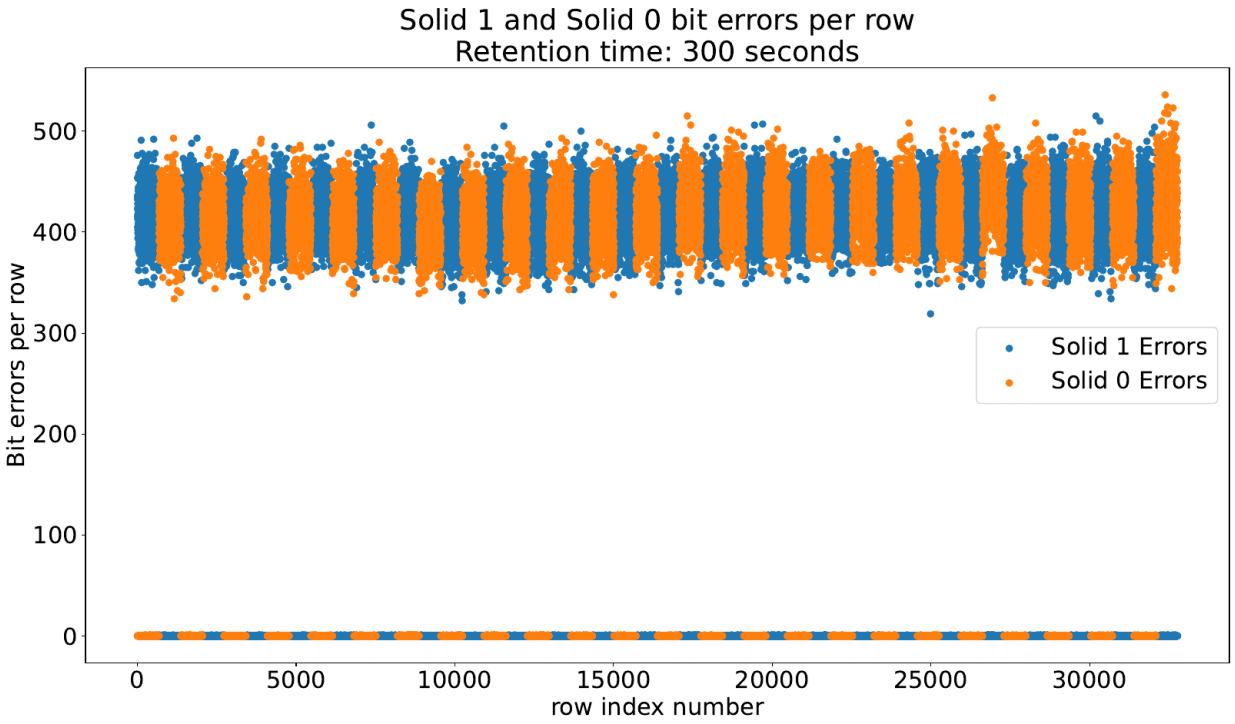


Figure 5.10: Solid 1 and solid 0 data pattern errors per row

The total number of errors per data pattern is listed in Table 5.6. The bit error rate of the two data patterns are compared to one another. Both data patterns seem to cause roughly the same number of bit flips per row. There is no significant difference between the true- and anti-cell error rate for the 300 second retention time.

Table 5.6: Solid 1 compared to Solid 0 total bit errors, average per row and difference

Data pattern	Total number of bit errors	Average bit errors per row	Difference compared to Solid 1
Solid 1	6824529	20.34	-
Solid 0	6869236	20.47	0.66%

The true- and anti-cells rows are grouped. Rows that have true-cells are grouped together whilst rows that have anti-cells are grouped together. The groups have a size

of either 680 or 688 rows. To better explain Table 5.7 shows the first 4 groups. These have been classified as true- and anti-cell groups and their group size is provided.

Table 5.7: True- and anti-cell groups

Row Addresses	True/anti cells	Group size
0-679	True	680
680-1367	Anti	688
1368-2367	True	680
2368-3047	Anti	680

Excluding the first and last group in the memory bank a repetition of these group sizes can be seen. A group size of 688 rows is followed by two groups of 680. With each group alternating between true- and anti-cells. There are a total of 24 true-cell groups as well as 24 anti-cell groups.

5.2.1.2 Size of rows

The row size has been verified by first inserting a columnstripe data pattern into the memory bank, activating a single row, waiting 300 seconds and finally precharging the row. By using the columnstripe data pattern there is no need to keep track of true- and anti-cell location in the memory. Random rows where selected in the memory to verify the row size. One example is shown in Figure 5.11, here the row is activated throughout the retention test. The activated row has 0 bit errors whilst the other non active rows show significant bit errors.

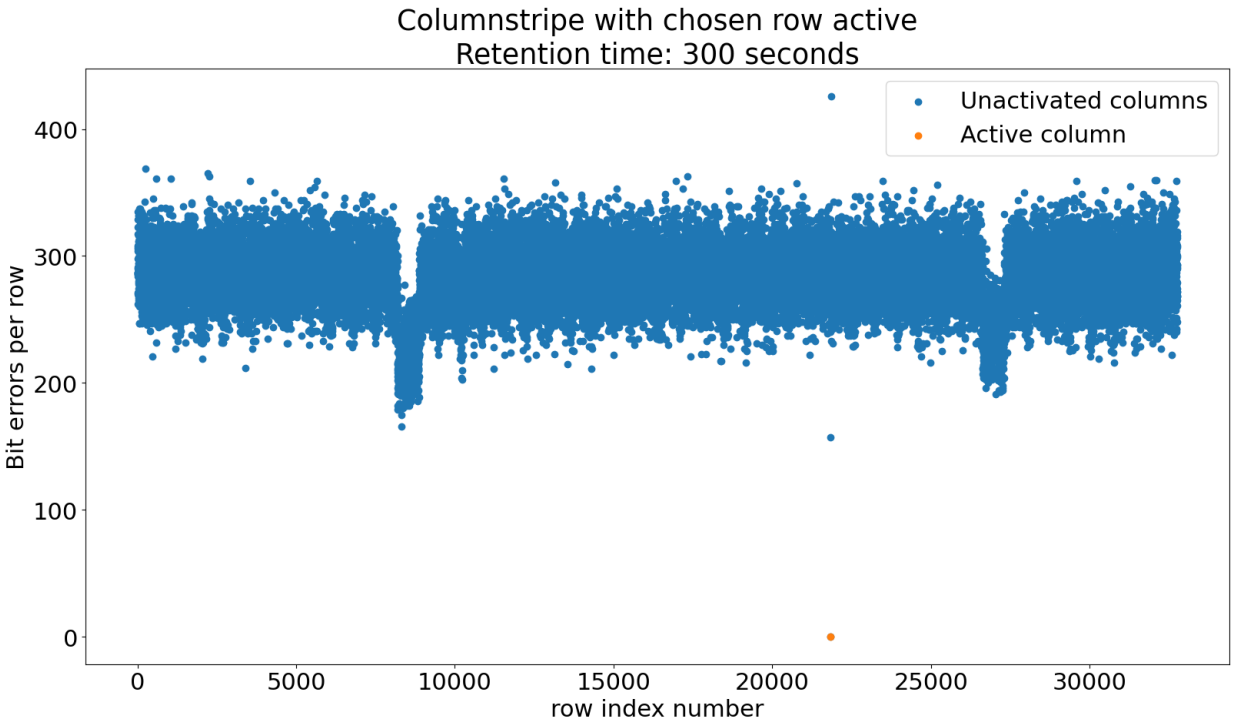


Figure 5.11: Bit errors per row of activated row compared to others

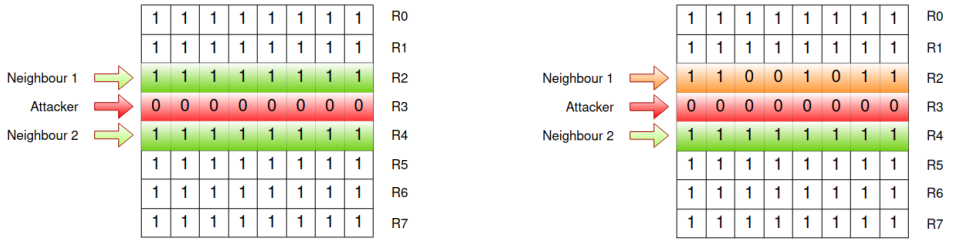
This result shows that during the experiments only one row shows 0 bit errors. Meaning that the row buffer only holds the exact row size, no other rows are hidden in the row buffer.

5.2.1.3 Neighbouring rows

A single sided rowhammer attack is performed to find the neighbouring rows of the attacking row. The data pattern is loaded into the memory (solid 0 or solid 1). After this the attacking row is loaded with the opposite data pattern. This row is accessed consecutively throughout the round together with its opposite row number in the other half of the memory bank. Meaning row 0 together with row $2^{15} - 1 - 1$ are accessed in the first round. Row 1 and row $2^{15} - 1 - 2$ are accessed in the second round, and so forth. The rows with bit errors in them are logged at the end of each round. The experiment has been performed with a hammercount of 3 million hammers per row. Meaning each round the attacking row has been accessed 3 million times. This high hammercount is

chosen to also inflict bit errors in the neighbour's neighbour, if possible. The results are discussed below.

Each row has a single row affected by rowhammer. After each round the memory array is read out and analysed. During the analysis of the memory only one row shows bit flips, meaning that the attacking rows (n or $2^{15}-1-n$) only affect one single victim row. To illustrate this Figure 5.12 shows the expectation based on the understanding of a single sided rowhammer attack and the findings in the results. Figure 5.12a shows the attacking row *R3* with the two neighbouring rows *R2* and *R4* expected to have bit flips. The reality is shown in Figure 5.12b where the attacking row *R3* only seems to flip bits in *R2* whilst keeping *R4* unaffected.



(a) Attacking row and expected affected rows (R2 & R4)

(b) Attacking row (R3), actual affected row (R2) and expected affected row (R4)

Figure 5.12: Neighbour affected by attacker row

All the rows in the memory bank have been an attacker or a victim at some point in the experiment. When the data pattern solid 0 or solid 1 is inserted into the memory bank only one attacker row n or $2^{15}-1-n$ seems to have an impact. This is due to the fact that one of the attacking row addresses can be in a true- or anti-cell row region and depending on cell charge only will affect one row. If a row shows bit flips it is paired together with n or $2^{15}-1-n$ depending on its locality. Table 5.8 shows the first and last 4 attacker rows n impacting only one single neighbour local to them. This neighbour is classified as the victim row and is paired together with the attacker row. It is interesting to see that the attacker and victim roles are swapped at some point.

Table 5.8: Attacker row n with neighbouring rows n-1 and n+1 showing only one row being affected

Attacker row address (n)	Neighbour address (n-1)	Neighbour address (n+1)	Affected Address
0	-	1	1
1	0	2	0
2	1	3	3
3	2	4	2
...	...	...	...
32764	32763	32765	32765
32765	32764	32766	32764
32766	32765	32767	32767
32767	32766	-	32766

Using the results a formula is composed to map the victim row to the attacker row and vice versa. From the results shown above a formula is created in order to simplify future experimentation. The Equation 5.1 shows the relation of the attacker and victim row. The input row number n can be the attacker or victim row and the output of the formula will be its antithesis. Each victim row has one attacker row mapped to it and vice versa. No victim rows share the same attacker row and no attacker rows share the same victim row.

$$V(n) = \begin{cases} n-1, & \text{if } n \mod 2 = 1 \\ n+1 & \text{otherwise} \end{cases} \tag{5.1}$$

5.2.1.4 Neighbouring columns

The column order of two neighbouring rows is investigated. The victim row has been loaded with a fully charged data pattern, *solid 1* or *solid 0* depending on the cell type, while the attacker row has only a single bit charged at a time. The attacking row performs a rowhammer attack with a walking 1 or 0 (depending on cell type) and the impact on the victim row is recorded. Figure 5.13 has the results of 4 attacker and victim rows. The plot shows that if a victim cell is flipped it is caused by an attacker cell in the neighbouring row with the exact same column address. Meaning that the attacker and victim cells are vertical neighbours.

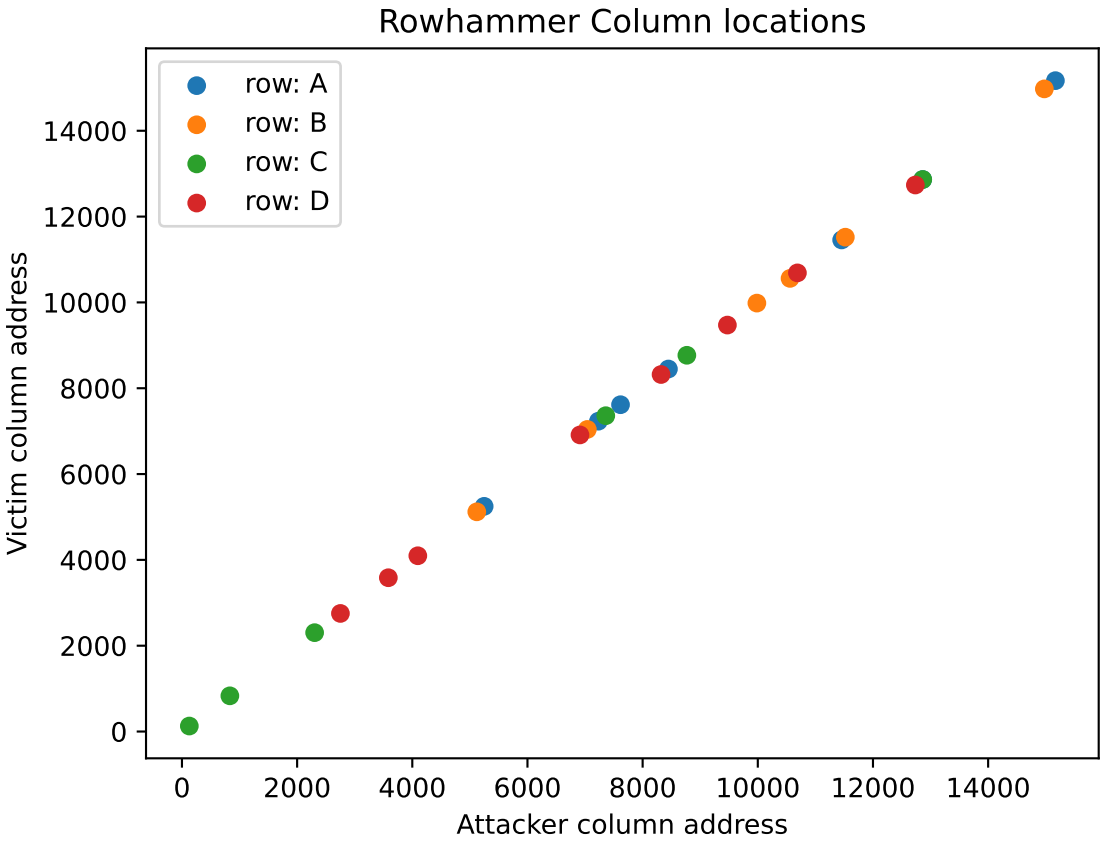


Figure 5.13: Bit flip in Victim column caused by Attacker column

5.2.1.5 Vulnerable cells within refresh period

Every row in the memory bank has been hammered and the vulnerable cell locations have been logged. The hammer sequence is $act \rightarrow pre$. This will give an overestimation as in practice *read* or *write* commands will be issued as well. The duration of a single $act \rightarrow pre$ sequence is 47.91 ns. The memory clock is set to 2.5 ns. This means that a single sequence takes 20 memory clock cycles as seen in Equation 5.2. However the maximum hammercount has been set to 667000 to account for the theoretical maximum hammercount within 64 ms.

$$\left\lceil \frac{47.91}{2.5} \right\rceil = \lceil 19.164 \rceil = 20 \; ck \tag{5.2}$$

The data patterns that are chosen are solid 1, solid 0 and row-stripe (also negated). The victim and attacker rows are loaded with the data pattern before executing the rowhammer attack. After this the victim row is read in its entirety and saved for further processing. Every victim row is attacked with these 4 different data patterns. Solid 1 and solid 0 mean that the attacker and victim have the same cell charges whilst rowstripe (also negated) mean that these two rows have opposing cell charges.

Error rate of Data patterns The bit errors induced by the *Solid 1* and *Solid 0* data patterns are shown in one plot, Figure 5.14. Here the errors per victim row are shown being caused by the rowhammer attack. It is clearly visible that the two data patterns are prone to bit flips in specific row locations. When comparing these results to Figure 5.10 shown previously it is clear that these bit flips occurring are dependant on the true- and anti-cell row locations. Bit flips only occur when the victim cell is charged.

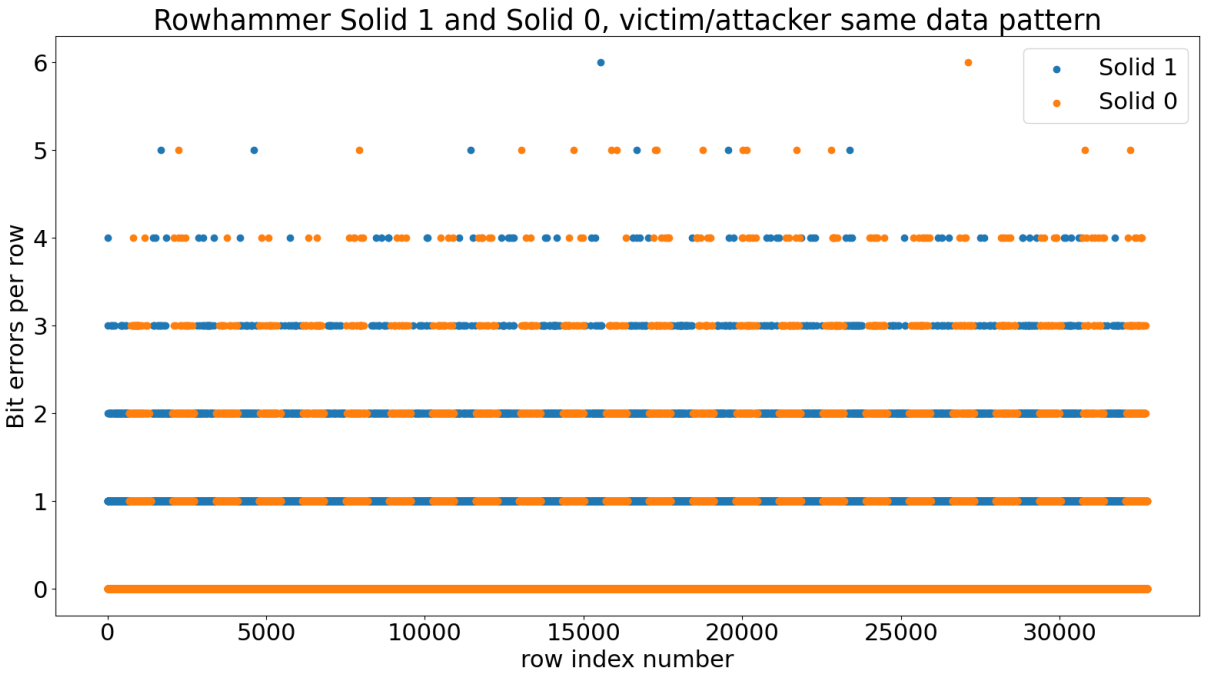


Figure 5.14: Solid 1 and solid 0 errors per row caused by 667k rowhammer

The bit error rate of the two data patterns is compared in Table 5.9. It is clear that the rowhammer attack causes a higher bit error rate in the *Solid 0* data pattern. The result is a 10.49% increase compared to *solid 1*.

Table 5.9: Bit error rate of solid 1 and solid 0.

Data pattern	Total Bit Errors	Bit Error Rate	Difference Solid 1
Solid 1	9973	0.0297%	-
Solid 0	11020	0.0328%	10.49%

The following plot in Figure 5.15 shows the errors per row caused by rowhammer for the *Rowstripe* (victim 1 and attacker 0) and $\overline{Rowstripe}$ (victim 0 and attacker 1) data patterns. Here the victim and attacker have opposing cell charges. It is immediately clear that these data patterns have a significantly higher bit error count per row compared to

the *Solid* data patterns. Once again, bit errors only occur in locations where the certain data pattern has the victim cell initially charged. Meaning that *Solid 1* and *Rowstripe* cause flips in the true-cells during rowhammer and *Solid 0* and $\overline{Rowstripe}$ cause flips in the anti-cells. The plot also shows something not seen previously. There are two regions where the bit errors per row are significantly higher compared to the adjacent rows.

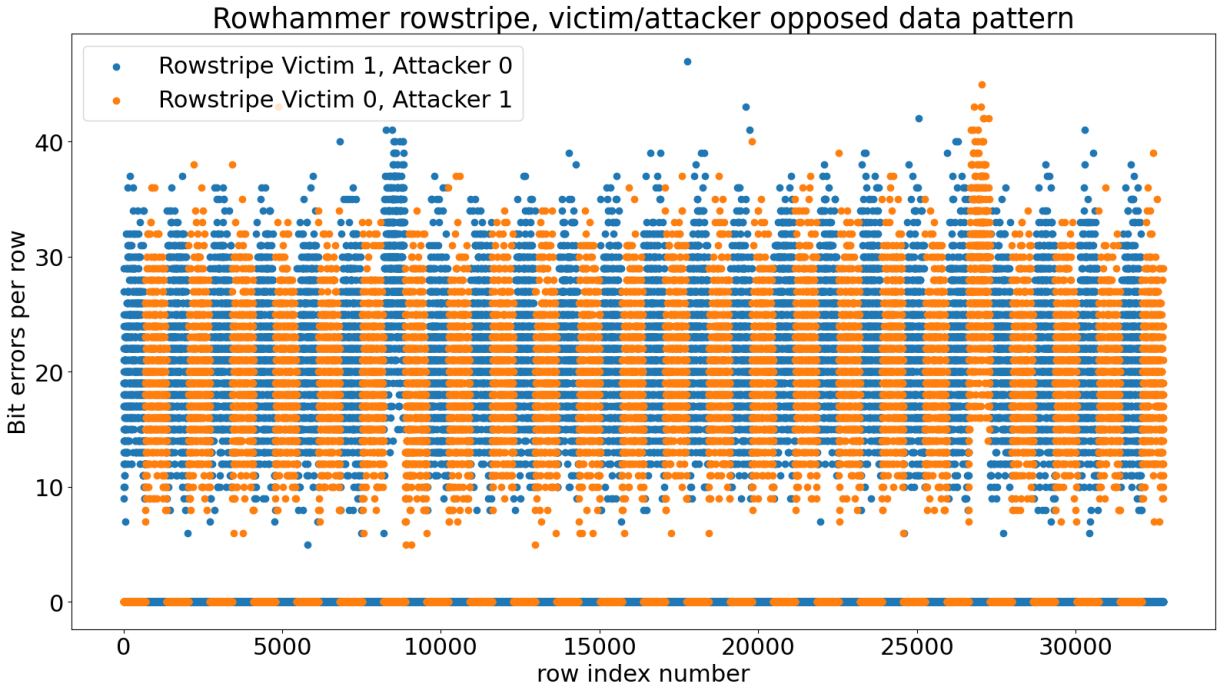


Figure 5.15: Rowstripe and $\overline{Rowstripe}$ errors per row caused by 667k rowhammer

In Table 5.10 the bit error rate of *Rowstripe* and $\overline{Rowstripe}$ are compared to one another. The results show that *Rowstripe* suffers from a higher bit error rate compared to $\overline{Rowstripe}$. The increase is 7.82%. This means that the charged victim true-cells being attacked by uncharged true-cells are prone to a higher failure rate compared to the charged anti-cells being attacked by uncharged anti-cells.

To summarize, victim cells only flip when they are initially charged. The charge of the attacker also is a determining factor in the bit error rate. There is a higher bit error rate for victim cells being attacked by uncharged attacker cells. Anti-cells suffer more

Table 5.10: Bit error rate of Rowstripe and $\overline{Rowstripe}$.

Data pattern	Total Bit Errors	Bit Error Rate	Difference Rowstripe
Rowstripe	354929	1.06%	-
$\overline{Rowstripe}$	327193	0.98%	-7.82%

when attacked by charged anti-cells, whilst true-cells suffer more when being attacked by uncharged true-cells.

5.2.2 Experiment set 2: Cell Level

First, cells have been selected from the vulnerable cell set provided by Section 5.2.1.5. The selection includes true- and anti-cells, cells in rows with a high bit error count and cells in bytes with a high bit error count. Due to the extremely long execution time of the experiments in set 2 and set 3 the vulnerable cell set is limited in size. The selected rows are shown in Table 5.11. A total of 43 victim cells have been selected. Here there are two categories namely True- and anti-cell rows. Two rows per group contain victims cells neighbouring each other horizontally. All the cells have been named and their exact locations can be found in Table A.1 and Table A.2. These cells will be referred to throughout the following sections.

Table 5.11: Selected rows

Victim row nr	Attacker row nr	Nr of selected cells	Maximum bit error per byte	Vulnerable Neighbour cells?
True-cell rows				
15503	15502	6	3	No
16966	16967	5	3	Yes
30546	30547	13	1	No
Anti-cell rows				
13069	13067	11	1	No
13655	13654	3	3	Yes
26939	26938	5	3	no

As mentioned in Section 4.2.2.1 the three cell data patterns vary for the attacker

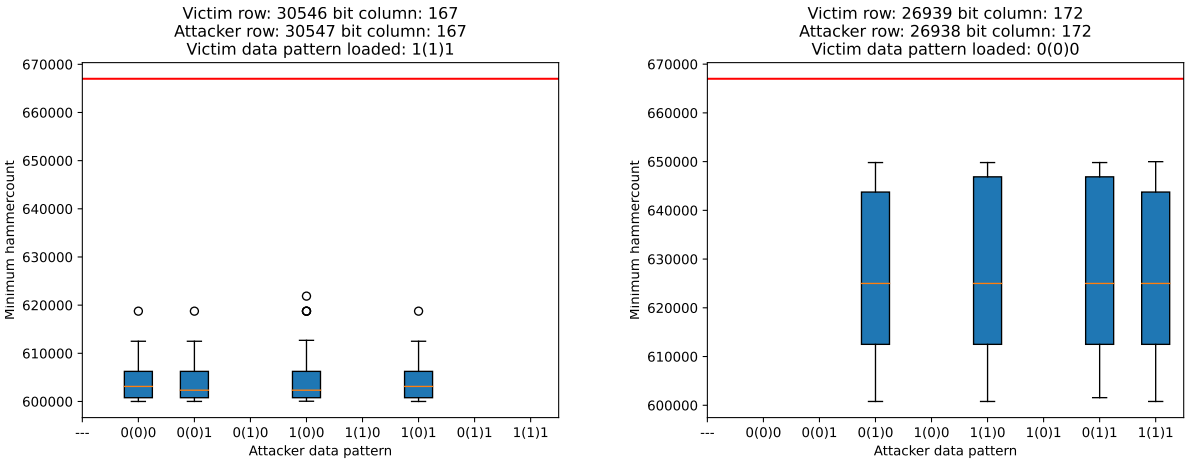
and victim cell groups. The minimal hammercount has been found using binary search. Each victim/attacker data pattern pair has been retested 100 times with the data being logged. The binary search set a limit of 800000 hammers per row as the maximum and 0 hammers as the minimum. The data pattern format will be shown as $\{0(0)0, 0(0)1, 0(1)0, 1(0)0, 1(0)1, 1(1)0, 0(1)1, 1(1)1\}$ throughout this chapter. The 1/0 in between the parentheses indicates the bit value of the cell of most interest (victim cell or attacker cell) whilst the two values to the left and right of the parentheses are the horizontal neighbouring cell values. For example, when loading 0(1)1 into a victim group, the victim itself will be loaded with a value 1, while its left neighbour is loaded with a 0 and its right neighbour with a 1. As a reminder, V-LN and V-RN stand for the victims left- and right-neighbours, these are adjacent to the victim cell in the same row. Subsequently, A-LN and A-RN indicate the attackers neighbours.

5.2.2.1 Impact of Victim and attacker cell charges

All of the victim cells showed bit flips during rowhammer while being charged and attacked by an uncharged attacker cell. To show an example the results of two victim cells have been shown in Figure 5.16. Here Figure 5.16a shows that a true-cell flip occurs when the victim cell is 1(1)1 and the attacker cell is $X(0)X$ (with x being don't care). For the anti-cell the opposite is shown in Figure 5.16b, here the victim is loaded with a 0(0)0 and the attacker with a $X(1)X$. This shows that bit flips occur for charged victim cells being attacked by uncharged attacker cells.

This verifies the point in Section 4.2.2 of victim cells losing their charge during rowhammer. The results showed no evidence of bit flips caused by victim cells being uncharged during rowhammer.

Amongst the set of victim cells one particular cell has been found to flip during a rowhammer attack with a charged attacking cell. In Figure 5.17 the boxplot shows the victim cell hammercount with the attacker cell data patterns. The red line in the figure shows the theoretical maximum hammercount allowed within a 64 ms time period. So all hammercount values below this are considered to be successful. The results show that the attacker cell can be uncharged as well as charged to cause a bit flip.



(a) True-cell with data pattern 1(1)1

(b) Anti-cell with data pattern 0(0)0

Figure 5.16: True- and anti-cell bit flip box-plot with varying attacker cell data patterns

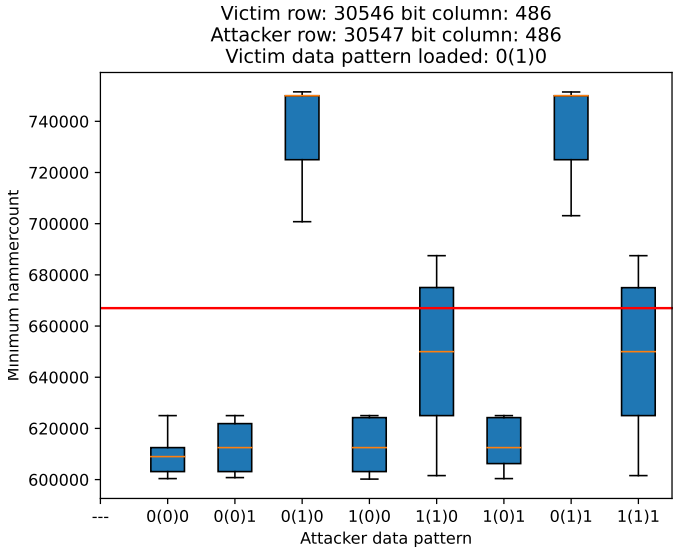
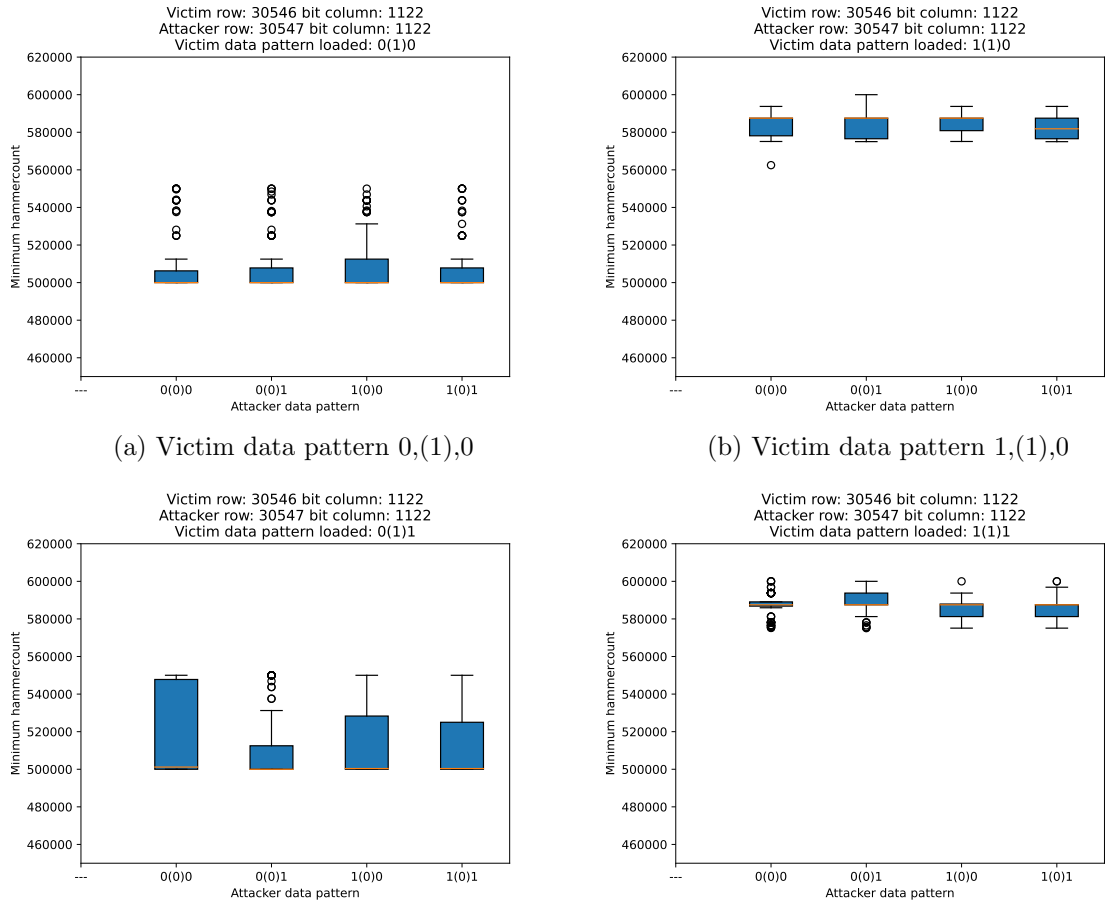


Figure 5.17: Victim cell bit flip with all attacker data patterns

Summarizing, for a bit flip to occur the victim cells need to be charged. While the set of chosen victim cells mostly show bit flips occurring with uncharged attacker cells there is also evidence of an attacker cell that causes a bit flip while being charged.

5.2.2.2 Horizontal cell impact on hammer resilience

The horizontal neighbouring cells (V-LN & V-RN) seem to have an impact on the hammercount of the victim cell. A few examples will be discussed. Starting with Figure 5.18. Here a boxplot of the minimal hammercount is shown of a charged true-cell being hammered by an uncharged attacker cell. The neighbouring attacker cells A-LN and A-RN do not seem to have a large impact on the hammercount. However the neighbouring victim cells do have an impact on resilience of the victim cell. When V-LN is uncharged the hammercount of the victim cell is lower compared to when V-LN is charged. This can be seen in Figure 5.18a where the victim cell pattern is 0(1)0 and in Figure 5.18c where the victim pattern is 0(1)1. Compare this to the two other patterns 1(1)0 and 1(1)1 in Figure 5.18b and Figure 5.18d.



(a) Victim data pattern 0,(1),0

(b) Victim data pattern 1,(1),0

(c) Victim data pattern 0,(1),1

(d) Victim data pattern 1,(1),1

Figure 5.18: Varying victim true-cell data pattern and minimal hammercount.

Statistics in Table 5.12 show the vulnerability of the victim cell in terms of the average minimal hammercount. The table shows the average per victim/attacker combinations and the total average of the victim or attacker data patterns. As stated earlier the V-LN being uncharged shows a decrease in the resilience of the victim cell. To understand the impact of the V-LN the average of the victim patterns with an uncharged V-LN (0(1)0 and 0(1)1) are compared to the data patterns with the V-LN being charged (1(1)0 and 1(1)1). There is a 12.6% decrease in minimum hammercount when the V-LN cell is uncharged compared to the V-LN cell being charged.

Table 5.12: Average minimal hammercount of True-cell with left victim neighbour impact

Victim \ Attacker	0(0)0	0(0)1	1(0)0	1(0)1	Avrg. victim Min. HC
0(1)0	507819.72	509126.38	509044.42	508202.26	508548.20
1(1)0	583824.89	583527.56	584195.52	582427.05	583493.80
0(1)1	516897.11	511015.98	514961.12	514243.58	514279.45
1(1)1	586923.0	588950.14	586158.78	586429.93	587115.46
Avrg. attacker min. HC	548866.18	548155.02	548589.96	547825.71	548359.22

There is a subset of cells in the selected cell set (Table 5.11) that show a significant decline in resilience by one of their horizontal neighbours (V-LN or V-RN). Table 5.13 shows their average minimal hammercount and the impact that V-LN and/or V-RN has on their resilience. The percentage of change is shown relative to the cells average minimal hammercount. A negative percentage shows a decrease in resilience compared to the average minimal hammercount of the cell, whilst positive percentage shows an increase in resilience. The victim cell *T:23* for example has a total average minimal hammercount of 553475, when the cell's left neighbour V-LN is uncharged the resilience decreases 7.39% compared to the average minimal hammercount, whilst a charged V-LN cell increases the resilience. The results show evidence of resilience decrease caused by uncharged adjacent cells in the victim's row. A total of 11 from the 43 victim cells show this phenomenon.

Table 5.13: Impact of neighbouring victim cells V-LN and V-RN on victim cell resilliance

Victim Tag	Total Avrg. Min. HC	V-LN Charged %	V-LN Uncharged %	V-RN Charged %	V-RN Uncharged %
True-Cells					
T:3	624719	2.04	-2.04	0.26	-0.26
T:10	548359	6.74	-6.74	0.43	-0.43
T:15	468011	4.98	-4.98	-1.04	1.04
T:17	490597	3.46	-3.46	0.11	-0.11
T:23	553475	7.39	-7.39	-0.45	0.45
Anti-Cells					
A:5	576247	4.15	-4.15	0.05	-0.05
A:7	576247	0.23	-0.23	3.3	-3.3
A:9	558380	2.82	-2.82	-0.26	0.26
A:12	381161	3.60	-3.60	0.04	-0.04
A:15	642446	0.43	-0.43	2.48	-2.48
A:17	618959	2.82	-2.82	0.23	-0.23

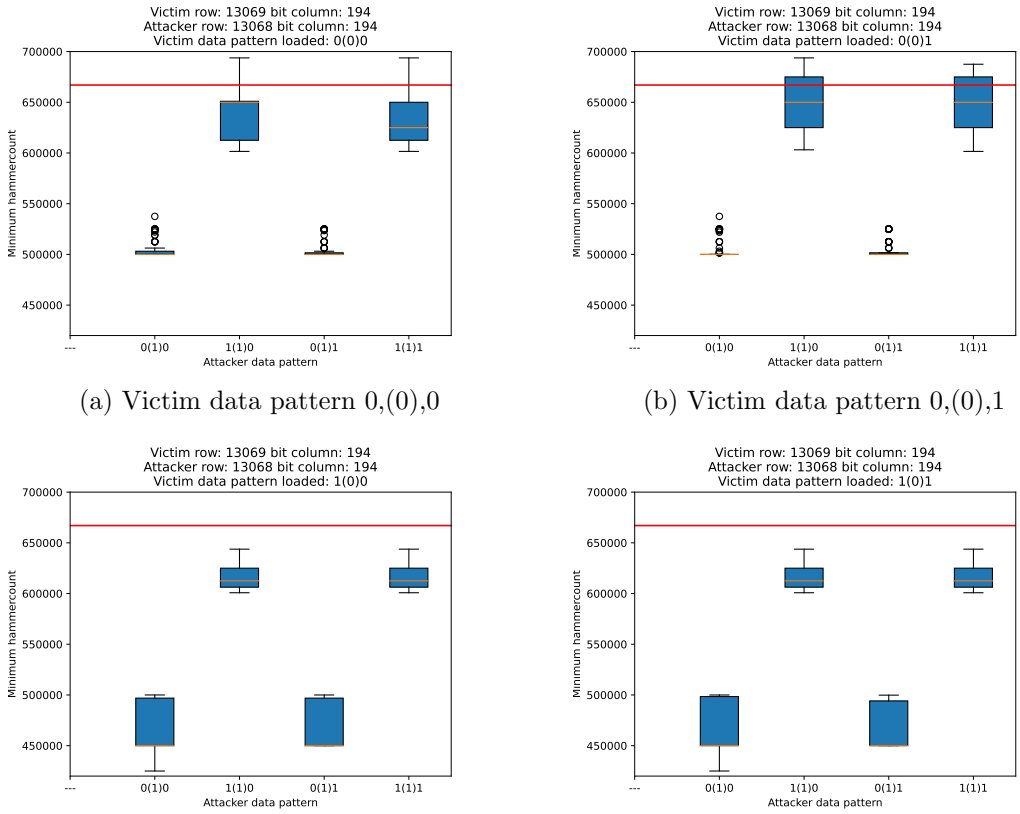
5.2.2.3 Diagonal cell impact on hammer resilience

The results earlier showed evidence of horizontal impact. In this part the impact of the neighbouring attacker cells will be discussed. Meaning that the attacker left neighbour (A-LN) and right neighbour (A-RN) have a diagonal impact on the victim cell. In the set of rows two attacking cell locations have been found to have a diagonal impact on the victim cell resilience. In Table 5.14 the impact is shown regarding the diagonal impact. In these two cases having a charged A-LN cell causes a negative impact on the victim cell resilience. In the case of *T:15* the *(*A-RN) cell being uncharged impacts the victim cell resilience less.

Table 5.14: Impact of neighbouring attacker cells A-LN and A-RN on victim cell resilliance

Victim Tag	Total Avrg. Min. HC	A-LN Charged %	A-LN Uncharged %	A-RN Charged %	A-RN Uncharged %
True-Cells					
T:15	468011	-7.61	7.61	1.43	-1.43
Anti-Cells					
A:9	558380	-12.91	12.91	0.08	-0.08

The cell that shows the largest influence of the diagonal impact is *A:9* as seen in Figure 5.19. Here the anti-cell shows a decrease in resilience caused by an attacker having patterns 0(1)0 and 0(1)1. Meaning that when A-LN is charged it has a diagonal impact on the resilience of the victim cell. Let it also be clear that the average minimal hammercount is influenced by the victim data pattern, this has already been discussed earlier and is presented in Table 5.13.



(a) Victim data pattern 0,(0),0

(b) Victim data pattern 0,(0),1

(c) Victim data pattern 1,(0),0

(d) Victim data pattern 1,(0),1

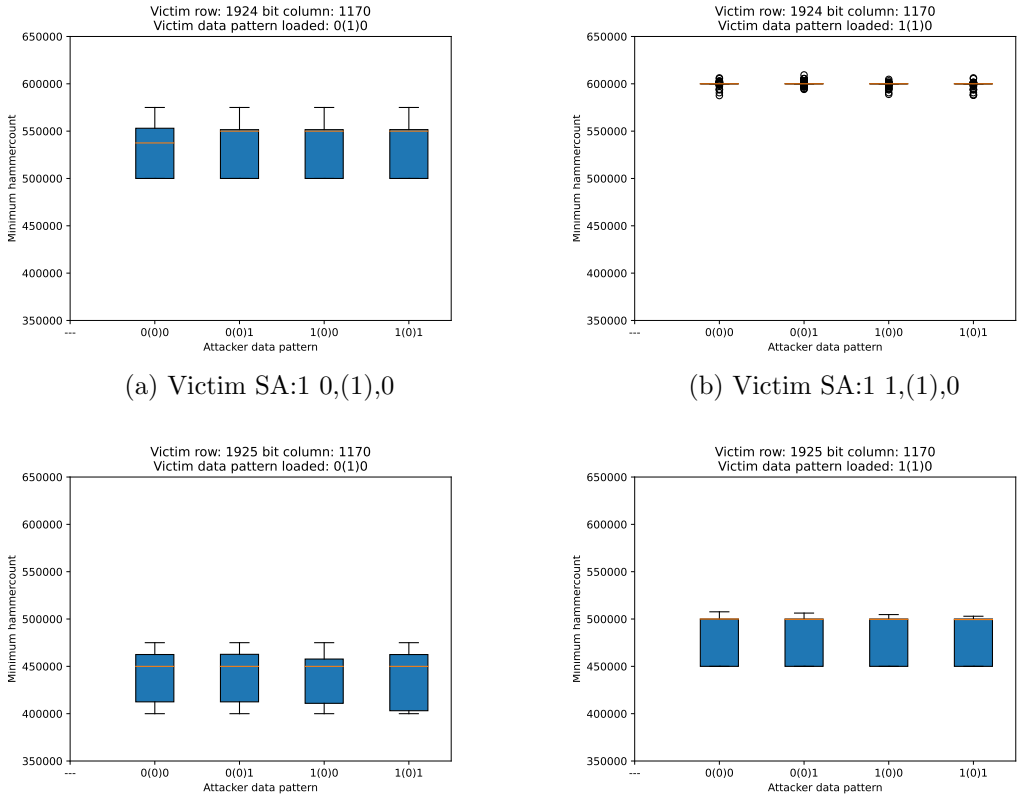
Figure 5.19: Attacker anti-cell neighbour diagonal impact on victim anti-cell

Seeing the results it becomes clear that aside from a victim cell being influenced by its direct horizontal neighbours it can also be diagonally influenced by the horizontally adjacent neighbours of the attacking cell.

5.2.2.4 Mirroring cell symmetry

In this part of experiment set 2 the results will be shown regarding the symmetry of mirroring cells, meaning cells in the same column but in opposite neighbouring rows. These cells have performed a rowhammer attack on each other on separate occasions and have shown that is was possible to flip their opponent. This impact is vertical and it is shown to be bi-directional. The cells locations are shown in Table B.1 and their group tag will be referenced throughout this section.

Horizontal impact is mirrored. The results show that both victim cells in the neighbouring row pair are either influenced or not by their horizontal neighbour (*V-LN* or *V-RN*). Figure 5.20 shows an example of a mirroring cell pair. These two cells both are influenced by their left neighbour *V-LN*. They do however show different average minimal hammercounts.



(a) Victim SA:1 0,(1),0

(b) Victim SA:1 1,(1),0

(c) Victim SA:2 0,(1),0

(d) Victim SA:2 1,(1),0

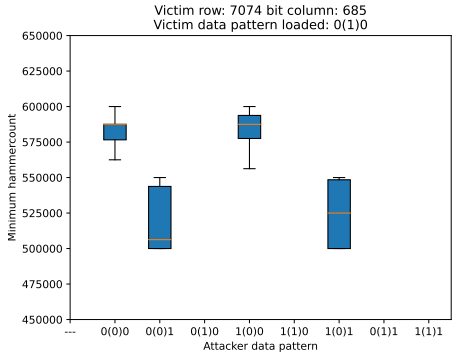
Figure 5.20: Victim cell pair SA:n hammercount influenced by horizontal neighbour V-LN

This phenomenon has been found in other mirroring cell pairs. In Table 5.15 the results show that either both cells in the mirroring cell pair suffer from their horizontal neighbour or neither. See *SE:1* and *SE:2*, these both suffer from their *V-LN* being uncharged, whilst *SD:1* and *SD:2* show no significant difference regarding their *V-LN* or *V-RN* cell charges. A total of 4 pairs shown to suffer from this phenomenon.

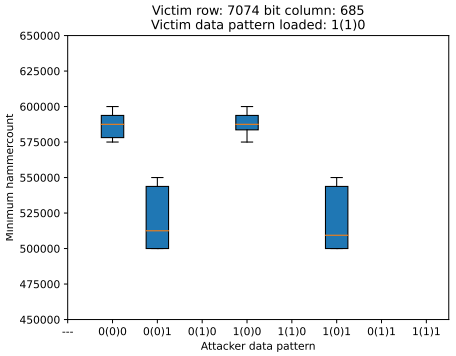
Table 5.15: Comparison of mirroring cell groups with their horizontal neighbour impact

Mirroring Victim pairs Tag group:n	Total Avrg. Min. HC	Diff (n-1) %	V-LN Charged %	V-LN Uncharged %	V-RN Charged %	V-RN Uncharged %
SA:1	564468	-	6.34	-6.34	0.07	-0.07
SA:2	460521	-18.42	4.31	-4.31	0.24	-0.24
SB:1	573309	-	0.02	-0.02	3.50	-3.50
SB:2	632380	10.30	-0.03	0.03	2.51	-2.51
SC:1	513554	-	-0.16	0.16	-0.03	0.03
SC:2	499928	-2.65	0.14	-0.14	0.15	-0.15
SD:1	473420	-	0.08	-0.08	-0.08	0.08
SD:2	628628	32.78	0.00	0.00	0.00	0.00
SE:1	595047	-	2.69	-2.69	-0.02	0.02
SE:2	560042	-5.88	6.26	-6.26	0.05	-0.5
SF:1	558300	-	0.02	-0.02	6.04	-6.04
SF:2	729926	30.74	0.00	0.00	3.45	-3.45

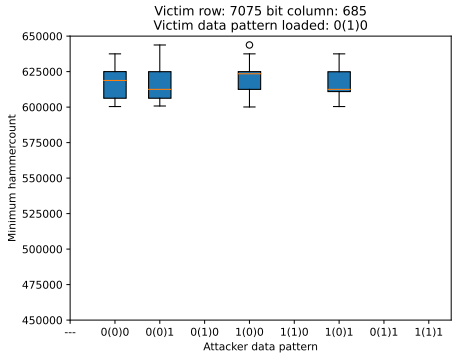
There is limited evidence of diagonal impact on mirroring cells. Only one cell was found to be influenced diagonally by the attackers right neighbour *A-RN*. This is shown in Figure 5.21. Here the one victim cell *SB:1* shows a significant difference in hammercount by the attacking cell's right neighbour whilst the other, *SA:2* shows no significant difference.



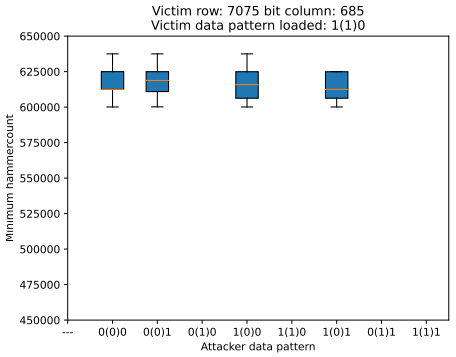
(a) Victim SB:1 0,(1),0



(b) Victim SB:1 1,(1),0



(c) Victim SB:2 0,(1),0



(d) Victim SB:2 1,(1),0

Figure 5.21: Victim cell pair SB:n hammercount influenced by diagonal neighbour A-RN

Similar to Section 5.2.2.3 there is less evidence of cells showing diagonal impact on victim cells compared to horizontal impact. Once again the results show the attacker cell's neighbour being charged decreases the resilience of the victim cell though. The results for cell pair *SB* are shown in Table 5.16. Here it is shown that only *A-RN* has influence on the resilience of the victim cell.

Table 5.16: Comparison of mirroring cell groups with their diagonal neighbour impact

Symmetric Victim pairs Tag group:n	Total Avrg. Min. HC	Diff (n-1) %	A-LN Charged %	A-LN Uncharged %	A-RN Charged %	A-RN Uncharged %
SB:1	573309	-	0.07	-0.07	-4.47	4.47
SB:2	632380	10.30	0.09	-0.09	-0.09	0.09

The average hammercount of mirroring cells varies between the pairs. Comparing the minimal hammercount of the cells within the mirroring pairs shows that there is significant difference in rowhammer resilience. One cell might be more resilient whilst its mirroring cell might succumb to a rowhammer attack sooner. These differences are shown in Table 5.15. There is as little as 2.65% difference in minimal average hammercount in the cell pair *SC*, whilst one pair of mirroring cells, *SD*, shows a difference of 32.78%.

5.2.3 Experiment set 3) Proposed Protection Method Results

The experiments that evolve around the protection method have been executed and the results are discussed in this section. To start the impact of the DDR3 commands in the rowhammer sequence are evaluated. These will act as the baseline when analysing the impact of the proposed protection method.

5.2.3.1 Impact of read and write commands in rowhammer sequence

Previously in Section 4.2.2.1 the experiment is run with rowhammer sequences composed of an an activate and precharge memory command. In this part the rowhammer sequence has been extended with read and/or write operations in between the activate and precharge commands.

The total number of memory clock cycles is shown in Table 5.17 for each type of rowhammer sequence. Keep in mind that during the one-sided rowhammer attack the row buffer in memory must be overwritten by performing an $activate \rightarrow precharge$ on a dummy row. When calculating the maximum hammercount within the 64 ms refresh period it is assumed that the same sequence of memory commands is performed on the dummy row.

The timing delay of a read followed by a precharge is shorter compared to a write followed by a precharge. This is due to the write command requiring a longer delay before closing the row.

5.2.3.2 Read and/or write commands affect hammercount

The results show that the rowhammer sequence with the additional memory commands does impact the hammercount. The average minimal hammercount only includes the

Table 5.17: The duration of the rowhammer sequences (CK) and the maximum hammercount within the refresh period

Rowhammer Sequence	Duration in clock cycles (CK)	Maximum Hammercount within 64 ms
$(act \rightarrow read \rightarrow pre)$	20	640000
$(act \rightarrow write \rightarrow pre)$	21	609523
$(act \rightarrow read \rightarrow write \rightarrow pre)$	34	376470
$(act \rightarrow write \rightarrow read \rightarrow pre)$	30	426666

results of charged victim cells and uncharged attacker cells as other combinations yield no results in the baseline. These cell group patterns are shown in Table 5.18.

Table 5.18: The cell data patterns per True and anti victim/attacker group

Cell type	Victim pattern	Attacker pattern
True	0(1)0, 0(1)1, 1(1)0, 1(1)1	0(0)0, 0(0)1, 1(0)0, 1(0)1
Anti	0(0)0, 0(0)1, 1(0)0, 1(0)1	0(1)0, 0(1)1, 1(1)0, 1(1)1

The results are compared to those of the baseline, this being from the the *activate* $\rightarrow$ *precharge* rowhammer sequences discussed earlier in Section 4.2.2.1. When including read and/or write commands in the rowhammer sequence the impact on the minimal hammercount does not seem to change much overall. Some read/write combinations do impact the minimal hammercount somewhat. Though some cells do show significant resilience increase or decrease when rowhammering with these combinations. Table 5.19 shows some of these examples. Here it victim cells *T:1* and *T:4* shows an overall improvement when being hammered by combinations including read and/or write. When these cells are hammered with the sequences including $read \rightarrow write$ and $write \rightarrow read$ their average minimal hammercount already exceeds the maximum allowed hammercount within the refresh period of 64 ms. This means that they would already be safe. This however is not the case for the sequences including *write* or *read* alone. Two cells, *T:18* and *A:13* show resilience decrease for some rowhammer sequences. For *T:18* it is especially problematic as the resilience victim cell is particular vulnerable for the rowhammer sequence including only a *read* command.

Table 5.19: Some outlying results

Victim cell Tag	Baseline Average Minimal Hammercount	Difference with Read	Difference with Write	Difference with Read -> Write	Difference with Write -> Read
T:1	471739	5.62%	8.07%	4.41%	7.09%
T:4	528707	5.16%	11.08%	7.17%	5.16%
T:18	591110	-10.67%	0.13%	-1.94%	-0.55%
A:13	628429	-1.07%	-0.80%	-6.53%	-10.62%

As mentioned earlier including the read and/or write commands increases the sequence execution time and decreases the maximum hammercount allowed within the refresh period. The set of victim cells that are hammerable within the refresh period has dropped significantly for the two longest rowhammer sequences. A total of 40 victim cells have been tested successfully. Following the experiment, 38 victim cells are hammerable using the $act \rightarrow read \rightarrow pre$ rowhammer sequence, 26 are hammerable by a $act \rightarrow write \rightarrow pre$ sequence, 6 are hammerable by a $act \rightarrow write \rightarrow read \rightarrow pre$ sequence and 4 are hammerable by a $act \rightarrow read \rightarrow write \rightarrow pre$ sequence.

5.2.3.3 Proposed Protection Method Results

The results regarding the rowhammer sequences without and with the proposed protection are compared. The attacking cell group is flipped throughout the rowhammer sequence. A *read* command alone cannot flip the attacker cell group content and thus cannot be tested. The 40 cells have been evaluated with the flipping method put into practice. In many cases no minimal hammercount can be measured when introducing the protection method. Meaning that the hammercount has been increased to a value outside of the maximum bound of the binary search. The results for the victim cells that are on the lower side of the resilience spectrum are shown in Table 5.20. Flipping during the various rowhammer sequences influences the resilience of the victim cell differently. In some cases the protection method eliminates the potential to flip the victim cell entirely and for other cases it decreases the resilience.

The protection method is considered a success if the victim hammercount is increased to a value above the maximum allowed hammercount that each sequence can perform. As seen in Table 5.21 the $act \rightarrow write \rightarrow pre$ rowhammer sequence causes 26 victim cells to fail without the protection method, whilst only 9 successful failures occur when

Table 5.20: Impact of protection method per rowhammer sequence

Victim cell Tag	Avrg. min HC write	Diff write*	Avrg. min HC read->write	Diff read->write*	Avrg. min HC write->read	Diff write*->read
T:1	471739	-	492540	-6.79%	505179	-
T:9	386250	16.81%	361906	-9.76%	357012	44.56%
T:14	346695	17.81%	340453	0.02%	342125	42.30%
A:4	380977	3.25%	381242	-	381836	49.57%
A:5	582582	3.25%	549196	-	533544	22.16%

the protection method is put to use. The other two rowhammer sequences already have a lower potential to cause failures in the victim cells within the refresh period. The protection method eliminates all victim cell failures during a $act \rightarrow write \rightarrow read \rightarrow pre$.

Table 5.21: Vulnerable victim cells without and with protection method

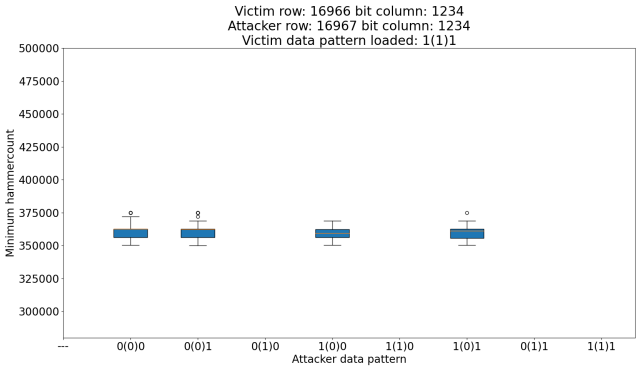
Hammer Sequence	Max HC ref. per. (64 ms)	Non-protection failures	Protection failures	Improvement rate
$(act \rightarrow read \rightarrow pre)$	640000	38	-	-
$(act \rightarrow write \rightarrow pre)$	609523	26	9	65.38%
$(act \rightarrow read \rightarrow write \rightarrow pre)$	376470	4	2	50.00%
$(act \rightarrow write \rightarrow read \rightarrow pre)$	426666	6	0	100%

All victim cells that are protected by the flipper protection method during an $act \rightarrow write \rightarrow pre$ rowhammer sequence are protected during the other two rowhammer sequences. The potential maximal hammercount of this sequence is highest compared to the other two. However maximal potential hammercount together with the protection method does not indicate success as two victim cells have been found to fail during the rowhammer attack with $act \rightarrow read \rightarrow write \rightarrow pre$ while not failing during $act \rightarrow read \rightarrow write \rightarrow pre$. These cells are *T:9* and *T:14*.

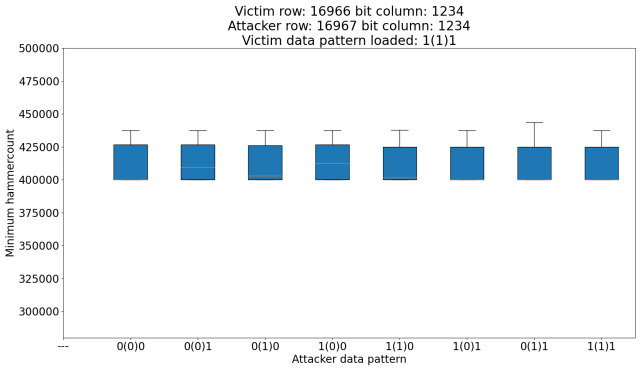
5.2.3.4 Victim cell flip under initially non-threatening circumstances

The ideal situation has been discussed previously. Here the victim/attacker cell combinations have been selected (charged victim, uncharged attacker) to measure the impact of the protection method compared to the baseline. However the protection method does have an unwanted side effect. The baseline results show practically no bit flips when the victim cell is charged and the attacker cell is uncharged. However the protection method results do show otherwise. The attacker cell initially may be charged before rowhammer, however when the protection method is put into use the attacker cell will also transition to, and, from an uncharged state throughout the rowhammer attack. Figure 5.22 shows that the victim bit is flipped under initially non-threatening circumstances. In Figure 5.22a the expected results are shown where a victim cell is hammered by an $act \rightarrow write \rightarrow pre$ sequence. Introducing the protection method causes the victim cell to fail for all attacker cell data patterns as seen in Figure 5.22b.

This negative side effect has impacted at least 9 victim cells during an $act \rightarrow write \rightarrow pre$ sequence and 2 during an $act \rightarrow read \rightarrow write \rightarrow pre$ sequence. Meaning that these cells could not be protected due to this negative side effect.



(a) ($act \rightarrow write \rightarrow pre$) With no protection



(b) ($act \rightarrow write \rightarrow pre$) With flip protection

Figure 5.22: True- victim cell vulnerability without and with protection in place

Chapter 6

Discussion

In this chapter the results following the experiments are discussed. Comparisons to literature are be made, new findings are discussed, an overview of victim cell influences is shown and the Row Flipper protection method is compared to state of the art.

6.1 Comparison and Verification

To start off, the DDR3 module that is used during experimentation is limited to one-sided rowhammer attacks. Because an attacking row only influences one single victim row during a rowhammer attack. This differs to what the literature describes where a one-sided rowhammer attack has the potential to inflict bit errors in two neighbouring victim rows. Figure 6.1 shows an open memory array architecture. This architecture has pairs of cells sharing the same potential, $V_{cc}/2$. This inhibits unwanted interaction between the row pairs that share the same potential, meaning *WL0* and *WL1* could disturb one another during rowhammer (as for *WL2* and *WL3* disturbing each other). Understanding this, it is highly likely that the DDR3 test subject has this array structure implemented within its memory banks.

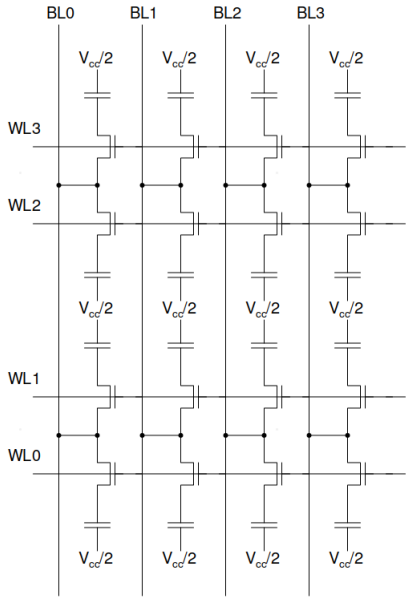
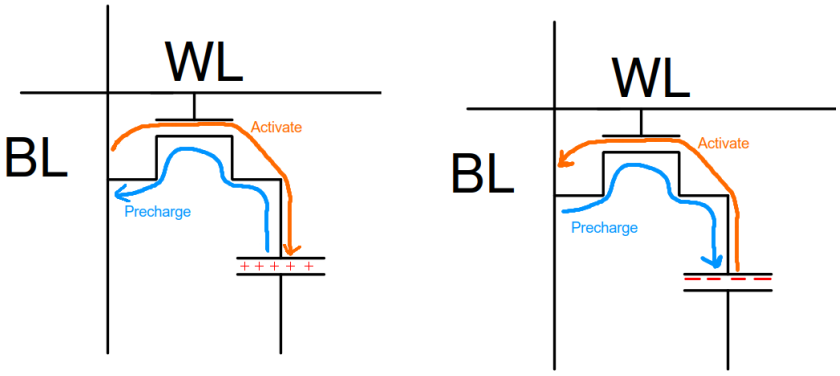


Figure 6.1: Open memory array structure

The existence of both true- and anti-cells have been found in the DDR3 module. These have been found using specific data patterns. There is no significant difference in bit error rate for the two types of cells after a 300 second retention test. However, there is a significant difference in bit error rate during rowhammer with solid patterns compared to rowstripe data patterns. When rowhammering with the solid data patterns the overall bit error rate is very low compared to the rowstripe data patterns. However, comparing the cell types during rowhammer with a solid data pattern shows that the anti-cells have an in increased bit error rate compared to the true-cells. Contrary when rowhammering with rowstripe data patterns the true-cells show to have a higher bit error rate compared to the anti-cells.

This is similar to findings in literature where victim cells only fail when they are initially charged, this is the case for retention time and rowhammer. It is considered possible to flip victim cells with charged attacker cells though, this is much less prevalent compared to rowhammer with uncharged attacker cells. Consider the trap-based failure mechanism that is described to accelerate capacitor discharge in the victim cell caused by traveling electrons. These electrons have become trapped in the attackers transistor during the closing of the attacker row and manage to escape to the victim cell. Since

rowhammer induces a higher bit error rate when the attacker cells are discharged one, could hypothesise that the direction in which these electrons flow through the transistor is essential to the effect caused by this failure mechanism. Figure 6.2 shows the direction of the electrons travel depending on cell charge during activate and precharge. In Figure 6.2a electrons travel from the bitline to the cell during activation, and in the opposite direction when precharged.



(a) Electron travel when cell initially charged

(b) Electron travel when cell initially uncharged

Figure 6.2: Electron travel to and from bitline depending on cell charge and command.

Wordline-to-wordline cross talk is another failure mechanism that causes bit errors to occur in victim rows during rowhammer. Under the assumption that the capacitive coupling is stronger the further away the wordline reaches from the wordline driver it would be expected that bit errors occur in cells furthest away from these components. To fully verify this the exact memory array size and structure must be discovered.

6.2 New Findings

Experiment set 2 was conducted with the aim to investigate the impact of data patterns on a cell level basis. The hammering of victim cells is performed including the victim cell's horizontal neighbours as well as, the attacker cell's horizontal neighbours. The victim cells resilience is measured by its minimal hammercount.

Results have shown that 19 of the 55 (Table A.1,Table A.2 and Table B.1 all included) total tested victim cells not only suffer from vertical impact (attacker cell to victim cell),

but are also significantly influenced by their adjacent neighbours in the same row. This is seen as horizontal impact. The results show that 35.5% of the victim cells suffer due to one of their uncharged neighbours. This is interesting considering that these cells do not play an active role during rowhammer.

Diagonal impact on the victim cell coming from adjacent attacker cells also seems to exist. However in a lesser form. Only 3 of the 55 (5.5%) victim cells have shown to suffer from this phenomenon. The 3 cases have shown that a charged cell in the attacker row, neighbouring the actual uncharged attacker has a negative impact on victim cell's resilience.

Pairs of cells that show vertical impact on one another have been tested in a similar setting as discussed earlier. It was discovered that these cells mirror each other when it comes to horizontal impact, meaning that they both suffer significantly or not at all. A total of 4 pairs of the 6 showed to have this phenomenon. Since this is shown to happen within pairs, it indicates that there is some kind of coupling between their bitlines. Meaning that when the victim cell and adjoining uncharged cell are simultaneously activated the voltage of the victim bitline decreases. Figure 6.3 shows the situation where the bitlines are coupled. Consider *Cv* to be charged and *Cv-rn* to be uncharged (as in the results). When *WL* is activated charge will flow from *Cv* to *BLv* and from *BLv-rn* to *Cv-rn*. After rowhammer *Cv* already has a lower charge and struggles to stay above the threshold voltage required by the sense amplifiers, due to the coupling effect between the bitlines the voltage on *BLv* will be pulled down by *BLv-rn*, thus decreasing the voltage of *BLv* below the threshold voltage.

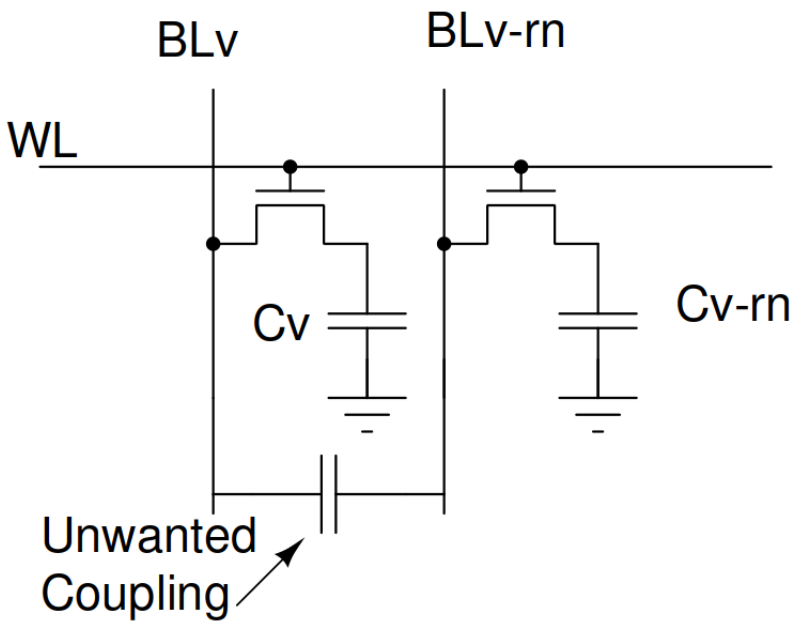


Figure 6.3: Potential vulnerability increase due to bitline coupling

The existence of these secondary effects need to be investigated with a larger test group and a higher number of iterations to give it statistical value. The findings in this thesis merely indicate the existence of these effects. In order to investigate the potential of bitline coupling one should perform the provided experiments on multiple victim cells within the same column.

6.3 Overview of Unwanted Influence on Victim Cell

An overview showing a number of unwanted influences on the victim cell is provided in Figure 6.4, these could potentially impact the resilience of the victim cell to rowhammer. The victim and attacker cell are marked separately. The image shows *Bitline coupling* between neighbouring bitlines, causing the horizontal neighbours to pull down the bitline voltage. *Electron De-trapping* from the attacker transistor causes the victim cell to lose charge during rowhammer. *Crosstalk* between the wordlines is shown, this causes the transistors to open slightly accelerating discharge of the victim capacitor during rowhammer. The victim cell suffers from *Capacitor leakage* and *Transistor leakage*. Finally *Diagonal Cell Charge Impact* is somewhat ambiguous, though has been discovered during experimentation.

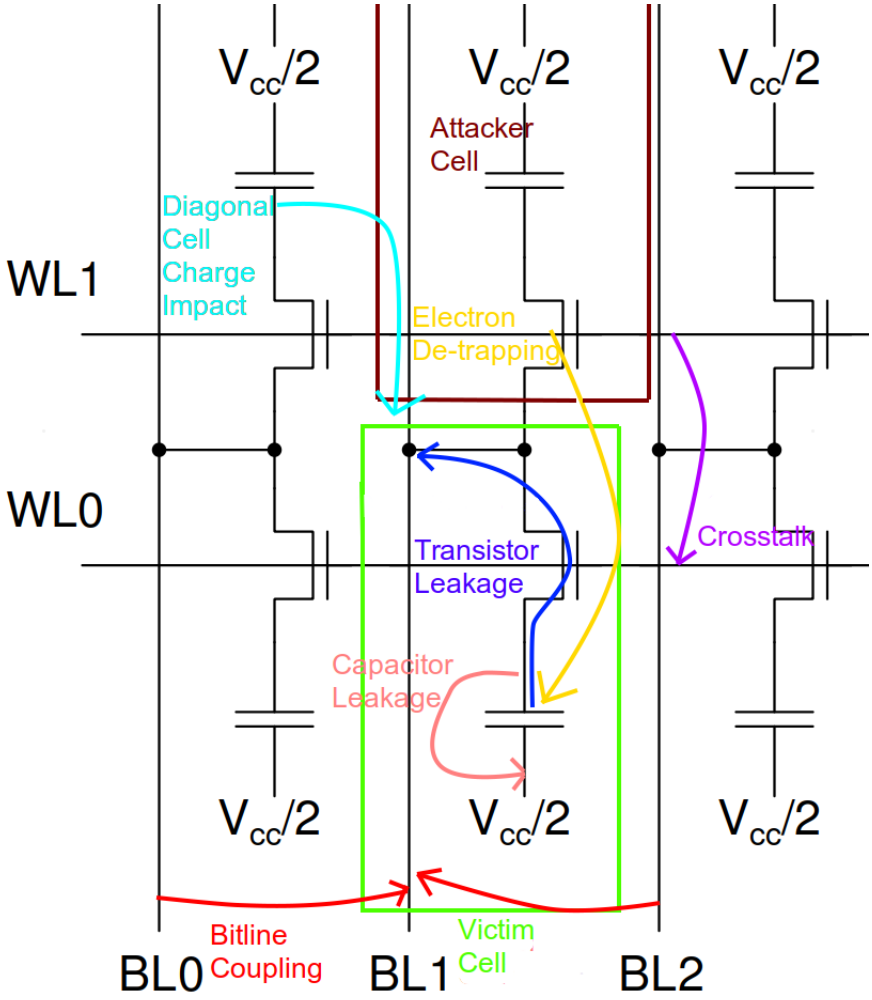


Figure 6.4: An overview of the influence on the victim cell

6.4 Proposed Protection Method & State of the Art

The proposed Row Flipper protection method has been tested by means of executing DDR3 commands. The write command is the actual command used to flip the attacker cell content.

First, the hammer sequence determines maximal potential hammercount within the refresh period. Certain rowhammer sequences require more clock cycles to execute and

thus are limited to a lower maximal hammercount within the DRAM refresh period. However, there are some cells that suffer regardless. The order of the write and read command also seem to influence the resilience.

The Row Flipper protection method shows potential. Many cells with varying resilience have shown to be fully protected by the protection method. Of the 26 victim cells that are vulnerable to an $activate \rightarrow write \rightarrow precharge$ rowhammer sequence 9 remain vulnerable when the protection method is used. When performing an $act \rightarrow read \rightarrow write \rightarrow pre$ the failure count drops from 4 to 2. Finally, $act \rightarrow write \rightarrow read \rightarrow pre$ shows that all 6 cells no longer fail when the protection method is used. All cells that where protected by the protection method during an $act \rightarrow write \rightarrow pre$ rowhammered sequence showed no failures when being hammered by the other sequence. The results also suggest that performing a read as the last command in a rowhammer sequence, while using the protection method gives the best protection coverage. Understanding this, one should consider inserting a read operation before a precharge for all other sequences in combination with the protection method for the best coverage.

The Row Flipper protection method does come with a negative side effect. When in use, it could potentially flip a victim cell under initially non-threatening circumstances. Rowhammer is significantly less successful when the attacker cell is charged. However due to the continuous flipping of data content the attacker cell will be discharged half of the time during a rowhammer attack.

As a reminder the protection PARA [9], PRoHIT [10] and MRLoc [11] operate on a probabilistic basis, whilst methods ideal-refresh-rate [6], CBT [12], TWiCe [13], Graphene [14], ARMOR [15] and BlockHammer [16] actively detect the attacking row. These protection methods protect the victim row by refreshing it under certain conditions (probabilistic or due to access patterns). The proposed Row Flipper protection method only concerns itself with the attacking row. This is more beneficial in terms of overhead and a plus side is that it can be implemented into DRAM chips.

With the understanding that newer DRAM technology is increasingly vulnerable to rowhammer one should consider using the proposed Row Flipper protection together with other protection methods. Using the protection method with those mentioned above would allow for lower victim row refresh rates. Another option is to decrease the refresh period in combination with the Row Flipper protection method. The Row Flipper method increases the minimal hammercount of a victim cell, whilst increasing

the refresh rate decreases the maximal allowed hammercount within the refresh period. The closer the minimal and maximal values are, the better the protection coverage.

Chapter 7

Conclusion

This is the concluding chapter of the thesis providing a short summary of the previous chapter, a conclusion in and recommendations in for additional work in the future.

7.1 Summary

Chapter 1, the introduction, provides the motivation to investigate the subject of this thesis. Modern DRAM technology is becoming increasingly vulnerable to rowhammer, and existing protection methods are unable to keep pace with this trend. The section State of the art discusses these protection methods. There are mainly 3 types of protection method, probabilistic, counter-based and prevention. The contributions of this research are to propose the Row Flipper protection method, to investigate the effects on the cells during rowhammer and to provide the basic results of the protection method.

Chapter 2, provides the basic background information needed to read this thesis and covers the basics of DRAM and rowhammer alongside with topics such as failure mechanisms and data patterns.

In chapter 3 the Row Flipper protection method is proposed. The general idea is presented and the expected results are discussed regarding the effects of the protection method during rowhammer. In addition some recommendations to implement this protection method are provided.

Chapter 4 covers the methodology and design and the methods used throughout the research are discussed. The list of experiments that are executed are discussed in a more abstract sense, and the design for the experimental infrastructure is shown.

Chapter 5 discusses the actual implementation of the experimental infrastructure and experimental results. An FPGA development board with DDR3 modules is used to execute the experiments and additional components are developed in order to execute precise memory commands on the DDR3 module. The PC sends the experiment instructions and receives memory data from the FPGA. The results following the experiments provide a characterization of the DDR3 module and show only a one-sided rohwammer attack has effect. The cell level experiments shows the existence of horizontal and diagonal interference aside from the expected vertical impact during rowhammer. The propose protection method experiment results show that there is an increase in resilience to rowhammer for most cells, with some suffering regardless.

Chapter 6 provides the results which are compared to, and verified with the findings in literature. Some new findings have been made following the results provided by the cell level experiments. The proposed Row Flipper protection method is compared to state of the art and a possible combination is also discussed.

7.2 Conclusion

In this thesis the impact of rowhammer has been researched to properly understand the impact of the Row Flipper protection method. Results have shown that victim cells need to be initially charged in order for a rowhammer attack to be successful. Furthermore, it is shown that attacking cells initially charged have a lesser impact on the vulnerability of the victim cells and that uncharged attacker cells cause a significant higher bit error rate during rowhammer. Specific victim cells have been analysed during rowhammer and it was found that in a number of cases one of the uncharged horizontal neighbouring cells caused a negative impact on the victim cell during rowhammer. It was also shown that opposing vulnerable cells mirror this effect. Pairs of cells have shown that both are impacted by their horizontal neighbours during rowhammer. Significant diagonal impact between the attacking cell's adjoining neighbour and the victim cell is also present, though to a much lesser degree, and was not mirrored in the cell pairs.

The proposed Row Flipper protection method has been tested by flipping the data in the attacker cell by using the write command. Increasing the rowhammer sequence length, by adding read and/or write commands, shows that the potential to successfully flip a victim cell drops even without applying the protection method. Applying the protection method shows a success rate of 50% for the $act \rightarrow read \rightarrow write \rightarrow pre$ hammer sequence, 65.38% success rate for the $act \rightarrow write \rightarrow pre$ sequence and a 100%

success rate for the $act \rightarrow write \rightarrow read \rightarrow pre$ sequence. Adding a read command after the write helps the Row Flipper protection method to increase its overall success rate. The protection method does however come with a negative side effect in that it aids the attacker cell under initially non-threatening circumstances.

In conclusion, the Row Flipper protection method has the potential to help current and future DRAM technology protect itself against rowhammer. The strength of the Row Flipper protection method is that it can be combined with other protection methods, and adds minimal overhead when implemented on the DRAM chip. Both are key reasons that applying the Row Flipper protection method should be investigated further on newer DRAM devices.

7.3 Future Work

The results presented in this thesis have raised some points for future work as follows:

- **Row Flipper Protection Method**
  The protection method has been proposed and some results show the impact it has on rowhammer. However, the experiments have been limited to DDR3 memory and a relatively small set of victim cells. Further work will be required to fully understand the impact of this method on newer DRAM devices and in combination with other protection methods.

- **Horizontal cell impact**
  Results have shown the impact of uncharged cells adjacent to the victim cell within the same row. Evidence of bitline coupling should be researched by experimentation on multiple victim cells within the same column.

- **Diagonal cell impact**
  Evidence suggests that there is diagonal cell impact on the victim cell. It is shown that charged adjacent cells next to the attacker cell have a diagonal impact on the vulnerability of the victim cell. This should be investigated further.

Appendix A

Selected cells for experiment set 2 and 3

Table A.1: Vulnerable true victim cells selected for experimentation

True-cell Tag	Victim location (row,column)	Attacker location (row,column)
T:0	(30546,167)	(30547,167)
T:1	(30546,267)	(30547,267)
T:2	(30546,448)	(30547,448)
T:3	(30546,486)	(30547,486)
T:4	(30546,556)	(30547,556)
T:5	(30546,572)	(30547,572)
T:6	(30546,615)	(30547,615)
T:7	(30546,652)	(30547,652)
T:8	(30546,892)	(30547,892)
T:9	(30546,1012)	(30547,1012)
T:10	(30546,1122)	(30547,1122)
T:11	(30546,1227)	(30547,1227)
T:12	(30546,1541)	(30547,1541)
T:13	(16966,1233)	(16967,1233)
T:14	(16966,1234)	(16967,1234)
T:15	(16966,1238)	(16967,1238)
T:16	(16966,1687)	(16967,1687)
T:17	(16966,1994)	(16967,1994)
T:18	(15503,428)	(15502,428)
T:19	(15503,832)	(15502,832)
T:20	(15503,1436)	(15502,1436)
T:21	(15503,1664)	(15502,1664)
T:22	(15503,1667)	(15502,1667)
T:23	(15503,1670)	(15502,1670)

Table A.2: Vulnerable anti victim cells selected for experimentation

Anti-cell Tag	Victim location (row,column)	Attacker location (row,column)
A:0	(26939,152)	(26938,152)
A:1	(26939,170)	(26938,170)
A:2	(26939,172)	(26938,172)
A:3	(26939,175)	(26938,175)
A:4	(26939,707)	(26938,707)
A:5	(13655,2026)	(13654,2026)
A:6	(13655,2028)	(13654,2028)
A:7	(13655,2029)	(13654,2029)
A:8	(13069,23)	(13068,23)
A:9	(13069,194)	(13068,194)
A:10	(13069,260)	(13068,260)
A:11	(13069,760)	(13068,760)
A:12	(13069,1106)	(13068,1106)
A:13	(13069,1299)	(13068,1299)
A:14	(13069,1324)	(13068,1324)
A:15	(13069,1409)	(13068,1409)
A:16	(13069,1724)	(13068,1724)
A:17	(13069,1990)	(13068,1990)
A:18	(13069,2003)	(13068,2003)

Appendix B

Selected symmetrical victim and attacker cells

Table B.1: Selected symmetrical cells

Symmetrical Cell Tag (group:number)	Victim location (row,column)	Attacker location (row,column)
SA:1	(1924,1170)	(1925,1170)
SA:2	(1925,1170)	(1924,1170)
SB:1	(7074,685)	(7075,685)
SB:2	(7075,685)	(7074,685)
SC:1	(9774,1860)	(9775,1860)
SC:2	(9775,1860)	(9774,1860)
SD:1	(11096,1168)	(11097,1168)
SD:2	(11097,1168)	(11096,1168)
SE:1	(12896,202)	(12897,202)
SE:2	(12897,202)	(12896,202)
SF:1	(14068,1989)	(14069,1989)
SF:2	(14069,1989)	(14068,1989)

Bibliography

[1] Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J., Lee, D., Wilkerson, C., Lai, K. & Mutlu, O. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *2014 ACM/IEEE 41st International Symposium On Computer Architecture (ISCA)*. (2014)

[2] Seaborn, M. & Dullien, T. Exploiting the DRAM rowhammer bug to gain kernel privileges . (2015), https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

[3] Kim, J., Patel, M., Yaglikci, A., Hassan, H., Azizi, R., Orosa, L. & Mutlu, O. Revisiting rowhammer: An experimental analysis of modern DRAM devices and mitigation techniques. *2020 ACM/IEEE 47th Annual International Symposium On Computer Architecture (ISCA)*. (2020)

[4] Jattke, P., Van Der Veen, V., Frigo, P., Gunter, S. & Razavi, K. Blacksmith: Scalable Rowhammering in the frequency domain. *2022 IEEE Symposium On Security And Privacy (SP)*. (2022)

[5] Park, K., Lim, C., Yun, D. & Baeg, S. Experiments and root cause analysis for active-precharge hammering fault in DDR3 SDRAM under 3 × NM technology. *Microelectronics Reliability*. (2016)

[6] Orosa, L., Yaglikçi, A., Luo, H., Olgun, A., Park, J., Hassan, H., Patel, M., Kim, J. & Mutlu, O. A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses. *CoRR*. (2021)

[7] Walker, A., Lee, S. & Beery, D. On dram rowhammer and the physics of insecurity. *IEEE Transactions On Electron Devices*. (2021)

[8] Keeth, B., Baker, R., Johnson, B. & Lin, F. DRAM Circuit Design: Fundemental and High-Speed Topics. (2008)

[9] Wang, Y., Liu, Y., Wu, P. & Zhang, Z. Discreet-para: Rowhammer defense with low cost and high efficiency. *2021 IEEE 39th International Conference On Computer Design (ICCD)*. (2021)

[10] Son, M., Park, H., Ahn, J. & Yoo, S. Making dram stronger against row hammering. *Proceedings Of The 54th Annual Design Automation Conference 2017*. (2017)

[11] You, J. & Yang, J. MRLoc. *Proceedings Of The 56th Annual Design Automation Conference 2019*. (2019)

[12] Seyedzadeh, S., Jones, A. & Melhem, R. Counter-based tree structure for row hammering mitigation in dram. *IEEE Computer Architecture Letters*. (2017)

[13] Lee, E., Kang, I., Lee, S., Suh, G. & Ahn, J. Twice. *Proceedings Of The 46th International Symposium On Computer Architecture*. (2019)

[14] Park, Y., Kwon, W., Lee, E., Ham, T., Ahn, J. & Lee, J. Graphene: Strong yet Lightweight Row Hammer Protection. *2020 53rd Annual IEEE/ACM International Symposium On Microarchitecture (MICRO)*. (2020)

[15] ARMOR A Run-Time Memory Hot-Row DetectOR. , https://apt.cs.manchester.ac.uk/projects/ARMOR/RowHammer/armor.html#top

[16] Yaglikci, A., Patel, M., Kim, J., Azizi, R., Olgun, A., Orosa, L., Hassan, H., Park, J., Kanellopoulos, K., Shahroodi, T., Ghose, S. & Mutlu, O. BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows. *2021 IEEE International Symposium On High-Performance Computer Architecture (HPCA)*. (2021)

[17] Baeg, S., Yun, D., Chun, M. & Wen, S. Estimation of the Trap Energy Characteristics of Row Hammer-Affected Cells in Gamma-Irradiated DDR4 DRAM. *IEEE Transactions On Nuclear Science*. (2022)

[18] Ryu, S., Min, K., Shin, J., Kwon, H., Nam, D., Oh, T., Jang, T., Yoo, M., Kim, Y. & Hong, S. Overcoming the reliability limitation in the ultimately scaled DRAM using silicon migration technique by hydrogen annealing. *2017 IEEE International Electron Devices Meeting (IEDM)*. (2017)

[19] Yaglikci, A., Luo, H., Oliviera, G., Olgun, A., Patel, M., Park, J., Hassan, H., Kim, J., Orosa, L. & Mutlu, O. Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices. *2022 52nd Annual*

IEEE/IFIP International Conference On Dependable Systems And Networks (DSN). (2022)

[20] Digilent, Genesys 2 Kintex-7 FPGA Development Board, https://digilent.com/shop/genesys-2-kintex-7-fpga-development-board/

[21] Micron Technology, Inc., 4Gb DDR3 SDRAM. (2008), https://static6.arrow.com/aropdfconversion/f1d4531e76b6d92bfc5a900a6e1ba0a3062d2282/234gb_ddr3_sdram.pdf,Rev.M4/13EN

[22] Xilinx, Xilinx Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions v4.2. *User Guide*. (2018), https://docs.xilinx.com/v/u/en-US/ug586_7Series_MIS

[23] Xilinx, Xilinx Answer 51204 MIG 7 Series DDR2/3 – PHY Only Design. (2012), https://support.xilinx.com/s/article/51204?language=en_US