POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

Power consumption measurements during NVM tests and solutions for low power embedded test code development

Relatore:
Prof. Paolo Bernardi

Candidato:
Lavinia degli Abbati

October 2020

Acknowledgments

This project was done in cooperation with Infineon Technologies AG and Politecnico di Torino. First, I would like to express my sincere gratitude to my supervisor Paolo Bernardi for his precious help and constant encouragement throughout these six months. He has shared with me his knowledge and experience, and I will always grateful to him for this.
Then I would like to thank Rudolf Ullmann for the great opportunity to do my master thesis in Infineon Technologies, for the constant willingness to help me and the huge professionalism.
Furthermore I want to thank everyone at Infineon who was involved into the project and who gave me all the support and information I needed. Thanks to the whole team of productive memory testing (PTE) and especially Michele Gasparini and Georg Horn for having supported me patiently and for the enthusiasm for their job that they have passed over to me.
Inoltre un sentito riconoscimento a tutte le persone conosciute in questi mesi di permanenza a Monaco. In particolare a Fabio, con cui ho condiviso questi mesi di quarantena lontana da casa. Sono stata davvero fortunata ad incontrare una persona del tuo spessore, mi hai trasmesso tanto. Ti porteró con me e faró miei i tuoi insegnamenti, soprattutto "non ti devi preoccupare per quello su cui non puoi avere controllo".
Grazie a P(i)erpaolo, che non riesce a liberarsi di me da ben due anni. Abbiamo condiviso ogni successo, sogno, gioia o dolore e ora anche una pandemia mondiale.

A tutte le persone che ho incontrato durante la mia carriera universitaria, ma soprattutto a Giuliana che ho sempre adorato e stimato fin dal primo giorno e a quelle tre pazze delle mie amiche: Elena, Eleonora e Sara. Siete state la mia famiglia per ben quattro anni, come dico sempre porto un pó di ognuna di voi in me. Grazie per i vostri saggi consigli, per esserci sempre state quando ho dovuto prendere importanti decisioni e per aver sempre creduto in me. Ma soprattutto grazie per i vostri sorrisi, che mi hanno accompagnato con gioia lungo tutto questo percorso universitario.
A Francesco, con cui ho condiviso sogni, aspirazioni e un pezzo di cuore. Grazie perché sei stato fondamentale in questo percorso di crescita. Da te ho imparato cosa cosa sia la "cazzimm" e di come sia essenziale avere grinta e una spiccata curiositá per arrivare fin dove si vuole arrivare. Ti auguro il meglio.
Alla mia amica storica, la mia omonima, Lavinia. Tu sei qualcosa di eccezionale, dovrei scrivere un diario intero di ringraziamenti per te. Grazie che nonostante questi anni di lontananza fisica, tu ci sia sempre stata. Grazie pr ogni messaggio

mandato prima di un esame :"falli secchi tigre!". Ti voglio bene.
A Flaminia, una cara amica che mi accompagna nel mio percorso di crescita fin da quando al liceo ripetevamo insieme la "MMMitosi e la MMMeiosi".
A Marco, che conosco oramai da piú di dieci anni e che é sempre presente nella mia vita, nelle mie scelte e soprattutto quando deve rimettermi la testa a posto perché sto perdendo la via.
Ad Elena, ancora, perché é dal liceo che mi é vicina e insieme abbiamo imparato a rialzarci dopo ogni difficoltá.
A Michele, che mi stimola intellettualmente e culinariamente.
A Leonardo, che mi ha spinto oltre i miei limiti, facendo sempre il tifo per me e dandomi la forza necessaria per affrontare ogni problema. Grazie per la tua pazienza e generositá.

Ai miei nonni e ai miei zii, a chi portruppo non c'é piú. Rimpiango gli anni universitari solo perche mi hanno rubato un pó di vita che avrei potuto passare con voi, vi ringranzio per la vostra immensa fiducia nei miei confronti e per aver sempre creduto nella vostra nipotina.
Infine alla mia famiglia, che é la piú grande fortuna della mia vita. Nonostante tutti questi anni di lontananza non mi avete mai fatto sentire sola, mi avete sostenuto sempre e mi avete fatto sentire amata come qando eravamo tutti sotto lo stesso tetto. Grazie mamma, papo e bea per il vostro amore e i vostri insegnamenti, ve ne saró per sempre infinitamente grata. Tutto quello che fino ad ora ho raggiunto é grazie a voi.

Table of contents

List of tables

List of figures

Acronyms

NVM: Non Volatile Memory
IC : Integrated Circuit
DUT : Device Under Test
TT : Test Time
HW : Hardware
SW : Software
ROM : Read Only Memory
RAM : Random Access Memory
MBIST : Memory Built-In Self-Test
MSIST : Memory Software-Implemented Self-Test
LFSR : Linear-Feedback Shift-Register
MISR : Multiple-Input Signature Register
mnsec : Minisector
Vcore : Voltage supply on the core domain (1.30V)
Icore : Current that flows on the core domain
TC : Test Code

Chapter 1

Introduction

In the last decades memories, the densest part of an Integrated Circuit, are increased in both density and performances. Their evolving complexity is becoming more and more important, this makes memory faults modeling and test algorithms areas of significant concern.

In the 2016 it has started a project about the Non-Volatile-Memory (NVM) testing, from the collaboration between the Politecnico of Torino and Infineon Technologies AG. The first steps of this project concerned choosing the most appropriate fault models, test algorithms and DFT techniques in order to achieve the highest fault coverage ($= \frac{\#detectedfaults}{\#overallfaults}$) in the smallest time possible and to have an IC easily testable and diagnosable.

The last part of this project started in March 2020 with the goal of validating the tests on the real device, in particular doing an analysis of the power consumed by the Device Under Test (DUT) when the tests are executing on it. The DUT can be tested with two different approaches : virtually or on the real physical device.[4] Virtual testing means using emulators or simulators that test the model of the silicon chip, of course the cost of the test is reduced, but the user conditions can not be replicated. Instead testing directly the real device is more expensive because certainly we need the physical device, but it has the advantage of being the fastest

approach. In this project the tests are executed on the real device and so the measurements that are taken.

Infineon Technologies AG has the aim to understand how much power is really consumed by the DUT during the NVM tests execution and to identify the major causes of this consumption. Consequently, trying to reduce them with the goal of executing more tests in parallel decreasing the overall test time, speeding up the time-to-market.

These are the objects of this work, in particular the test that are going to be characterised are the ones performed on the flash memories embedded on one of the last product of Infineon Technologies AG: A2G-TC38EVOx.

As all the Integrated Circuits, the TC38xEVO is mainly composed of two parts: the DIE, which is the silicon chip and the package which has the main function of protecting the DIE and to make it more handled. Every IC undergoes several phases before being ready to go into the market and according to STMicroelectronics [1] these phases can be mainly subdivided in two : front-end and back-end, which correspond respectively to the wafer fabrication and the packaging of the DIE.

Testing is a procedure that is done during each phase of the circuit production because the earlier the DUT is tested the more is the money we can save if it is faulty and clearly the single chip at wafer level costs less than the IC already encapsulated. Testing can chiefly be subdivided in wafer probing done on the DIE and final tests done on the IC.

The wafer probing is a step of testing done between the wafer fabrication and the packaging of the DIE. It verifies the physical wafer fabrication and also the functionality of each silicon chip, if one of the chips on the wafer does not pass the tests it is marked as faulty while the others are packaged in the form of Integrated Circuits.

The next tests are performed on the IC and to execute them we need a socket, whose function is to connect the DUT and the tester and it is soldered onto a Printed Circuit Board, in this way it is easy to add and remove an IC from the socket. It provides proper transmission paths between the (DUT) and the test board because in this way it is possible to apply on the DUT some stimuli from outside and collect the correspondence outputs. These outputs are compared with the responses given from an ideal good circuit and if they match it means that the circuit is good, otherwise that is faulty.

In this work are characterised the tests executed on the real DIE and on the real IC, and then the outcome of the measurements are compared to understand the main differences between these two results.

Once that all the critical tests have been characterised in terms of power consumption we have found three main solutions that do not have impact on the NVM tests quality but improve the power characteristic of every test. Each solution is based on the same assumption, which consists in saying that more are the context switches requested by the application and higher will be the overall power consumption.

The work is organized as follow:

- In Chapter two there is a brief introduction to the memories that can be embedded in a circuit and how they can be tested, focusing in particular on the FLASH memories.

- Chapter three contains a description of used the measurement environment, in particular how the measurements are collected and how it is possible to assure a consistency along the measurements taken in different interval of time.

- Chapter four develops an analysis on the effect that temperature, level of the

voltage supply and the operating frequency have on the device.

- In Chapter five there is an NVM tests characterisation and findings , with a short analysis on the portability of the test code among the different families of devices.

- Chapter six develops an analysis on the correlation between current jumps and voltage droops and how it is possible to reduce the voltage droops acting also via HW.

- Chapter seven contains a brief explanation of the wafer probe setup and comparison between the measurements taken with this setup and the ones taken while testing the IC.

- Finally, Chapter eight contains extra-analysis done but that unfortunately have not contributed to valid results.

Chapter 2

Volatile and Non Volatile Memories

Memories have the function to store data permanently or temporally depending on the needs of the IC where they are embedded in[16]. Therefore, the memories can be mainly classified in Non-Volatile-Memories(NVM) and Volatile-Memories (VM). In a volatile memory data are stored temporally and they are accessible only if the power is ON, so every time the system is rebooted the information is lost.

Instead in a Non Volatile Memory even if the power supply is switched OFF, the data are not lost.

Random-Access-Memory (RAM) is a VM used if there is no need to store information permanently but there is the need to have a fast access to the data stored inside it. As suggested by the name, this type of memory makes possible to access at the data in a random order and this characteristic makes the RAM the most used for holding data while the CPU is working on a task, since the CPU has not finished. According to Arunkumar Krishnan [20] there are two types of RAM: Static-Ram (SRAM) and Dynamic-Ram(DRAM), the first one is faster but less dense than the second one. The difference between these two is that a SRAM cell is composed by six transistors while a DRAM cell is composed just by a capacitor and a transistor so there is an advantage in terms of area occupied but due to the physical characteristic of the

capacitor the DRAM needs to be refreshed periodically. The two cells in comparison:

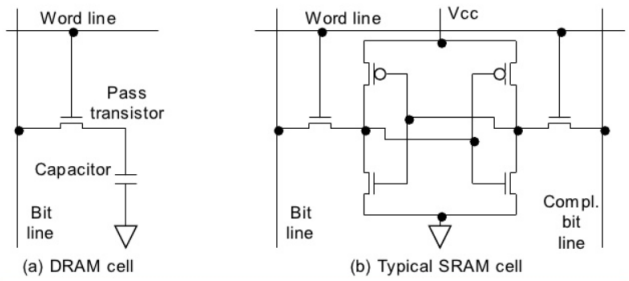

Figure 2.1: SRAM VS DRAM cell

On the other hand the NVM are used if it is needed to maintain the data stored inside the memory even if the power is switched OFF[17]. According to Paul Zandbergen [18], the NVMs can be classified as:

- ROM : Read-Only-Memory.

  It is used to store data that will remain the same for all the usage of the device, in particular it is mainly used to store information that are needed when the device is booted. A ROM is programmed during its manufacturing process because to achieve a certain configuration some specific internal fuses are blown and this makes the process irreversible.

- PROM : Programmable Read-Only Memory.

  It is programmed after manufacturing but also in this case the process is irreversible and so it can be programmed just once.

- EPROM :Erasable Programmable Read-Only Memory.

  This memory can be programmed more than one time, because it can be erased with a strong ultra-violet radiation through a window designed into

the memory itself. The erase process is not selective but it acts on the entire memory.

- EEPROM : Electrically Erasable Programmable Read-Only Memory.
  It is similar to the EPROM, but in this case to be erase it is not subjected to radiations, but to electrical signals that makes everything easier because the memory to be erased is not removed from where it is used.

- FLASH
  It is a form of EEPROM but faster due to it can be erased in block of bytes and not just byte after byte as for the EEPROM. This memory is the one that Infineon Technologies embedded in its products and in the next section it is described more in detail.

2.1 Flash memories architecture

Flash memories are Non Volatile Memories that can be electrically erased and programmed in block of bytes and they are composed by an array of floating-gate transistors, as written by T. Windbache [5]. There are two types of flash memories: NOR-flash and NAND-flash which names are explanatory of how the memory cells are organized. Infineon Technologies AG chooses the NOR-flashes to be embedded inside its ICs.
The DUT that has been analysed in this work is the A2G-TC38EVOx, which is one of the last versions of the Aurix devices designed by the company.
This IC has two type of NOR-flash memories embedded in it: Program Flash (PFLASH) and Data Flash (DFLASH). The first one is used to store programs

and it is divided in several independent banks while the second one is used to store data and it is divided into two separate banks. In a NOR-flash each cell has one end connected straight to ground and the other connected to a bit line. It acts like a NOR gate because when one of the word lines is high, the corresponding transistor pulls the output bit line low. Instead in a NAND-flash memory, the transistors are connected in series. The architecture of a NAND-flash and a NOR-flash memories are:

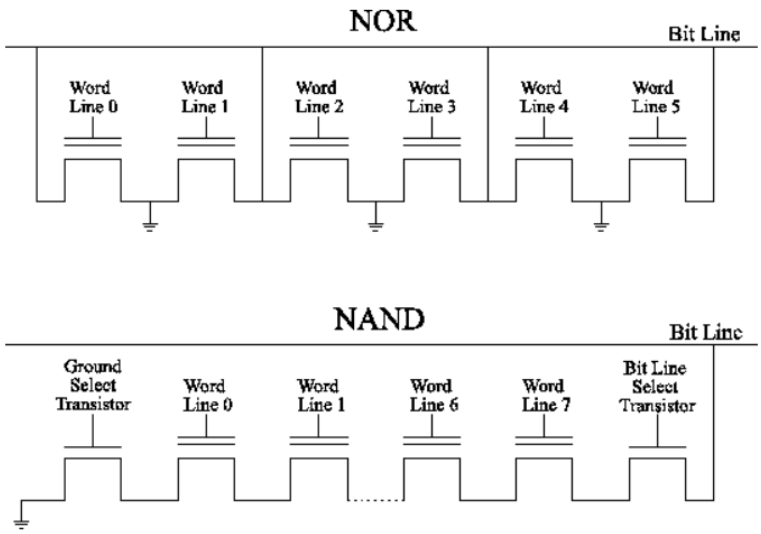

Figure 2.2: NOR-flash and NAND-flash architecture

As mentioned by Vatajelu Zambelli [6], a flash memory cell consists in a floating gate Metal Oxide Semiconductor field-effect transistor (MOSFET):

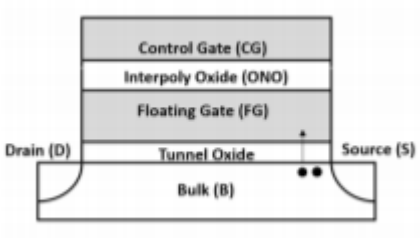

Figure 2.3: Cell flash structure

On top there is the control gate (CG) and below the floating gate (FG) which is insulated by two thick layers of oxide. The FG is inserted between the CG and the MOSFET channel and due to it is electrically isolated all around, any electrons placed on it are trapped there and when the FG holds a charge it screens the electric field from the CG and these electrons can remain there for many years. The presence of electrons in the FG modifies the threshold voltage (V_{Th}) of the MOSFET, in particular (V_{Th}) is shifted to be more positive if electrons are trapped in the FG, in contrary the presence of positive charges (absence of electrons) makes (V_{Th}) more negative. If the V_{Th} is lower than the one the MOSFET has in normal condition(so "positive" charges in the floating gate), we read a '0'(cell programmed), otherwise a '1' (cell erased).

The physical mechanism used to inject and extract electrons to/from the FG is the Fowler-Nordheim tunneling. It lies in applying a high electrical field to the thick tunnel oxide ($\sim 10MV/cm$) and this causes electron transfers across the insulator layer to the floating gate.

For programming the flash-cell it is applied an elevated voltage on the control gate that is coupled to the floating gate through the dielectric and in consequence also the FG is raised the to the programming voltage.

In this way the channel is turned on and the electrons flow from the source to the

drain with a great energy and they collide with the crystal lattice of the channel material. This dissipates heat which raises the temperature of the silicon, the electrons become "hotter" and many scatter towards the oxide layer, the ones which have a sufficient energy overcome the barrier and accumulate on the floating gate and they remain there until they are removed by an erase cycle.

After the programming operation the cell is set to '0'.

For erasing the cell it is applied a large voltage between the CG and source terminal, it makes the electrons jump the thin oxide layer and being trapped into the floating gate, in this way it is reset the cell to its natural state '1' [7].

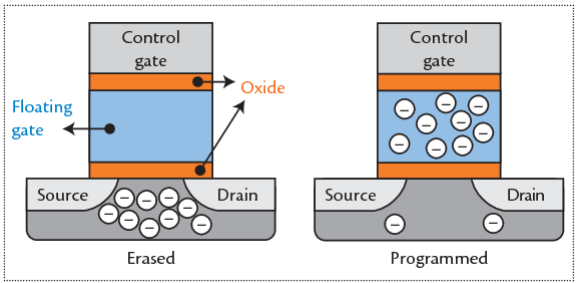

Figure 2.4: FLASH cell erased or programmed

Now it is clear how to program/erase memory flash cells and so we can explain how the flash memories are tested because basically testing memories consists of writing data, storing data and reading data from any cells.

2.2 Flash memories testing

Testing memories is essential because they are the denser part in an electronic circuit and so the probability of a fault to occur is very high. To analyze the memory circuit and develop techniques for the detection of failures, fault models are used. According to Einfochips [2] it is important to use good fault models able to test all the faults that could affect the memory cells and also implementing a self-repair of the faulty cells via redundant cells. The faults that could affect a memory can be grouped in two families:

- Faults among cells
- Faults in a single cell

The faults among cells are divided in Coupling Faults (CF) and Neighborhood Pattern Sensitive Faults (NPSF). CF manifest themselves as an interaction between the value of a cell and the value of another cell, so it could be that if in a cell it is stored a certain value, the value in another cell also changes. NPSF causes that a cell varies its content depending on the content of a group of cells.
On the other hand, the faults affect a single cell are the stuck-at-faults (when a cell is stucked at 0 or at 1), the stuck-at-open (the cell is inaccessible due to an open in the word line), the transition faults (cell can assume any value but then it cannot be changed) and the data retention faults (data stored in the cell is lost after a given period of time).
All these faults must be covered and this makes testing a very challenging and time consuming part during the development of a product. The March tests are the simpler and effective test algorithms to detect all the possible faults and to speed-up

the overall test time, According to Michel Linder [3]. Each March algorithm consists of a sequence of March elements, which in turns consist of a sequence of operations such as : writing a '0' into a cell, reading a '0' into a cell, writing a '1' into a cell or reading a '1' into a cell. Every element is applied to each single cell of the memory with a given order, from the lowest address to the highest or viceversa. An example of March algorithm is:

- write 0s (w0s) from the first to the last cell of the memory
- read 0s (r0s) and then write 1s (w1s) from the first to the last cell of the memory
- Read 1s (r1s) and then write 0s (w0s) from the first to the last cell of the memory

Test engineers decide which March element apply and in which order and basing on that, they write the test codes that will be then uploaded and executed on the device. To perform the test two approaches can be followed: MBIST(Memory Built-In Self-Test) and MSIST(Memory Software-Implemented Self-Test). [3]

- MSIST, Memory Software-Implemented Self-Test

  It has the advantage to be a flexible and cheap solution due to it does not require any additional hardware but everything is done via software. The tests are executed directly by the system itself and the input patterns are written directly inside the memory, as the test codes.

- MBIST , Memory Built-In Self-Test[19]

  It brings a reduction in the test time but also a hardware overhead, because

as suggested by the name, this technique demands for dedicated circuits built just for testing purposes able to run memory testing algorithms. The HW overhead is given by a MBIST controller which indicates when starts the test algorithm, and address generator, a data generator that produces the input patterns and a comparator which computes the signature of the outputs.

The BIST controller gives the input to start the self-testing procedure, once the input patterns have been produced by the data generator and applied to the memory, the output of the memory are compressed into a signature by the comparator. This signature is passed to the controller which compares it with the excepted signature, if the two do not match means the memory is faulty otherwise not.

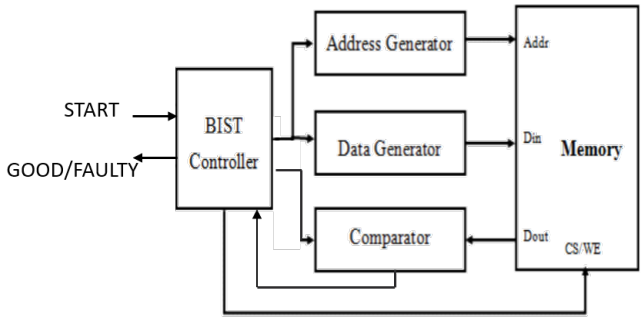

Figure 2.5: Memory BIST

Infineon Technologies AG organizes the NVM Testing with the execution of a software BIST. The CPU sends the "START" signal to the BIST controller that starts executing the tests, which mainly consist in successive sequences of flash readout at different conditions to verify the content of the FLASH after having erase/programmed it.

For this project we focus the attention just on one of these test sequences, which

is the first test sequence applied at the DUT and it is named WS1by Infineon. It is performed at room temperature($\sim 20°$). All the WS1 flow is collected in JAZZ, which is a tool to facilitate all the prototype debug, system validation, system characterization and productive test.

It provides built in tests or custom developed tests thanks to Perl Scripting support, but it allows also to use Python scripting.

Another feature is the easiness of reusability, because it is possible to use the same flow developed for testing / validating your product and apply it on several products or product samples at the same time, so increasing the production throughput without investing engineering efforts.

To test the IC, firstly it must be inserted in the socket, then we have to connect the board at the PC trough the MiniWiggler. This is an Infineon tool, it provides an USB interface to be attached at the computer and on the device side the connection goes over a 10-pin interface named DAP.

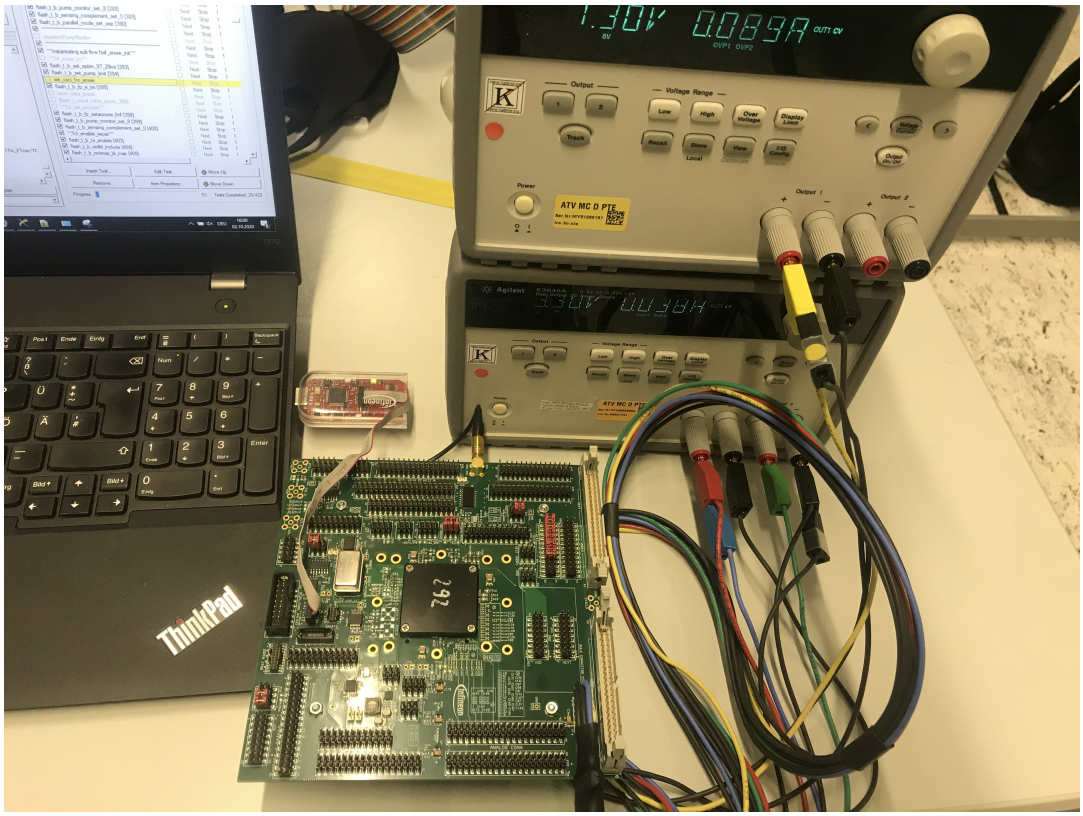

Figure 2.6: Setup

Then via JAZZ must be specified which is the executable that is intended to upload on the device. The executable is a file.hex built in TASKING and contains all the information about the tests we want to perform on the DUT, it has to be uploaded in JAZZ in the test *devcfg_t_b_ta_ld*.

After these two steps are concluded, we can power- on the system and run the JAZZ flow WS1. It includes an initial sequence of tests that have not impact in the FLASH memory itself but they are necessary to have all the analogue parameters that will be used as reference for the test that will be executed on the flash itself.

In the second part of the flow it is performed the chosen March algorithm, so there is a sequence of operations and each operation is repeated for all the memory cells

till all the operations are concluded. This sequence of operations is:

- Erase all the memory cells (w1s)
- Verify that the memory has been erased (r1s)
- Program all the memory cells (w0s)
- Verify that the memory has been programmed (r0s)

Between these tests it is possible to vary some test conditions, such as the frequency of the PLL and also to enable/disable circuitries inside the DUT, according to which is the final goal we want to achieve. All the above tests are characterised in terms of power and these characterisations are shown in the next paragraphs.

Chapter 3

Working setup

Having a "strong" and stable working setup is essential in project like this because it is important to assure the same conditions for all the measurements. In this way all the measurements taken by the same instrument, under the same conditions will give the same result.

The environmental conditions such as humidity and room temperature ($\sim 20°C$) remain stable for all the measurements campaign because this is guaranteed by the laboratory rules. Instead there are other parameters that need to be controlled by us. For instance the oscilloscope's channels must be set in the same way for all the same kind of tests. Differently, the risk is that we could misinterpret some measurements when we compare them if they have been collected with different time (s/DIV) or amplitude (V/DIV) scale.

Another important aspect is that we must use always the same IC for all the measurements and assure that the IC does exactly the same operations before performing the test we want to measure. Otherwise the risk could be that in a measurement are present power variations that in another measurement of the same test are not. Moreover when it is analysed a measurement and it is needed to compare it with another one, we have to be sure we are watching the same time window. For this reason it is necessary to trigger the oscilloscope always with the same signal.

3.1 Busy signal used as trigger

It is essential to trigger the oscilloscope when a test is being executing, because in this way we can measure the current requested by the DUT during this interval of time and do our analysis. Moreover it is important we visualize on the oscilloscope always the same interval of time because otherwise we could not make comparison between two measurements of the same test taken in different moments.

To have a signal that allows us to understand when the test is being executing it is taken advantage of the Infineon's FSIST Handshake Protocol for LL-FSIST commands [8]. This protocol uses two control signals: Busy (BUSY) and Error (ERR) , the BUSY is the one used as trigger for the oscilloscope. The BUSY signal remains high as long as the test is not completed. When it is completed, the signal goes low indicating that the results are available to be read. As said before, the BUSY signal has been mapped on the board in one of its external pin to make possible a connection of the board with the oscilloscope through a probe.

An example for what we visualize on the oscilloscope is reported below:

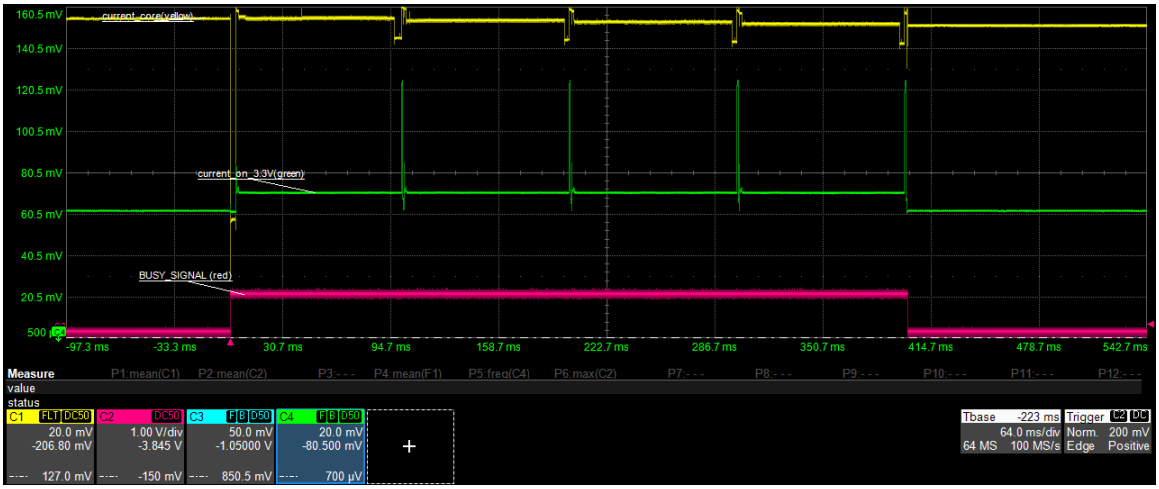

Figure 3.1: Erase test: Icore(yellow) ; Ivddp3(green) ; busy signal (pink)

In particular the image shows the current requested by the erase test from two domains and the busy signal. As a matter of fact the busy signal goes up when the tests starts and down when it is finished.
The colours correspond to these signals:

- yellow signal: Current on the core domain (Icore) (20 mA/div)
- green signal: Current on the 3.3V domain I(Vddp) (20 mV/div = 20 mA/div)
- pink signal: busy signal

These scales of time and amplitude are the same for every erase test that we have measured in this project.
Of course for the verify test the time scale and the amplitude scale are different with respect the one for the erase, because the verify test is characterised by another test time and it requires other levels of current to be executed.
In conclusion, when the tests are executed via JAZZ the oscilloscope is always triggered by the BUSY signal that goes high when the test starts and thanks to this, it is possible to compare measurements of the same test but taken in different moments under the same environmental conditions.
Notwithstanding to have a stable system but to avoid setting everytime manually the amplitude/time scale of the oscilloscope, it has been developed an interface between the laboratory instruments and the PC. Thanks to this interface we have control both on the oscilloscope and on the power supply and this means we can set the channels of these two instruments directly from the PC without doing it manually and also save the measurements as waveform objects saved on the PC. All the characteristics of the interface are explained in the next section.

3.2 Interface for measurements automation

The interface between the oscilloscope and the PC and between the power supply and the PC have the goals of setting the channels of the oscilloscope, of acquire the waveforms visualized on the screen of the oscilloscope and also of setting the power supply channels depending on the level of voltage with which we want to supply the device.

Three Python scripts have been developed to do all of these steps and two of them are inserted in Jazz in a specific order.

The order in which they are inserted in the Jazz flow is schematized in this flow chart:

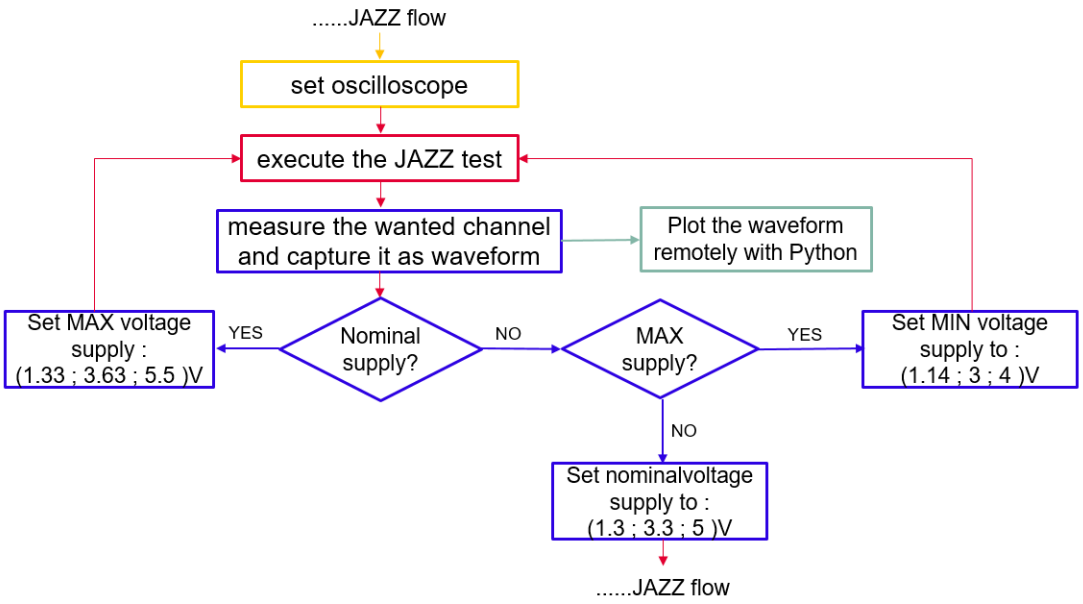

Figure 3.2: Project flow for test automation

Every colour corresponds to a Python script:

- Yellow: **Before_test.py**

This python code is inserted in Jazz before the test we want to analyse. Its function is to set the oscilloscope in a proper way. (trigger, time/DIV, V/DIV)

- Red: it is the **test** under analysis
- Blue: **After_test.py**

  This python script is inserted in the Jazz flow suddenly after the test we want to analyse. Its functions are to collect the data, save them in a "file.npz" and change, if requested, the voltage supply.

- Light green: **Manipulate_data.py**

  It is not inserted in the JAZZ flow and its function is to open the "file.npz" previously saved and remotely analyse the data collected and plot them again.

These scripts are reported in the Appendix A.

The first two scripts are inserted in the JAZZ flow, one before and one immediately after the test we want to evaluate, while the third one is run externally from Jazz.

Usually the oscilloscope needs to be set manually before performing a measurement and then if it is needed to save it, this measurement has to be saved as a screenshot and then import it into the computer.

This process requires a considerable effort for the engineer that has to stop the test flow on JAZZ; wait for the oscilloscope to be triggered; capture the image; save it and import it on the computer as a "file.npg".

All of these steps are automatize with the two scripts: "before_test.py" and "after_test.py" and so the repetitive manual labour is reduced, that results in time savings but also in an increased consistency and repeatability of the measurements taken.

The effort in the usage of the oscilloscope is mainly caused by:

- Saving screenshot requires to connect the USB on the oscilloscope, print the image on the USB, disconnect it and import the image on the PC.
- Once the image is captured and imported on the PC it is "freezed" and this makes impossible a remotely analysis on it.
- Difficult comparison between waveforms of the same type but taken in different moments
- The oscilloscope LeCroy WaveRunner 640Zi has four channels so it is impossible to visualize in the same time more than four signals together.

The original flow of JAZZ has been modified adding the developed Python scripts able to set the oscilloscope in the right configuration before the test is executed. Once the test is finished, the data we visualize from the screen of the oscilloscope are saved into the PC as waveform objects.
Once this is done from the PC it is possible to remotely processing the data collected with Python while the JAZZ flow can continue running.
In the meantime that the data have been saved, with some "if-branch" it is controlled which is the value of the voltage supply and we can change it without stopping JAZZ flow. After the new voltage supply has been set, the test is repeated again and data are collected.
Once we have all the wanted measurements saved in our PC, we can perform all the analysis we need. For example if we have peaks (such as in the verify0 test) it is possible to know how much they are just specify in the script to count them, or if we want to zoom in a particular part it is not needed to save again the data again from the oscilloscope to the USB, but it is needed just to zoom on this interested part with the mouse.

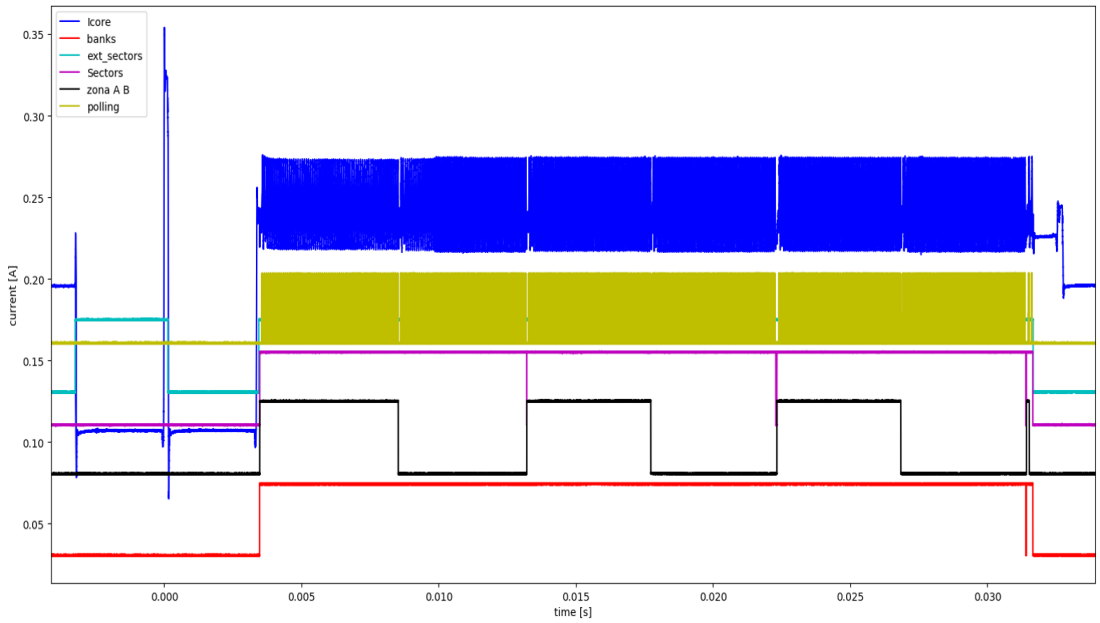

Figure 3.3: Example of measurements obtained with Python

This is the verify0 test and as it is possible to notice the signals plotted together are more than four.

Now it is possible to zoom on the image just with the mouse:

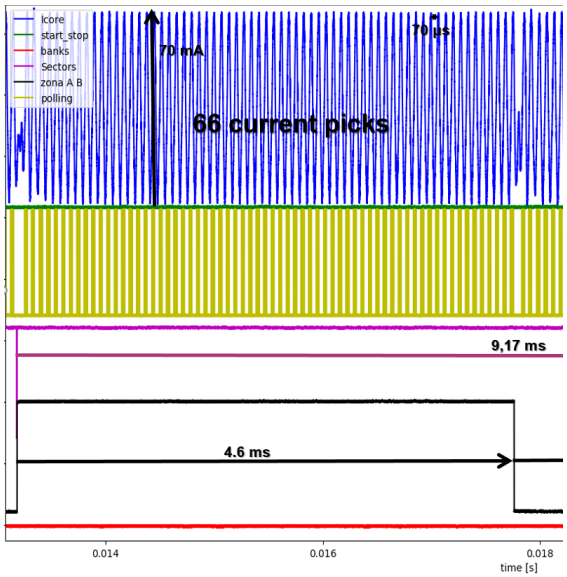

Figure 3.4: Zoom in the test

After having specified into the Python script to count the peaks of the current core for every zone, the result has been saved into a variable and it is 66. In this simply way now it is known that every zones is composed of 66 current peaks, each of 77mA with a periodicity of 70μs.

To better understand what this method is able to do are reported two images of the same measure, one obtain with the oscilloscope and save on a USB memory and one after having executed the three scripts:

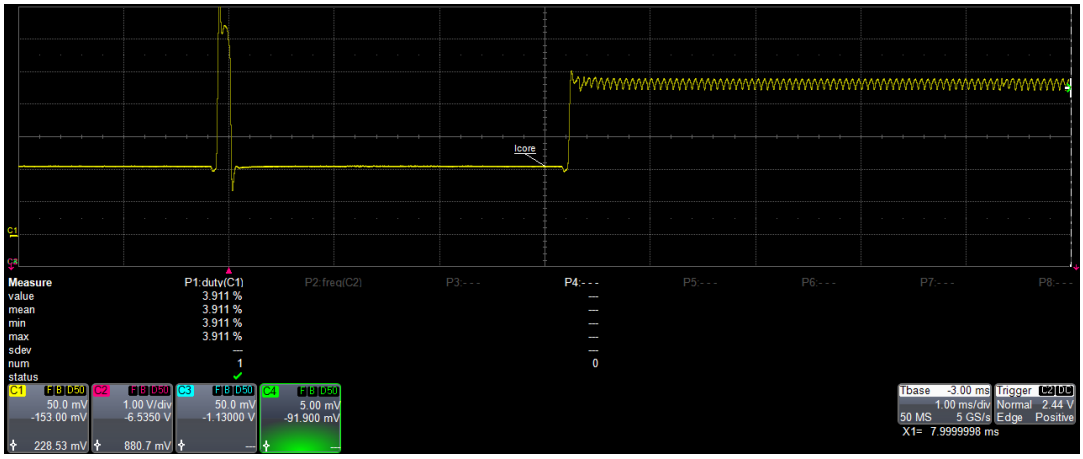

Figure 3.5: Verify test measured from the oscilloscope

Instead importing it as a waveform type from the oscilloscope to the laptop with an USB cable, we plot it with another code in python and it is obtained :

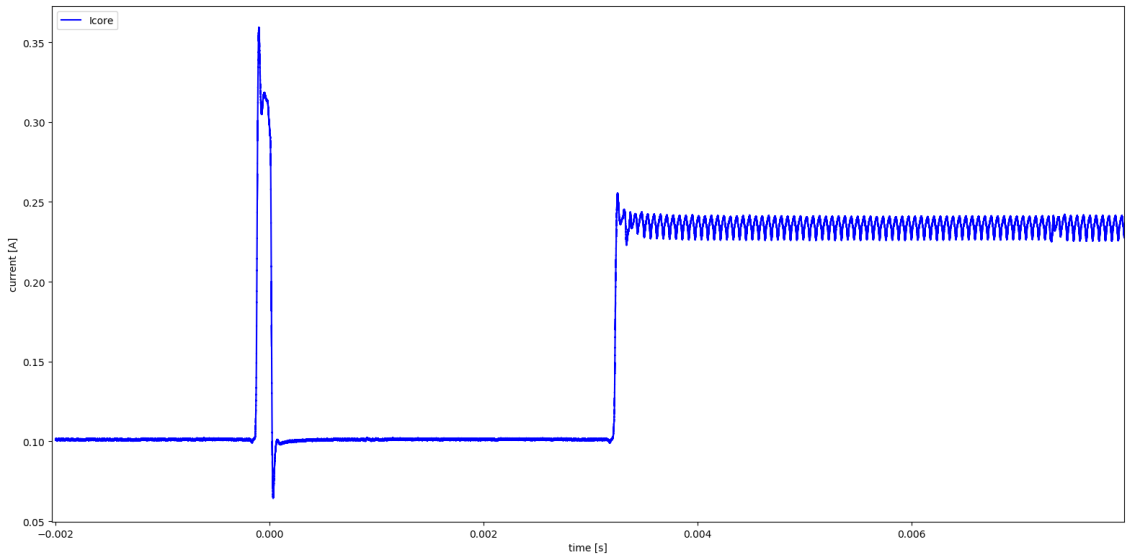

Figure 3.6: Verify test measured and plotted with Python

To develop the scripts it is used Python. It is an object-oriented language and the main idea is to define an object: "scope", which corresponds to the oscilloscope with which we are doing the measurements and then define its attributes such as the number of channels, the sampling frequency and the trigger channel, the trigger level and so on.

To specify which oscilloscope we are using we must connect it with the PC trough an USB cable, and from the National Instruments Measurement Automation Explorer (NI MAX) environment its IP number must be copied and pasted in the script, as reported below:

- *scope = SimpleScope(IIviScope())* it is created the object "scope" for the oscilloscope
- *resourcename = 'USB0::0x05FF::0x1023::2801N55814::INSTR'* IP number of the oscilloscope

"Before_test.py" is a script in which we specify the oscilloscope we are using , with the functions above, and then we specify its attributes, such the number of channels we will use, the characteristic of each channels in terms of amplitude/DIV, offset, zero point and then the sampling frequency , the time/DIV and the coupling mode. As explained before this script is inserted in JAZZ before the test that will be analysed.

It must be specified that to decide the value at which the each channel of the oscilloscope must be set, we have to run the test at least one time and observe which are its characteristics in terms of range in which the current could vary and test time in order to set correctly the time/DIV and the amplitude/DIV on the oscilloscope for that particular test and maintain it for all the successive measurement on the same test.

The code have been written including the PyVerify package, which supports automation. This package includes different functions that make possible the setting of the oscilloscope from the PC.

The function used to specify the attributes of the object "scope" are:

- *scope.ScopeSetup(0.6, 50e6)*

  it is set the sampling frequency of the object "scope", in particular it is chosen to set it at its maximum level which is f_{sample}=50 MS/s. In this way we have the highest resolution for the measurements.

- *scope.Trigger_Edge(Level=0.5, Slope='RISE', Position=1, Coupling='DC', ChannelIndex=2)*

  it is decided the trigger channel, level an position and also the coupling mode of the oscilloscope. In this case the trigger-channel is the channel number

2(where it is connected the busy signal), which triggers the oscilloscope every time there is a rising edge at its 50% of dynamic. The coupling mode chosen is the DC one, because with the AC mode it is introduced more noise on the waveform.

- *scope.ch(i) = scope.GetChannel(i)*
  it is defined which channels of the oscillosocpe will be used for measurements, in our project all the four channels are used.

- *scope.ch(i).ProbeSetup(Coupling='DC', Bandwidth=20e6, Vrange=0.160, Offset=-0.270, Position=0, Probe_Attn=1, Probe_Type='voltage', Impedance=50)*
  all the attributes of the specified channels "i" are set.
  It is choose the $50\Omega - DC$ coupling mode to have less noise overlapped on the measurement, then it is decided the bandwidth of the oscilloscope and then the offset, the probe attenuation. The amplitude/DIV for that particular channel, in this example the total dynamic visualized on the screen is 0.16V that means 16 mV/DIV due to the oscilloscope has 10 DIV in vertical, while the offset is set to -270mV.

"After_test.py" instead is executed after the execution of the test we want to analyse and its functions is to collect the signal on the oscilloscope and save them into a "file.npz" that then will be manipulated with another script ans also to set the properly the power supply. To do so, it is used again the PyVerify package and it is defined the "scope" object with its channels that are intend to measure and then for each channel are collected the data, that are saved in a "file.npz" in a specific folder specified in the path.

- *dataOfCH(i).save_to_file('C:/Users/file.npz')*

  After having save the data coming from the wanted channels, it is controlled the level of the power supply and change it if required. Others two objects are created "PWR_CORE" and "POWER_Vext_Vdp" because two power supplies are used, respectively one to generate Vcore(1.3V) and the other to generate Vext(5V) an Vddp3(3.3V).

  The flow is similar to the one we have already analysed for the oscilloscope, so for every object are then specified the attribute. To create the object:

- *PWR_CORE = SimpleDCSource(IIviDCPwr())*

  object "power-supply" created for Vcore

- *resourcename_core = 'GPIB0::5::INSTR'*

  IP number of the power supply used to define its attributes, that in this case it has just a channel for the generation of Vcore:

- *PWR_CORE.chcore = PWR_CORE.GetChannel(1)*

- *PWR_CORE.chcore.Configure_VoltageLevel(Level=1.30, CurrentLimit=1)* to specify the value we want generate from that channel, in this case 1.3V with a current limit of 1A.

Then it is done the same for the other power supply. In this script there are some nested "if-else" to specify which value we want to set on the power supply considering the situation in which we want to repeat the measurement changing the power supply from the nominal to the maximum and then to the minimum values for Vcore, Vext and Vddp3.

The last script is the **"Manipulate_data.py"**, its function is to post-process the

data that have been saved previously. For every waveform saved it is created a waveform-object :

- *wave1 = Waveform.load_from_file(filepath="C:/Users/file.npz")* in wave1 is copied the content of the "file.npz" that has been previously created with the data of the waveform captured with the oscilloscope Then for every waveform object are specified its attributes:

- *sample1= wave.data*

- *time1=wave.time* and then it is used the package "pyplot" to plot the waveform:

- *plt.plot(time1, data1, 'b', label=' Icore')* it is plotted the waveform with time1 on the abscissa, data1 on the ordinate, with blue color and with "Icore" as label.

  Of course it is possible also to plot more waveforms together just specifying others waveform objects with their attributes and plot them.

3.3 Current measurements

The starting step of this work is characterising each NVM test in term of power consumption. In the laboratory the usable tools for this purpose are the oscilloscope and the current probes, this is why are performed current measurements and not directly power measurements. The main assumption is that current is directly proportional to static power according to the formula:

$$P = I_{V_{dd}} \cdot V_{dd}$$

V_{dd} is the voltage set on the power supply, while 'I' is the overall current that flows through the cable from the power supply to the board.
The value of the voltage supply should remain almost stable for the entire period in which the device is ON, while the current changes depending on the amount of work the DUT has to do in this way we have an indirect measure of the power requested by the DUT from this domain.It is analysed the current $I_{V_{dd}}$ requested by the DUT during the test execution.
The device AURIX-2G TC38xEVO works with three power domains which are:

- Vcore = 1,25 V (the device still works in the range of $1{,}25V \pm 10\%$
- Vddp = 3,3 V (the device still works in the range of $3{,}3V \pm 10\%$
- Vext = 5 V (the device still works in the range of $5V \pm 20\%$)

Each of this domain is used for different purposes by the device and so the currents that flow from them into the device have completely different shapes. At the beginning of this project all the three currents were analysed, but after few analysis on the measurements we have discovered that the most critical domain in terms of variations is the core domain, this is why it is chosen this domain to perform the next analysis.
The oscilloscope does voltage-measurements, for this reason if we want to measure the current we need to use some dedicated instruments that allow us to visualize the current measured directly from the oscilloscope.
According to the availability of the Infineon laboratory, these instruments that are:

1. CURRENT PROBE

   It is used a current probe (Tektronix TCPA300) [9], that enables to measure

current without breaking the electrical circuit because it is not electrically connected to the DUT.

The probe is clamped around the power supply cable that carries the current in which we are interested and then thanks to the two sensors inside the probe it is measured the strength of the electromagnetic field around the conductor. The complete current measurement system consist of a the current probe Tektronix TCPA300, the probe amplifier and then the oscilloscope. The aim of the amplifier is to converts the current sensed by the probe to a proportional voltage that is displayed on the oscilloscope. The structure of the current probe is:

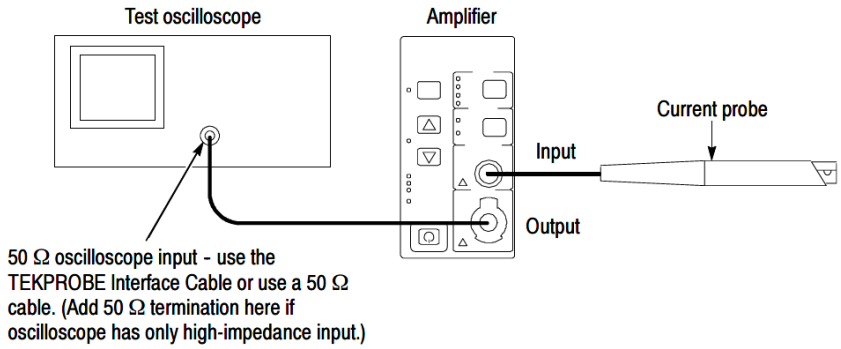

Figure 3.7: Current probe

In order to visualize correctly the current measurement on the oscilloscope, it has to be set with a 50Ω of input impedance and also once that the current probe is connected to the oscilloscope it is necessary to push the button of the 'autobalance' present on the current-probe amplifier in order to remove any residual magnetization that otherwise could introduce an offset in the measurements.

Below it is reported an example of current probe connected to the round cable

of a power supply:

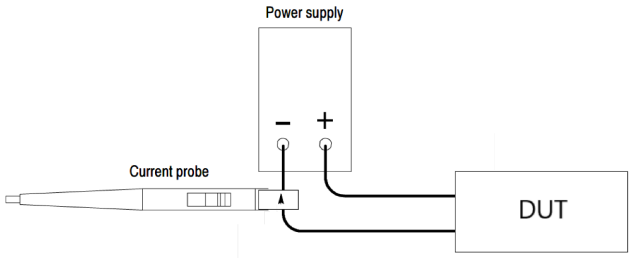

Figure 3.8: Current probe and DUT

2. OHM'S LAW : $I = \frac{V}{R}$

   It simply states that the current flowing through a resistor is directly proportional to the ratio between the voltage applied and the resistor.

   In this way, inserting a resistor $R = 1\Omega$ we have a 1:1 correspondence between the current and the voltage measured from the oscilloscope. [10]

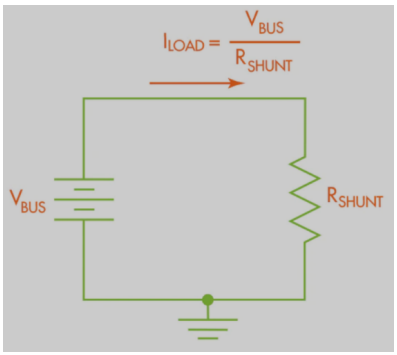

Figure 3.9: Indirect measure circuit

Surely the current probe is the one that gives the most accurate results between these two methods, for this reason it is chosen to use the current probe to measure the current on the core domain.

Chapter 4

Test flow variants

There are different parameters that impact the behavior of the system and these are in particular the temperature, the level of the voltage supply and the operating frequency.

At extreme conditions, so near to the maximum/minimum levels acceptable by the DUT the risk of failures is higher and moreover with a high temperature, a high voltage supply and a high operating frequency the DUT may overheat and this could cause the IC and the board to merge. While with a low voltage supply and low operating frequency the danger is that the system has no sufficient energy to start the operations.

It is important to test the DUT under these extreme conditions because it could behave differently from what expected and this could cause failures.

In this paragraph it is analyzed the behavior of the system subjected at high temperature (125°C), which is 100°C more than the room temperature and it is also measured its behaviour under the combination of high temperature and maximum voltage supply applicable.

Furthermore it is analysed the response of the DUT with different operating frequencies.

4.1 Temperature and voltage supply

To increase the temperature of the DUT until the desired value it is used the Thermonics Temperature Forcing System, which provides advanced temperature testing capabilities. [13]

Figure 4.1: Thermostream

This system can provide a temperature range of -90°C to 225°C in few seconds with an accuracy of ±1 °C, it delivers clean dry air at the board that is positioned at the end of the mechanical arm we see from Figure 4.1.

The board has sensors that allow to control the temperature on the device, in this way we have a precise information of the temperature at which we are submitting the device inside the socket. Once the probes of the oscilloscope and the MiniWiggler are connected to the board, it is positioned under the arm of the thermostream and we can start the measurements.

After the executions of all the test flow it is noticed that just Icore changes significantly with the temperature, while Ivddp (current on the 3.3V domain) remains

stable and this is the reason why in the next pages is shown only the analysis done on Icore (current on the core supply), in particular during the erase test.
During the execution of the erase test Icore has a particular step-shape that is independent from the temperature, while it is the level of the current that changes. Especially the current requested during the test increases with temperature. In the plot below we can see that if the temperature is set to 25°C (red curve) we have a certain level of current while if it is set to 125°C (green curve) the level increases, while the shape remains identical.

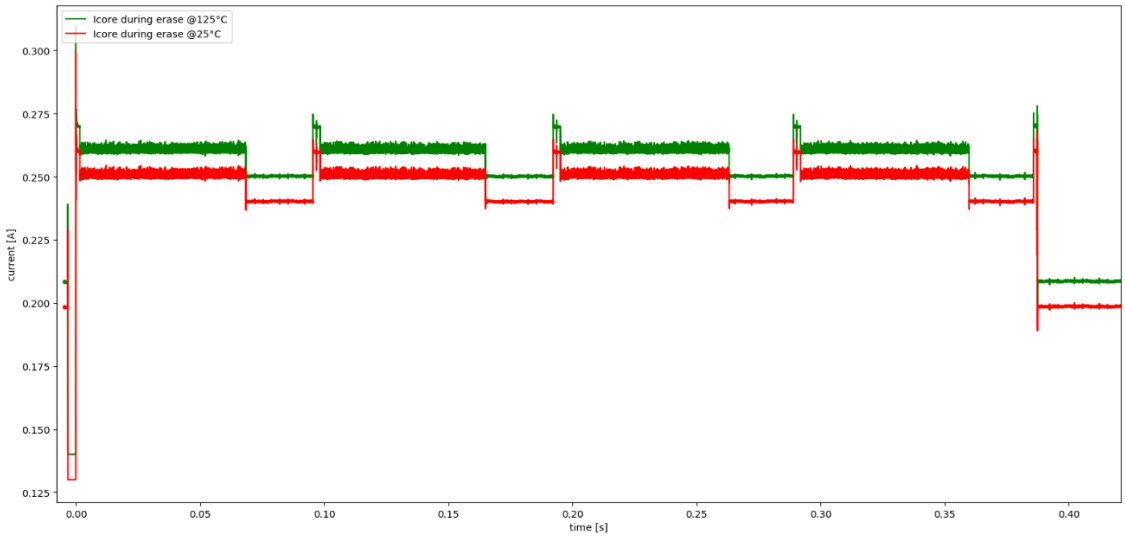

Figure 4.2: Icore at room temperature and at 125°C

Now it is better understandable that an increase in the temperature of 100°C brings to an increase in the current of about 9.5mA, in fact from the plot the two waveforms are exactly the same in terms of shape but there is just a shift between the two of about 9-10mA.
Keeping the high temperature T=125°C , it is repeated the measurement increasing the voltage supply from its nominal value: (1,25; 3,3; 5)V to the maximum value:

(1,36; 3,63; 5,5)V and what we observe is that during the test the current requested increases.

To better understand the differences between the waveforms in these two conditions (same temperature but different voltage supply), they are plotted together with Python and the result is:

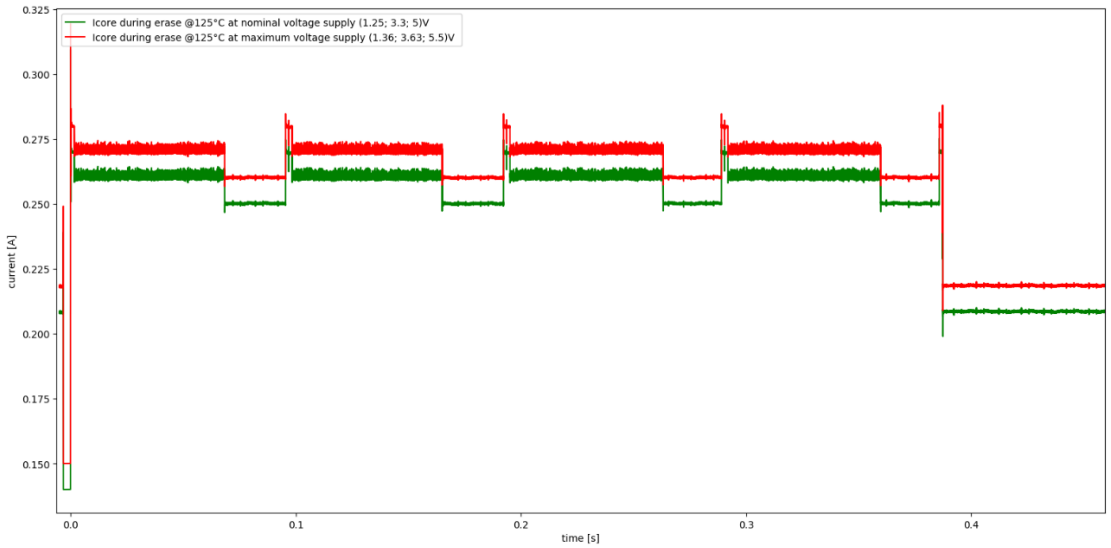

Figure 4.3: Comparison between Icore at different supply levels

As excepted an increasing in the voltage supply brings to an increase in the current of around 20mA for the Icore and 4mA for the current on the 3.3V domain, instead the current peaks remain of the same amplitude with temperature variations. These current variations are not the same for every device, because each family of devices acts differently if the boundary conditions are modified and to prove this statement is analysed also another device, the "2TC3Exx FF".

FF states for Fast-Fast, that means Fast-nmos and Fast-pmos. It is built with a technology for which both the transistors have the subthreshold voltage V_{th} that is lower with respect to the one of the normal transistors.

This is required for high-speed system, because if V_{th} is low the voltage required to switch-on the transistor is achieved in less time and so the time decreases but the drawback is that the leakages increase.

This is according to the formula:

$$I_{th} = CONST \cdot Vt^2 \cdot e^{\frac{Vgs-Vth}{m \cdot Vt}} \cdot (1 - e^{\frac{-Vds}{Vt}})$$

The leakage contributions are very effective for memories, because when we select a cell to be read; erased or programmed there is a huge number of other cells that are inactive and their leakage contributions are added to the current needed by the selected cell. The final effect is an increasing in the overall current requested.

Summarizing the effect of changing V_{th} in these two graphs, it is possible to see that decreasing V_{th} the delay decreases (first graph) while Ioff, which is the current when Vgs=0V, increases (second graph):

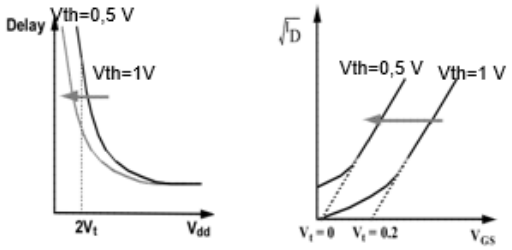

Figure 4.4: V_{th} dependencies

In the FF chip the current requested during the execution of the erase test is $\sim 100mA$ bigger with respect the one requested by the normal device due to these extra leakages contributions.

Moreover repeating the measurements of Icore with the nominal (1.25; 3.3; 5)V and the highest eligible value (1.36; 3.63; 5.5)V for the power supply, we measure an

increasing in the current level as observed also for the normal chip, but this time the induced variation is bigger.

The difference between the two current waveforms at nominal and maximum voltage supplies is better highlighted with this Python plot:

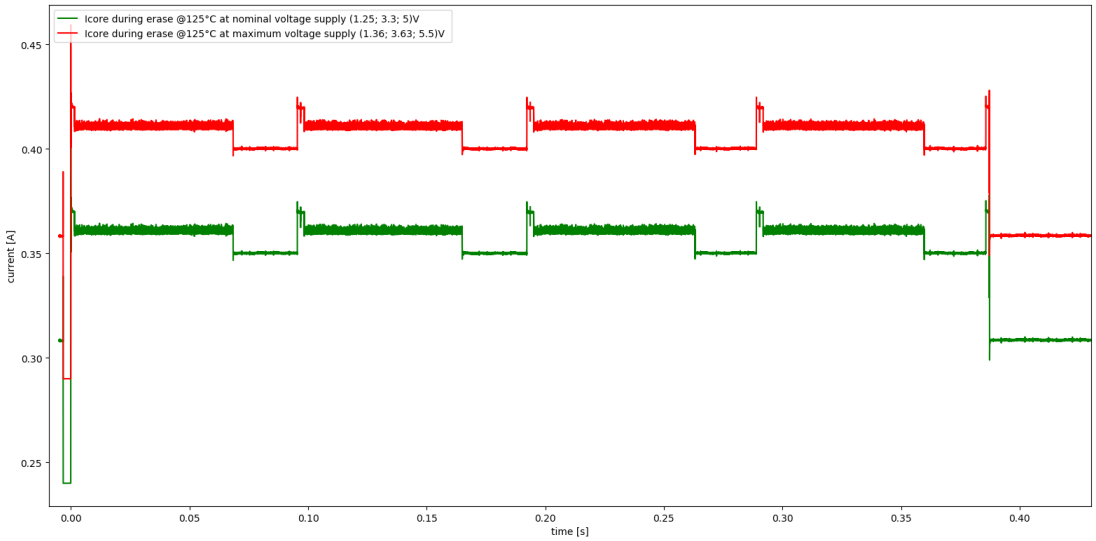

Figure 4.5: Icore during erase test for FF device at different supply values

As expected the 2TC3Exx FF acts differently with respect the TC38EVOx, in fact for the TCx38evo a voltage variation brings to a current variations of 13mA while for the FF device the current variation is about 50 mA.

In conclusion, as expected, increasing the temperature the current requested during test execution increases and so the power.

Moreover, the FF-chips with respect to the normal are faster but consumes more power, in fact if we focus on the level of current demand, we see that the normal chip with high temperature and a nominal supply voltage asks for 260mA, instead the FF for 360mA. This is due the leakages contributions, as explained before.

4.2 Operating frequency

The Phase Locked Loop (PLL) is an hardware circuit embedded in the DUT which function is generated an output frequency (f_{OUT}) whose phase is related to the phase of the input frequency (f_{IN}). The f_{IN} is the one of the internal oscillator equal to 20MHz and from it we can derive the wanted operating frequency by changing the value of the K2-Divider. This has a direct coupling to the power consumption of the device, therefore this must be done carefully.
The output frequency of the PLL is given by this formula:

$$f_{OUT} = \frac{N \cdot f_{IN}}{P \cdot K2}$$

Where f_{IN} = 20MHz.
For example if we want the output frequency from the PLL equal to 180Mhz the right values to set are: P = 02 , K = 03 and N = 36 . These are the values in hexadecimal, which correspondingly are 2, 3 and 54 in decimal.

$$f_{OUT} = 180MHz = \frac{54 \cdot f_{IN}(=20MHz)}{2 \cdot 3}$$

To communicate the wanted frequency at the system, it has to be written into this register: "0x7000090C" the value we want, that in this example is: "0x36020302" because N = 0x36; P = 0x02 and K = 0x03. The resulting test code that we have inserted in JAZZ is:

RW32, 0x7000090C, 0x36020302 , 0xFFFFFFFF

And these are the results:

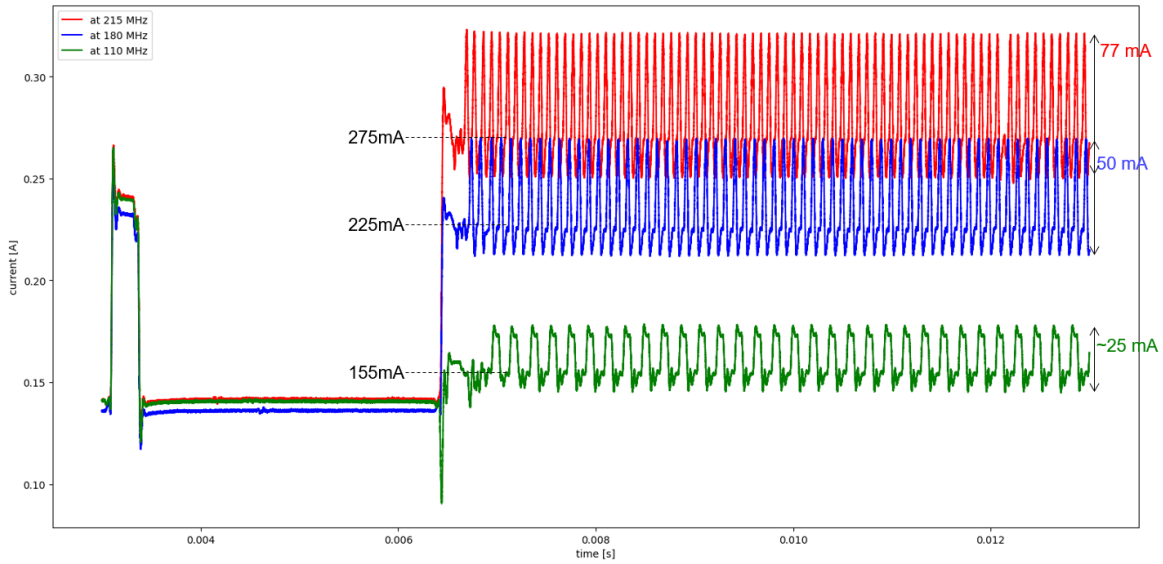

Figure 4.6: Icore during verify test at different operating frequency

As it is possible to see:

f_{PLL}	110MHz	180MHz	215MHz
Icore nominal	115mA	225mA	275mA
Icore peak	25mA	50mA	77mA

Table 4.1: Icore values changing f_{PLL}

Chapter 5

NVM tests characterisation and power estimation

Every memory tests is characterized by a certain level of current and in some tests are present peaks superimposed at this level. To understand what generates these peaks it is investigated inside the test codes. The codes are written in C and they are part of a project developed by Infineon Technologies AG. Once all the tests are compiled, they are built together to generate the executable (file.hex) that is a machine code file that will be then uploaded in the JAZZ flow in one of the first test executed: devcfg_t_b_ta_ld.

The first thing is to understand at what the power peaks refer to and to do so, we have insert in the scripts some signals that toggle when it is executed that particular part of the code in order to make easier the debug process. These signals have been mapped on the board in the ANALOG VTP CONNECTOR and they are:

PIN	CONNECTOR	INSTRUCTION RISE
P20_14	X33(1)	P20_OUT.B.P14=0x1;
P20_11	X33(3)	P20_OUT.B.P11=0x1;
P11_2	X33(5)	P11_OUT.B.P2=0x1;
P11_12	X33(6)	P11_OUT.B.P12=0x1;
P11_11	X33(11)	P11_OUT.B.P11=0x1;
P20_8	X33(16)	P20_OUT.B.P8=0x1;
P11_3	X33(17)	P11_OUT.B.P3=0x1;
P11_6	X33(19)	P11_OUT.B.P6=0x1;
P11_12	X33(20)	P11_OUT.B.P12=0x1;
P11_9	X33(21)	P11_OUT.B.P9=0x1;
P20_9	X33(22)	P20_OUT.B.P9=0x1;
P11_10	X33(23)	P11_OUT.B.P10=0x1;

Figure 5.1: Pin map

The probes can be connected at every pin of the ANALOG VTP CONNECTOR, which has this structure on the board:

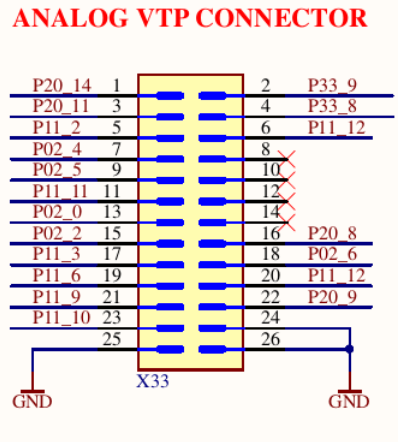

Figure 5.2: Analog VTP connector

The signals are added in the scripts of the erase, verify and program test. Thanks to these modifications in the test code, it is possible to characterise every test, and even if the main target of this project is the verify test, also the erase and program test have been characterised. In the table below it is reported the test time of each of them:

	Verify test	**Erase test**	**Program test**
test time	34 ms	384 ms	5,3 s

Table 5.1: Test time of erase, verify, program test

Now it is shown the current profile requested during the each test execution.

- ERASE TEST

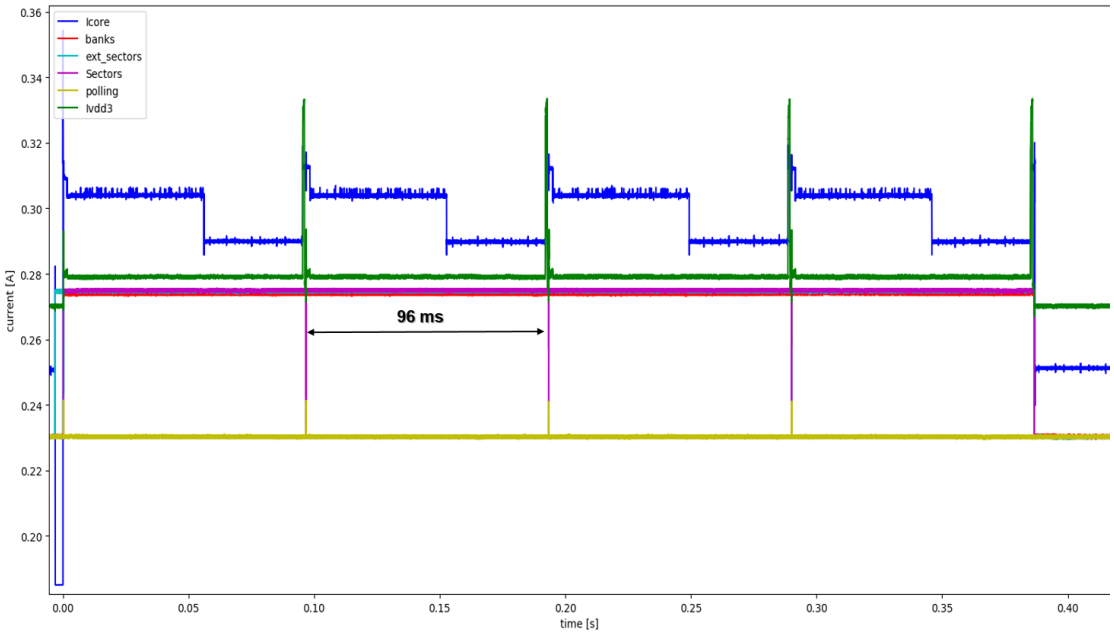

Figure 5.3: Structure of the erase test

The erase test is organized in four parts, each part corresponds to a bank which is erased separately. As seen from the image to erase a bank are spent 96 ms. Every 96 ms there is a jump in the current Icore (blue curve) and a quicker and more sudden jump on 3.3V domain (green curve). These jumps are due to the operations of the arithmetic circuits inside the microprocessor that are needed for switching from one bank to another.

If it is considered the current during the execution of the erase itself, so from

when it is selected a bank until it is completely erased, it is possible to see that the current on the 3.3V do not have variations while the Icore has a stepped shape.

There are three different values for the current Icore, at the begging a narrow pick that reaches 320mA, then a value of 300mA that lasts for 50ms and then the current returns to its nominal value (287mA).

- PROGRAM TEST

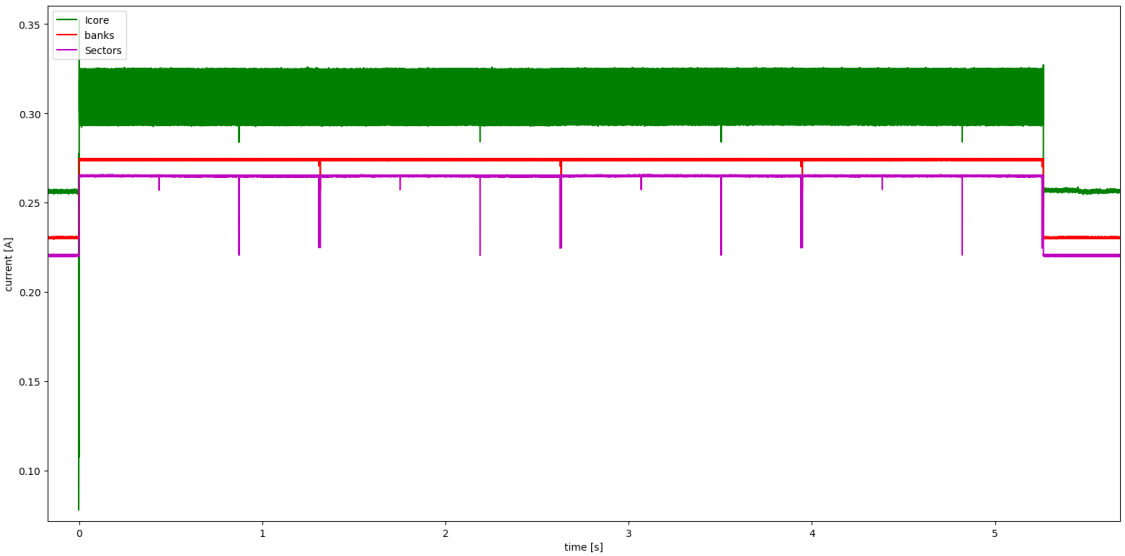

Figure 5.4: Structure of the program test

The program test lasts for 5.3s and principally it is organized such that it is firstly selected the bank and then the sector that has to be programmed, in particular there are three sectors for each bank. Once that three sectors of a banks have been programmed it is switched bank till all the memory has been programmed.

In the figure above the peaks of the red and the violet curves should have the

same amplitude, however even with the maximum sampling frequency set on the oscilloscope, the instrument was not able to sample correctly all the peaks in the curves, because they toggle to fast, but in reality they have the same amplitude.

If it is zoomed on the core current curve (green), it is possible to analyze its shape. It is characterized by giving peaks of 24mA each every 41ms.

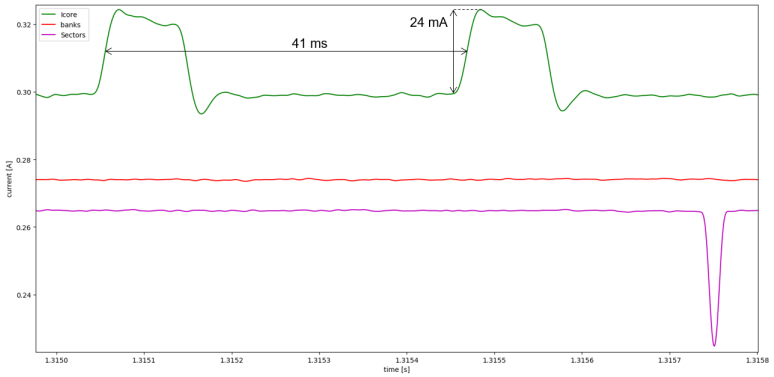

Figure 5.5: Zoom into the program test

- VERIFY TEST

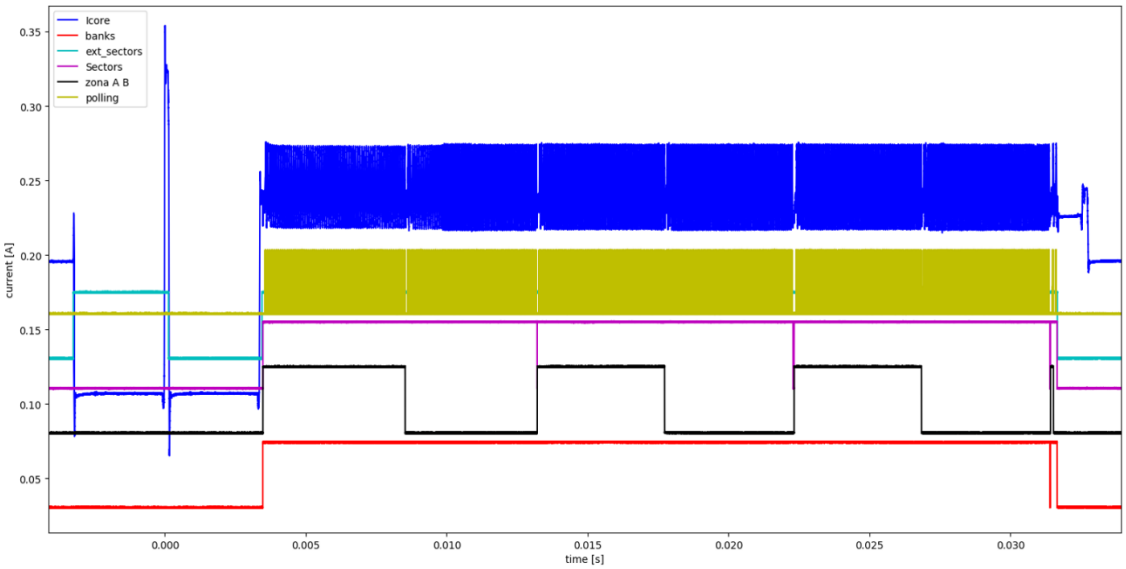

Figure 5.6: Structure of the verify test

The verify test is organized such that firstly it is performed the access to the sector, then to the zone and then to the minisector and it is started the verify process.

After that 66 minisectors are verified, it is possible to switch zone and then when the two zones of a sector are verified it is possible to switch also sector. The blue curve is the Icore, which has an initial peak of about 200mA due to the locking of the PLL and then other peaks of 77mA each. Counting them with Python the result is that for every zone there are 66 peaks and then two final peaks that corresponds to the part of the memory for the redundancy. Every peak of 77mA corresponds to the verify of a minisector.

If we zoom inside the verify test, considering Icore and the polling signal we have this shape:

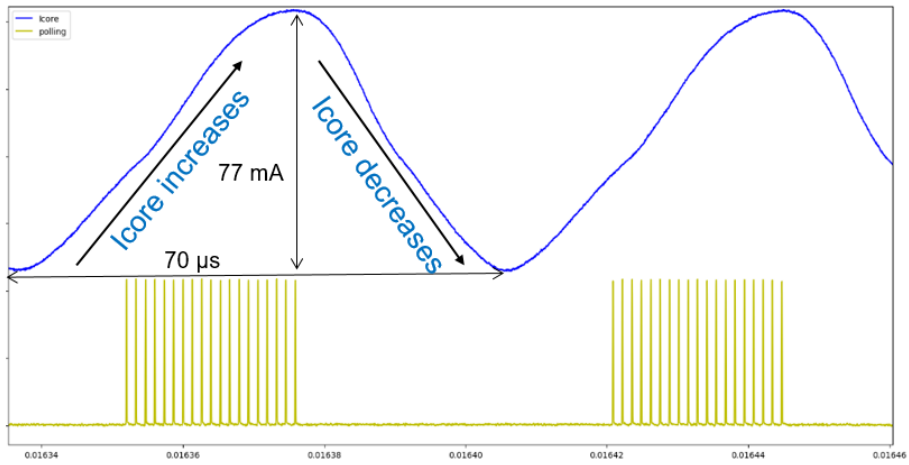

Figure 5.7: Zoom into the verify of two minisectors

From the measurements, it seems to be a strong correlation between Icore and the polling activity because Icore increases when the verify of a minisector begins and it reaches its maximum when the polling has finished, and so the test.

Polling is an operation in which it is repeatedly checked into a register the status of an operation, this means accessing the register several times and reading a bit written in it. This activity lasts until in this register is not read that the operation has finished correctly.

Every time that a register is read, it is called the function "ftlib_sfr_read()" which in turn invokes other functions, this requires a significant number of context switches. With the debugger it is analysed deeper the test code and it is discovered that there are three main functions that are called an infinitive numbers of times and in turn they require to invoke other 4/5 functions to be executed. For debugging it is used Universal Debug Engine (UDE) 5.0.

The three functions are: *ftlib_sfr_read()* and *ftlib_sfr_verify()* and *ftlib_sfr_write()*

which simply read, verify and write respectively at a certain specified address.
From here, we started to do the main hypothesis that the power spikes come from the fact that calling a function so often causes very fast and repeated context switches and this require extra time and power. The following three software solutions found to decrease the amount of power consumption are based on the above assumption.

- Reducing the frequency of the **polling activity**.

  When we want to monitor if a certain operation has finished, frequent accesses to a register are performed. Decreasing the number of times in which it is asking if a certain operation has finished also the current requested by the DUT decreases. More it is delayed the polling activity and lower is the amount of current requested

- Using a new version of **ftlib_sfr_write()** function.

  This is a function that simply writes in a specified address of a memory. Inside its code there are other functions which have only the scope of monitoring if the writing operation has finished correctly or not, but have no others functionalities. They in turns invoke other functions, if they are not executed there are less context switches and we save a certain amount of power

- Using a specific low level function named **MEM8()** by Infineon.

  It directly loads/stores into memory at a specified address. With this solution all the contest switched are bypassed and so a great amount of power is saved.

All these three techniques are explained in detail below.

5.1 Reduction of the frequency of the polling activity

A reduction in the polling activity means a reduction in the frequency at which we control the "ready-bit" inside a register that tell us if the operation we were controlled is concluded or not. To decrease this activity it is inserted a *for loop* in the code. In this section two analysis are reported: one on the verify and one on the erase test.

Starting with the analysis on the **verify test**, it is delayed the polling activity that checks if the verify of a minisector has ended or not.

In the previous paragraph we have seen that to verify every minisector, 77mA are required and also thanks to the probes inserted in the test code we have noticed that the maximum of 77mA is in proximity of the end of the polling activity. As it is shown in Figure 5.7 . The part of the test code for the verify in which the delay of the polling has been inserted is in in the *"ftlib_verify_sse_mnsec.c"* and in particular in the branch *else*:

```
while ( TRUE == continueLoop )
{

    P11_OUT.B.P6=0x1;
    P11_OUT.B.P6=0x0;

    ftlib_sfr_read( SFR_COMM_1, &comm1contents );


    if ( FTLIB_VERIFY_SSE_HALTED_HANDSHAKE == ( FTLIB_VERIFY_SSE_HALTED_HANDSHAKE && comm1contents ) )
    {
        /* ASB read back */
        FTLIB_ASSEMBLY_BUFFER_STORE( pbuffer, FTLIB_ASSEMBLY_BUFFER_SIZE );
        ftlib_sfr_write( SFR_COMM1, &param1 );
        ftlib_sfr_write( SFR_CCTRL, &param2 );
        processSseReadout( pparams, presults, pbuffer );
    }

    /* check if SSE is over */
    if ( ftlib_sfr_verify( SFR_COMM_2 , &endCommand ) == TRUE )
    {
        continueLoop = FALSE;
    }
    else
    {
        /* nothing to do */

    }
}
```

Figure 5.8: ftlib_verify_sse_mnsec.c with polling delay added into the TC

Depending on how much the polling is delayed, the current consumption changes, more the polling is delayed and lower is the current requested.

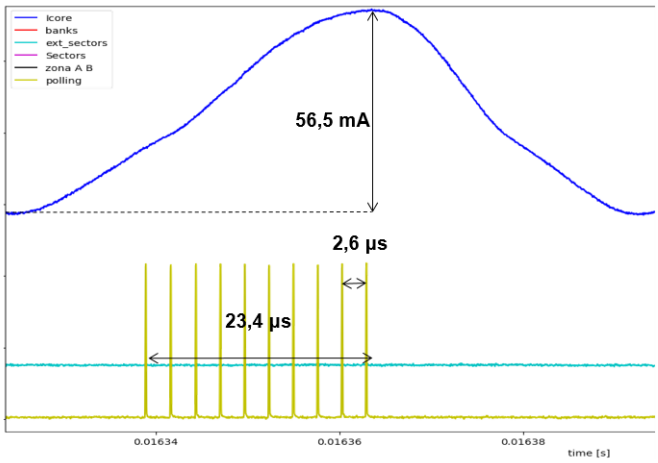

Figure 5.9: Icore during the verify of two minisectors with polling delayed of 100

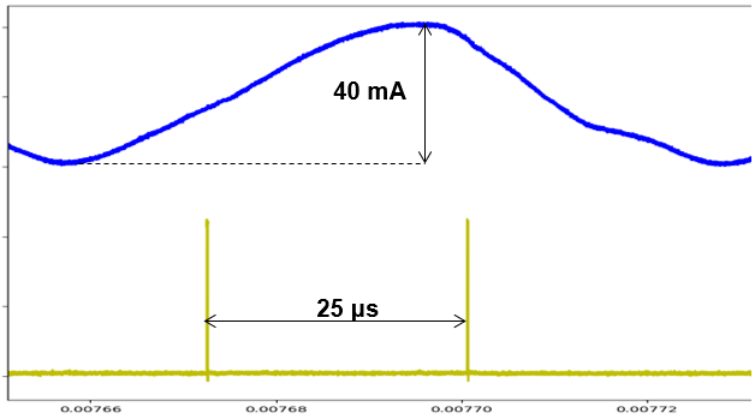

Figure 5.10: Icore during the verify of two minisectors with polling delayed of 2000

As expected with a delay of 100,the current decreases from 77mA to 56,5mA, so there are 20,5mA of difference, instead with a delay of 2000 the difference is 22mA.

The analysis is repeated for the **erase test**. During the erase test it is enabled the Pump Monitor, which monitors the states of the pumps to evaluate if faults have occurred. For example if the pumps are working on their 100% of capacity means that a short circuit is present in the system. There is a polling to understand if this action has finished or it is still running, in the file "*ftlib_pump.c*", and this polling activity is delayed as shown below:

```
VOID ftlib_pump_run(
        UINT32 address,
        UINT32 mask,
        UINT32 value )
{
    register UINT32 running = 0;
    FTLib_PumpFunction_p pmeasure = NULL;


    /* Initialize Pump Monitor function pointer */
    pmeasure = FTLib_PumpParameters.psetting;

    /* Polling execution term */
    do
    {
        if ( pmeasure != NULL )
        {
            /* Call measure function */
            pmeasure[FTLib_PumpParameters.setting].pfunction();
        }
        UINT32 time;
        running = FTLIB_PUMP_POLL( address, mask );
        for (int i = 0 ; i<500; i++) ; //debug

    } while ( running != value );

    return;
}
```

Figure 5.11: Ftlib_pump_run()

The polling in this case is executed to evaluate if the pump monitor has finished to monitor the pumps. Delaying the polling activity it is observed a decreasing in the Icore requested and also a change in its shape, moreover more the polling is delayed and lower is the current requested.

In the image below are plotted three curves of Icore, one is the original one(green curve), one is measured with a polling delayed of 100 (red curve) and one with a polling delayed of 500 (yellow curve).

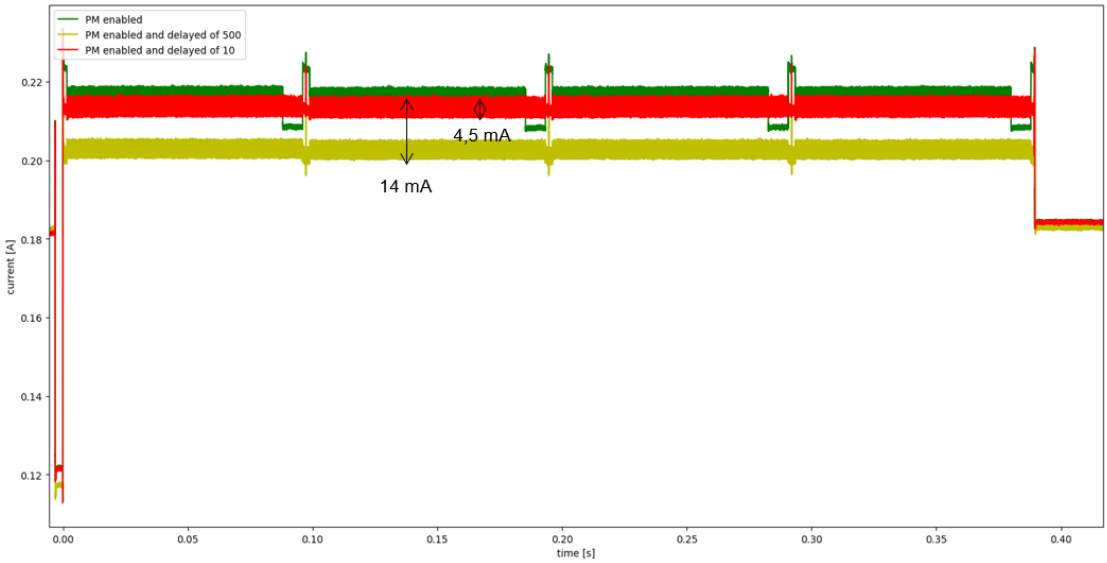

Figure 5.12: Impact of polling delayed on the Icore during the erase test

Also for the erase test, more the polling is delayed and better it is in terms of power consumption because the current decrease.

5.2 Modified version of ftlib_sfr_write()

Another finding to reduce the power consumed is that by changing just the code of the *ftlib_sfr_write()* function it is saved a bit of power. This function it is called many times during the execution of the test code, so it is important try to understand how to decrease the power that it requested to be executed. Looking inside the function code it is possible to notice that there are two functions: *FTLIB_SFR_TRACE_ENABLE(sfr)* and *FTLIB_SFR_TRACE_DISABLE()* that are used just for a debug purpose, because they are needed to understand if it has been written or not inside the sfr-registers.

The new *ftlib_sfr_write()* function is:

```
inline VOID ftlib_sfr_write(
        UINT32 sfr,
        UINT32 mask,
        UINT32 bitShift,
        PUINT32 pvalue )
{
    UINT32 sfr_value = 0;
    UINT32 address = 0;


    //FTLIB_SFR_TRACE_ENABLE( sfr );  // commented

    address = calculateAddress( sfr );
    shiftValue( &sfr_value, pvalue, bitShift );

    /* Write register */
    ftlib_register_sfr_write( address, mask, &sfr_value );

    //FTLIB_SFR_TRACE_DISABLE(); // commented

    return;
}
```

Figure 5.13: New test code of the ftlib_sfr_write()

Doing that, without substituting anything else in the code but just doing this changing in the *ftlib_sfr_write()* there is a reduction in the current requested during the execution of all the tests.

- **Erase test**

As it possible to see in the figure below, the current decreases of 8mA, a

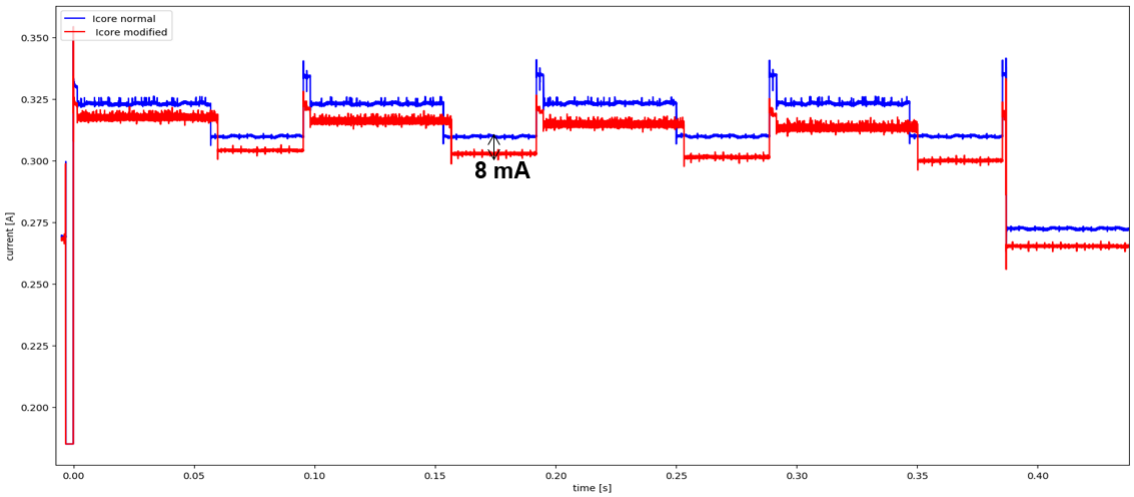

Figure 5.14: Icore during erase test with new version of ftlib_sfr_write()

- **Verify test**

  Maintaining the usage of *ftlib_sfr_veriy()*, *ftlib_sfr_read()* but with the new version of *ftlib_sfr_write()* the result is:

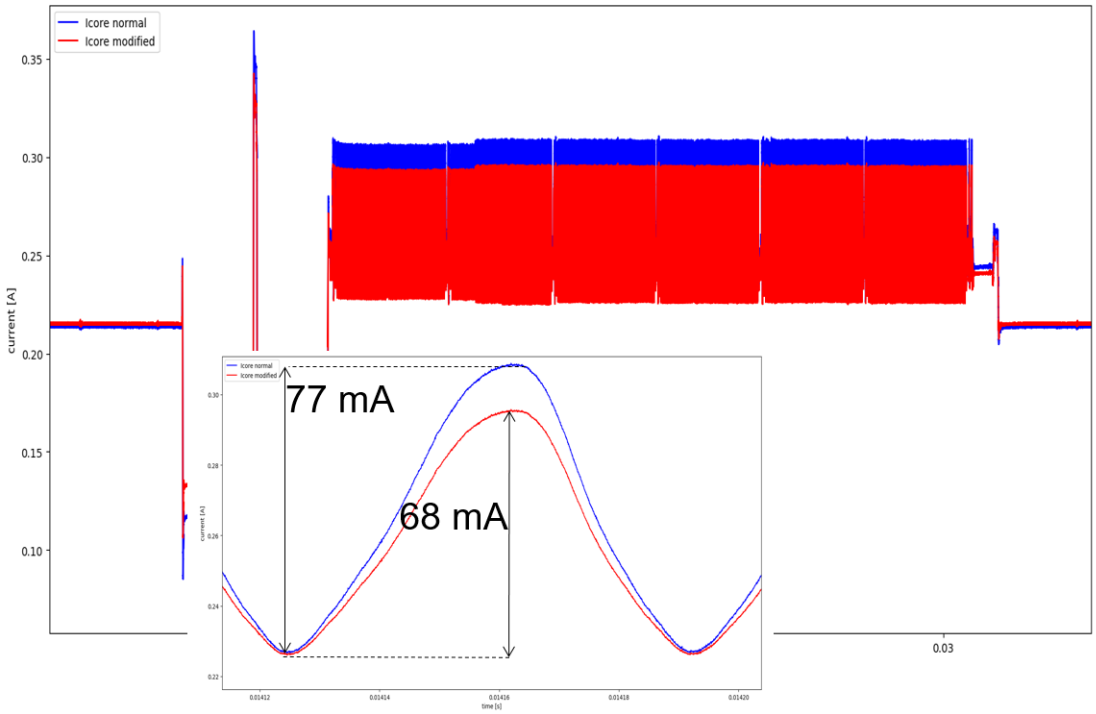

Figure 5.15: Icore during verify test with new version of ftlib_sfr_write()

5.3 Directly load/store into the memory using MEM8()

As it is explained before, during the test execution there are few functions called several times that simply write, read or verify a certain location in memory and correspondingly they are *ftlib_sfr_write()* , *ftlib_sfr_read()* and *ftlib_sfr_verify()*.
Every time one of these function is called, in turn they call others 4/5 functions and this requires extra time and power. These functions can be substituted with MEM8(), which is a function that Infineon Technology has implemented with the aim of loading/storing instruction depending if it is needed to read or write into a memory location.
At the beginning to study which is the effect of this substitution it is done an experiment just on the verify test and in particular it is performed a substitution in the *ftlib_verify_sse_mnsec.c* code as shown below:

```
/* poll for command completion */
while ( TRUE == continueLoop )
{


    P11_OUT.B.P6=0x1;
    P11_OUT.B.P6=0x0;

    //ftlib_sfr_read( SFR_COMM_1, &comm1contents ); //translated into a inline function
    comm1contents= MEM8(0xf8030004);               //replace previous instruction


    if ( FTLIB_VERIFY_SSE_HALTED_HANDSHAKE == ( FTLIB_VERIFY_SSE_HALTED_HANDSHAKE && comm1contents ) )
    {
        /* ASB read back */
        FTLIB_ASSEMBLY_BUFFER_STORE( pbuffer, FTLIB_ASSEMBLY_BUFFER_SIZE );
        ftlib_sfr_write( SFR_COMM1, &param1 );
        ftlib_sfr_write( SFR_CCTRL, &param2 );
        processSseReadout( pparams, presults, pbuffer );
    }

    /* check if SSE is over */
    //if ( ftlib_sfr_verify( SFR_COMM_2 , &endCommand ) == TRUE )
    if ( MEM8(0xf8030005) == FTLIB_VERIFY_SSE_TM_COMMAND )  // replace previous instruction
    {
        continueLoop = FALSE;
    }
    else
    {
        /* nothing to do */

    }
}
```

Figure 5.16: ftlib_verify_sse_mnsec.c

Where:

ftlib_sfr_read(SFR_COMM_1, &comm1contents);

is substituted with

comm1contents= MEM8(0xf8030004);

and

ftlib_sfr_verify(SFR_COMM_2, &endCommand)==TRUE

with

MEM8(0xf8030005) == FTLIB_VERIFY_SSE_TM_COMMAND

To explain how to calculate the address that must be specified inside the MEM8() function it is reported the example of the ftlib_sfr_read(SFR_COMM_1, &comm1contents) that it is used in the test code reported above. This functions is substituted with comm1contents = MEM8(0xf8030004); where the address "0xf8030004" has been calculated as specified in the file "ftlib_sfr_tc3xx.h":

#define SFR_COMM_1 4, 0xFF, 0

In this example the address of SFR_COMM_1 is 4 in hexadecimal masked with 0xFF, therefore it is not specified in the code because it is a mask of all 1s.

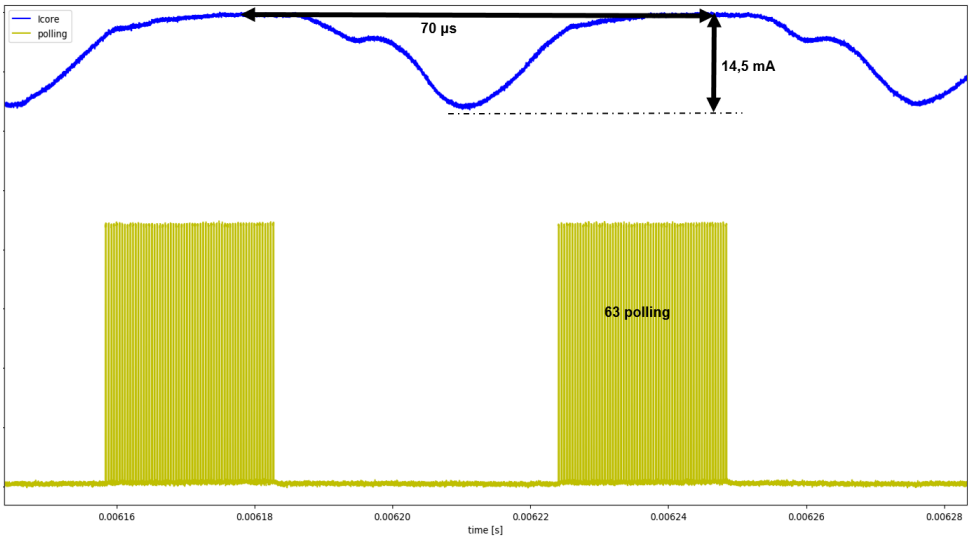

Figure 5.17: Icore(blue) during the verify of two mnsec using MEM8()

To see if mixing the usage of MEM8() and delaying the polling activity there is a reduction in the current consumption it is inserted a "for loop" with 2000 iterations inside the *else* branch.

The final result is the one reported below:

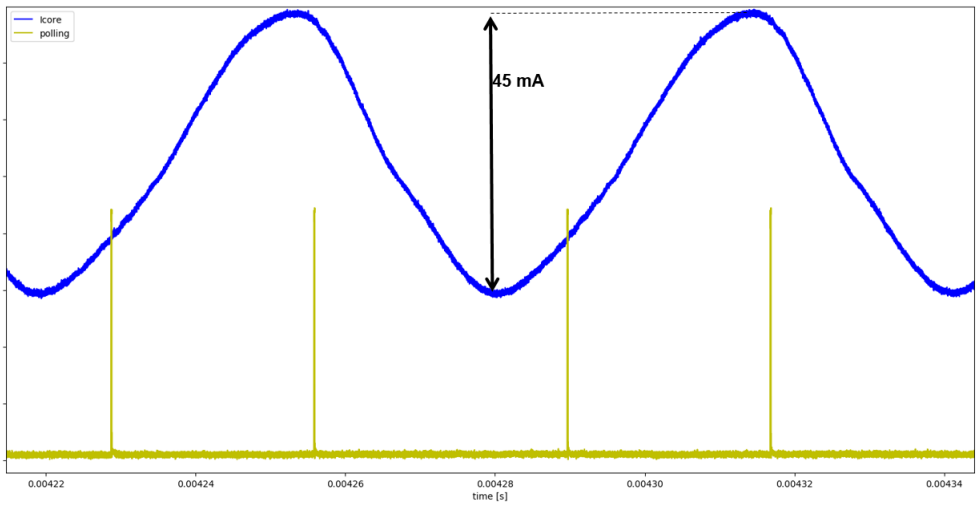

Figure 5.18: Icore during the verify of two mnsec using MEM8() with delay of 2000

Unfortunately, combining the delay of the polling activity and MEM8(), the current does not decrease but instead increases. This can be explained by for delaying the polling we have to insert a "*for-loop*" and this requires that the ALU increases by one the variable of the loop, and so this becomes the main contribution of the current requested.

So at the end, it is verified that using MEM8() brings to a strong reduction in the current request during the verify test and to prove that it is beneficial for all the tests, it is created a new project in Tasking in which all the functions *ftlib_sfr_read()* and *ftlib_sfr_verify()* are substituted with MEM8() specifying the proper addresses and masks. It is important to highlight that for every ftlib_sfr function substituted we must specify the right address and mask, but this makes the code no more reusable for all the families of products because every family has its own addresses.

Once all the ftlib_sfr functions are substituted it is built another file.hex which is then updated in the JAZZ flow. With this new executable all the tests are measured again and it is done a comparison with the previous measures. For the erase, the verify and the program tests there is a variation in terms of current Icore and below are reported the results.

Erase test

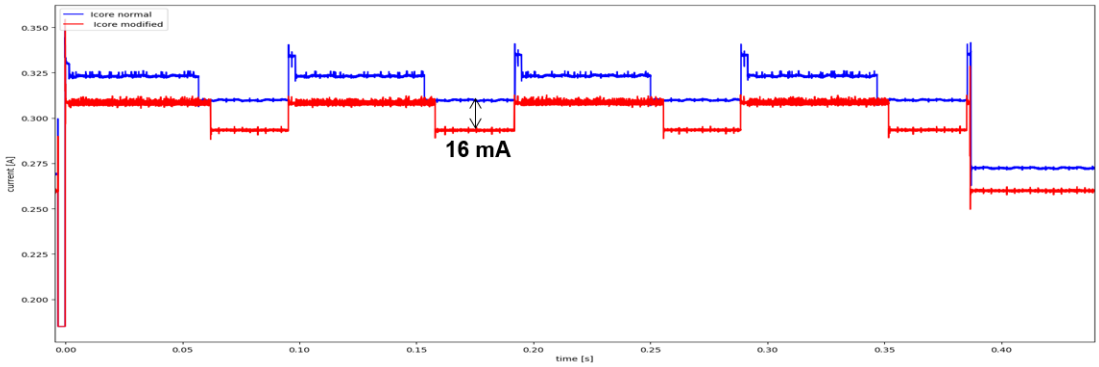

Figure 5.19: Erase test with MEM8()

The current Icore decreases of 16mA if it used MEM8() instead of the ftlib_sfr functions and moreover it seems that the high level of the Icore-step lasts for more.

Verify test

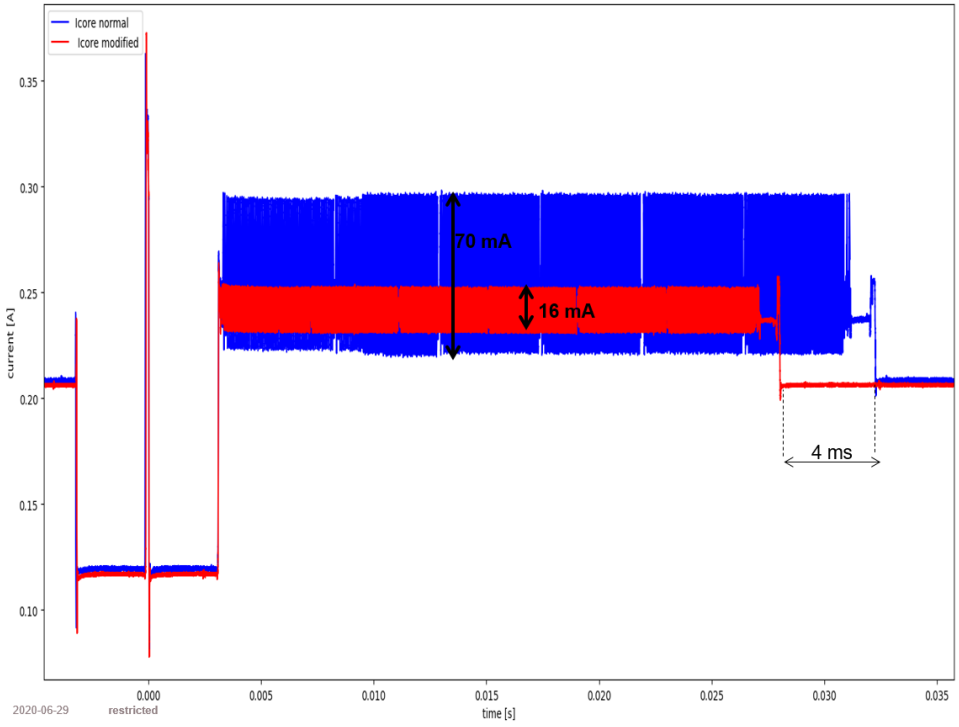

Figure 5.20: Verify test with MEM8()

The current decreases from 77mA to 14,5mA as shown before and also the test time is reduced. This delay is present in every test where we have substituting the sfr_functions with MEM8(), but in the verify is more evident because it seems that in this test there are more sfr_functions with respect to the other tests. The time is reduced due to the function MEM8() lasts less than ftlib_sfr functions, as reported below:

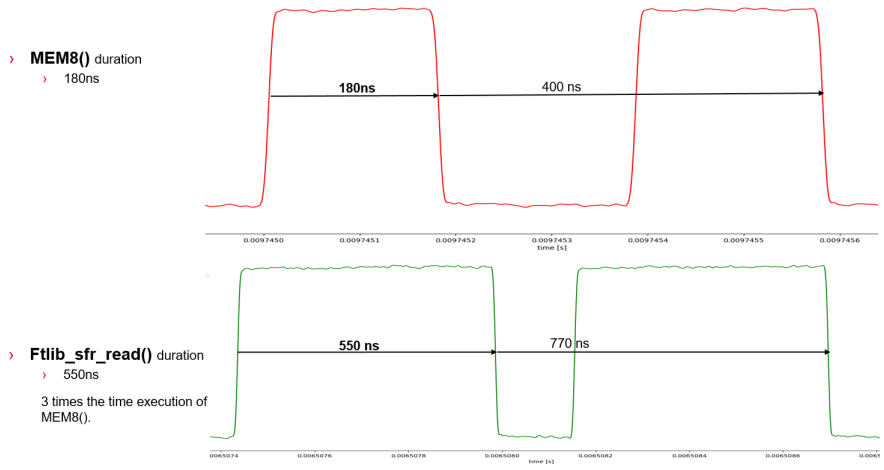

Figure 5.21: MEM8() VS ftlib_sfr_read() duration

Program test

In the program test there is a reduction of about 14mA.

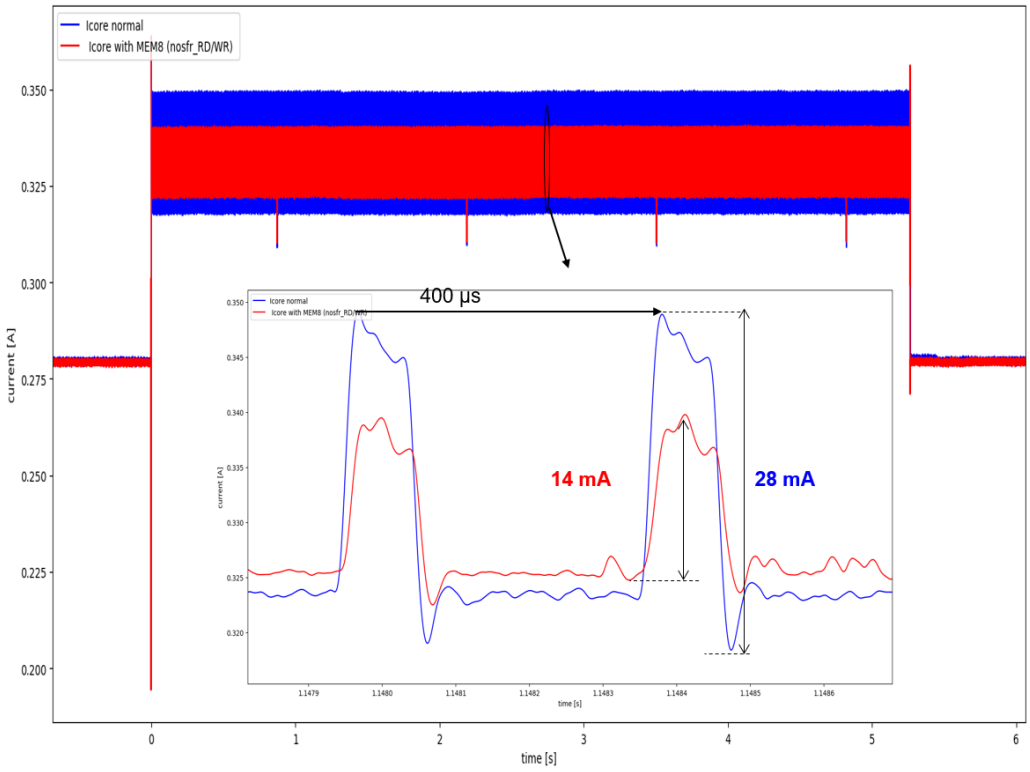

Figure 5.22: Program test with MEM8()

In conclusion the use of MEM8() is convenient for all the tests, and for some of them there is also a test time reduction. This is explained by the fact that using directly MEM8() several calling functions are avoided and so the context switches, this brings to a reduction in the current requested and also in the test time execution. The gain in terms of current saved of all the exposed techniques are summarized in this table:

	Verify test (of a mnsec)	**Erase test**
Technique:	Original Icore level	Original Icore level
Delayed polling	-20,5mA	-9mA
MEM8()	-62,5mA	-16mA
New ftlib_sfr_write()	-9mA	-8mA

Table 5.2: Results

5.4 Analysis on the portability of the test code

The test codes in Tasking are organized such that they are portable on any family of devices and this is an advantage because the skeleton of the project is the same for every DUT and what it is changed is just the file.h where the addresses of the single device are specified. The file *ftos_fsist_tricore.h* includes all the families usable, as reported in part below:

```
#elif defined( DERIVATIVE_TC33xAED )
    #include <ftos_fsist_tc33xAED.h>
#elif defined( DERIVATIVE_TC37xA_EDPD )
    #include <ftos_fsist_tc37xA.h>
#elif defined( DERIVATIVE_TC38xA )
    #include <ftos_fsist_tc38xA.h>
#elif defined( DERIVATIVE_TC3ExA )
    #include <ftos_fsist_tc3ExA.h>
#elif defined( DERIVATIVE_TC39x )
    #include <ftos_fsist_tc39x.h>
#elif defined( DERIVATIVE_TC39xB )
    #include <ftos_fsist_tc39xB.h>
#else
#endif
```

Figure 5.23: Choice of the right family device

The family of chip analyzed is the TC3ExA.

The system is portable thanks to the fact that every family of devices uses the same name for the variables that have the same function, every family of devices has a specific "file.h" where are defined these variables with their own values.

This solution is very attractive because has few advantages: high reuse and shorter time to market but the disadvantages is a higher energetic cost. An example of that is the use of MEM8(), in fact as shown in the section before if it is used the flexible structure of the system written in JAZZ the current is much higher with respect to the dedicated solution in which it is used the function MEM8().

The reason is that the usage of MEM8() requires to specify an exact address and mask and every family of device has its own addresses and masks whose values are

taken from file_family.h . If the goal is to maintain the portability of the code it must be use other solutions to combine flexibility and efficiency, and the alternative solutions found in this project are:

- Delaying the polling activity
- Modifying the *ftlib_sfr_write()* commenting the *ftlib_sfr_trace_anable()/disable()*
- Modifying the *ftlib_sfr_write()* commenting the *ftlib_sfr_trace_anable()/disable()* and using MEM8() just in the *ftlib_verify_sse_mnsec.c* code

With these methods the flexibility of code is maintained but in the same time also the power consumed by every test decreases.

Chapter 6

Voltage droops

As explained in the previous chapters, to evaluate the power consumption we have measured the current that flows from the power supply to the device. From the measurements it is possible to see how this current has big and quick variations, that are called jumps.

Until now we have assumed that the level of the voltage supply remains stable, but this is unreal because the current jumps present during the execution of tests cause voltage droops. The phenomena is that if the current required by the DUT suddenly increases, all the physical elements such as capacitances, inductances and resistances affect the voltage that supplies the device [12]. The effect on the power supply is better shown below:

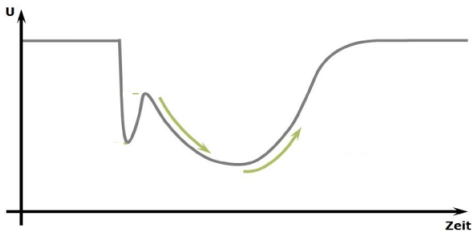

Figure 6.1: Voltage peaks due to current jumps

The droops on Vcore are dangerous because if the current jump is big, the corresponding voltage droop could exceed the acceptable dynamic and this could bring into a failure.[14]

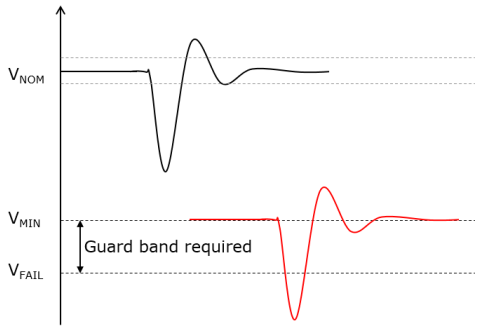

Figure 6.2: Vcore droops and possible fail

As it is possible to see from the image, if we set the nominal value for Vcore on the power supply and we have this kind of droop nothing happens. Instead if we set on the power supply the minimum value for Vcore and we have the same droop, the test will fail.

After having tested with few measurements that higher is the current jump and bigger is the correspondent voltage droop, we have verified that the techniques to reduce Icore explained in **Chapter 5** are valid also to decrease the voltage droops. To measure Vcore we have applied some modifications at the board, as shown in the figure below:

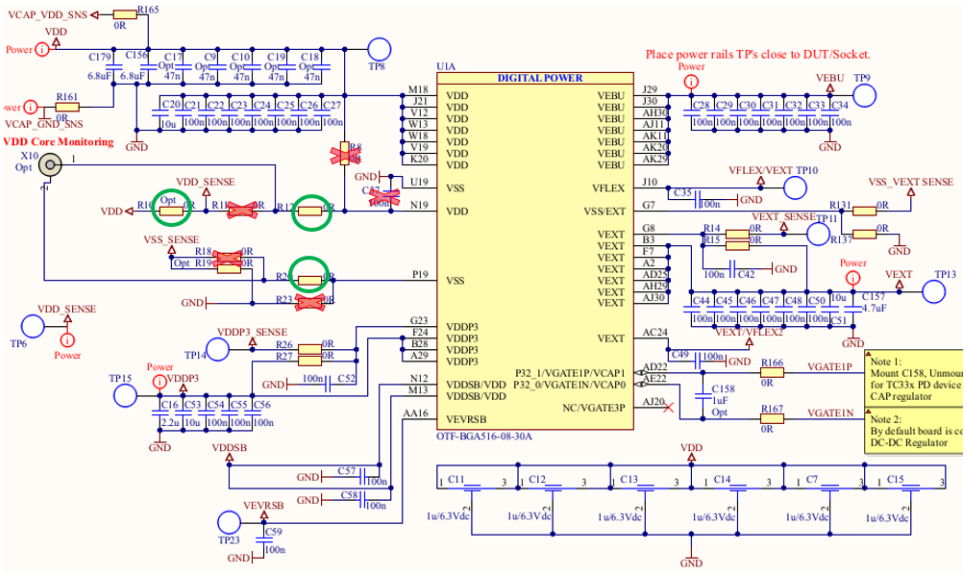

Figure 6.3: Board schematic

The resistance with the red crosses have been removed while the ones with the green circles have been soldered on the board.

6.1 Software solutions

As said before the current jumps are directly proportional to the voltage droops that we observed on the voltage, as a matter of fact the same techniques used to decrease Icore, have impact also on the droops of Vcore.

To prove that, it is reported the comparison between the verify test performed with the original test code versus the one performed with the test code with MEM8(). As excepted, considering the most effective low-power solution that consists in substituting *ftlib_sfr_read()* and *ftlib_sfr_verify()* with MEM8() we have a reduction of the Vcore droops, as shown in the next figure:

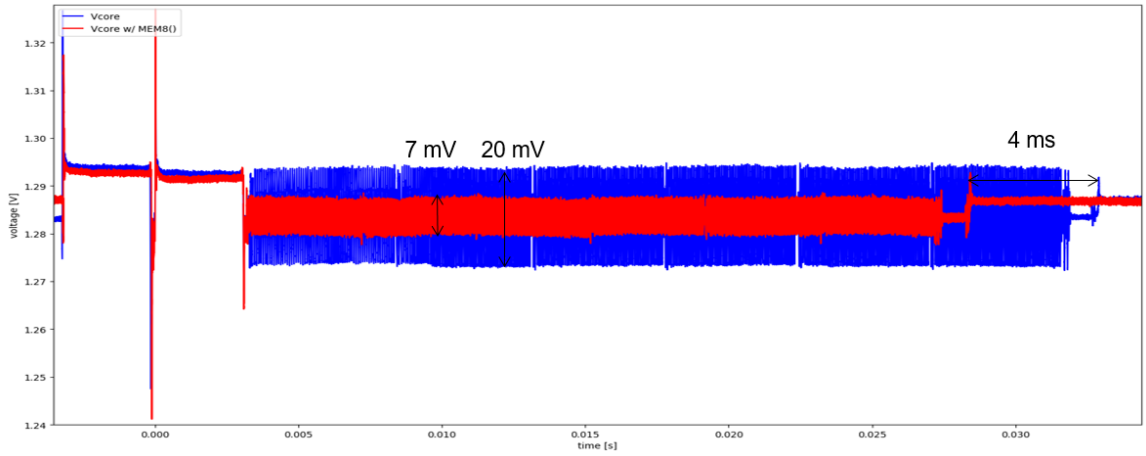

Figure 6.4: Vcore reduction during the verify test with the original test code (blue) and with the modified one(red)

The voltage peak for verifying a minisector decreases from 20mV to 7mV each and there is a decrease also in the test time, about 4ms.

Zooming on each minisector we see that:

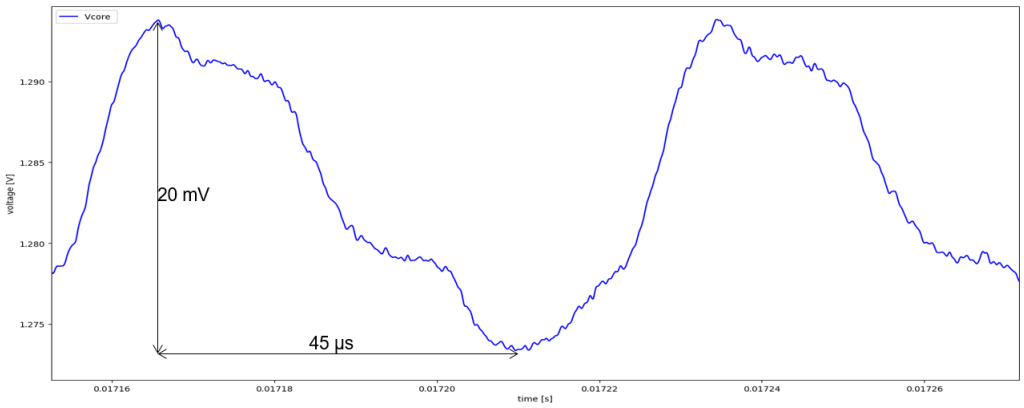

Figure 6.5: Vcore during the verify of a mnsec

While using directly MEM8():

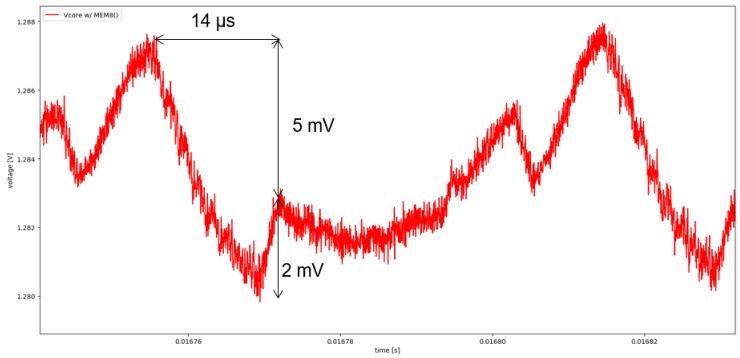

Figure 6.6: Vcore during the verify of a mnsec using directly MEM8()

6.2 Hardware solutions

Aside of acting via software there are other hardware possibilities to decrease the voltage droops, in particular we can act on the board itself changing the way in which it is supplied and soldered capacitances.

The board can be supplied into two different ways: with flat cables or with round cables that make possible also the usage of the current probe.

They are shown below:

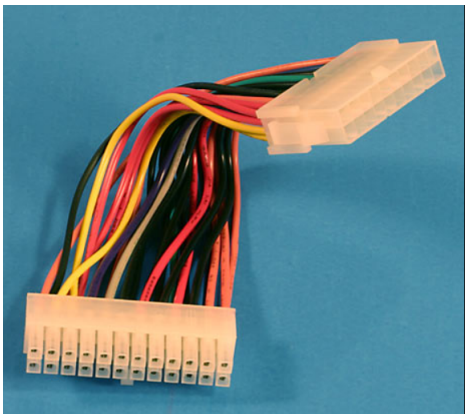

Figure 6.7: Round cables

Figure 6.8: Flat cables

One end goes to the power supply and the other to the board.

Flat cables are "new" in the market and Infineon Technologies is using them for many different reasons. First of all these cables have lower inductance and this is an advantage for the power integrity. The inductance of a cable depends on the wire diameter and the distance of the two conductors, this is why if there are X-inductor pairs in parallel instead of just a bigger one cable, the inductance is reduced by a factor X.

The flat cables are attached one with the other while the round cables are individual

and this makes possible to use the current probe that can wrap the cables for which we want to measure the current that flows in.

The effect of flat cables is to make Vcore more stable and so they reduce the possible peaks that are present on the signal.

Below it is reported the initial part of the verify, which is the more critical in terms of Vcore droops, because the initial droop cause by the PLL locking is the biggest one.

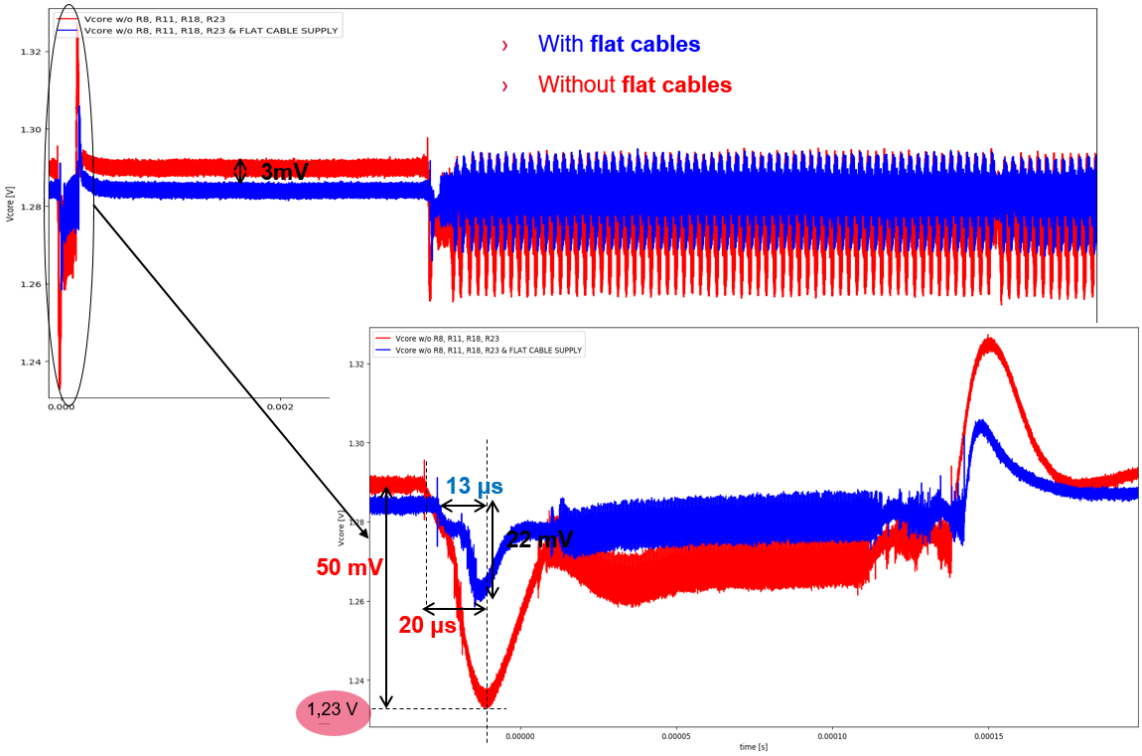

Figure 6.9: Vcore from the board supplied by flat cables during the initial part of the verify test

Is it possible to notice that all the peaks on the blue curve (Vcore measure on the board supplied with flat cables) are smaller with respect the ones on the red curve (Vcore measured on the board supplied with round cables).

In particular zooming on the initial peak we see a reduction of 27mV that is around the 50% of the initial peak.

Combining the effect of flat cables with the effect of the bulk capacitance we obtain a more stable voltage supply signal, because the peaks are furthermore attenuated, as it is possible to see below:

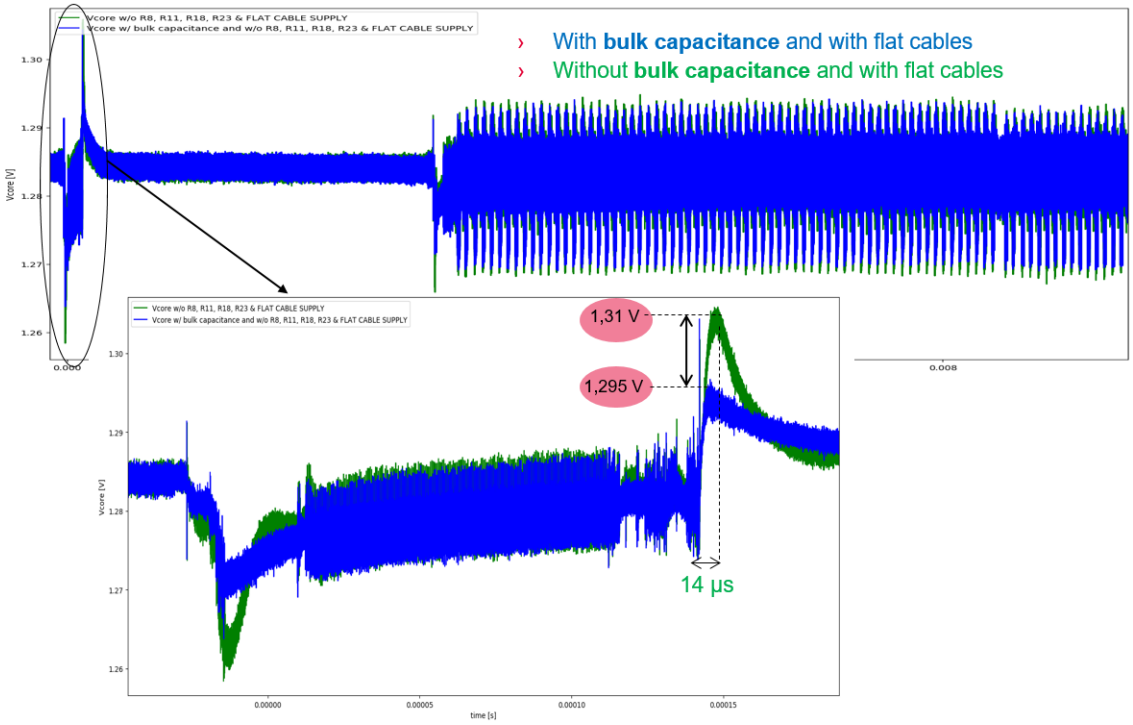

Figure 6.10: Vcore from the board with bulk capacitance and flat cables during the initial part of the verify test

As expected bulk capacitance make Vcore more stable, the peaks on the Vcore are attenuated of about 15 mV.

If we mix the contributes of using MEM8(), flat cables and bulk capacitance, we achieve a greater reduction of the Vcore droops.

In the following figure we show how the green curve (verify with MEM8()) is smaller with respect the blue curve (normal version of the verify) and in particular how the

first figure has the minimum equal to 1,27V while in the second one the minimum is equal to 1,24V.

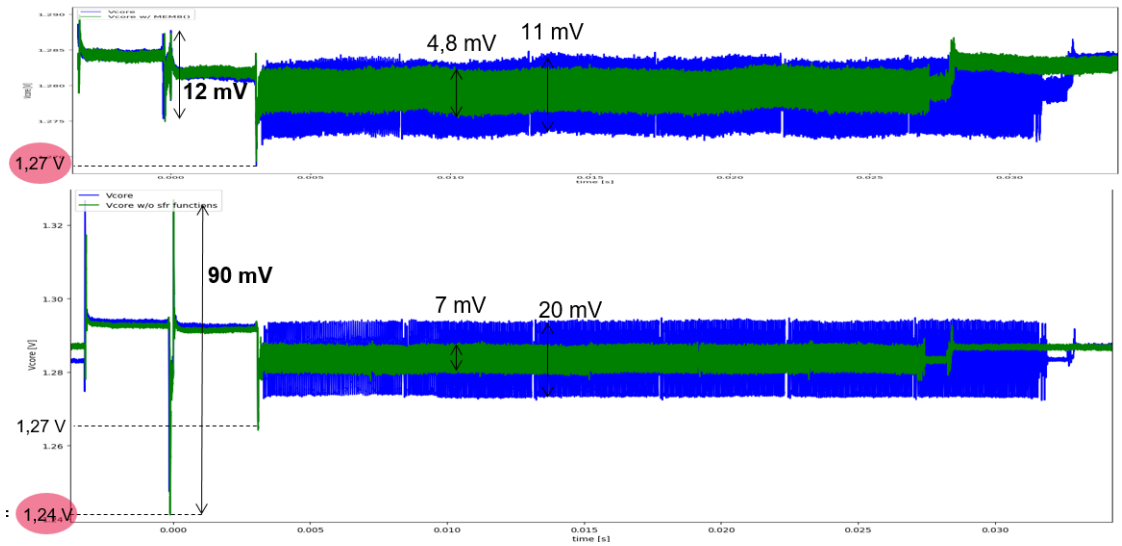

Figure 6.11: Vcore with MEM8(), bulk C and flat cables

This 30mV of difference are due to in the first figure the measurements have been collected after having plugged in the bulk capacitance and having supplied the board with flat cables, while in the second image the measurements are taken from the board supplied with round cables and without any capacitance.
This is a big advantage because the risk to exceed the voltage dynamic decreases.

Chapter 7

Wafer level

The wafer is a substrate of semiconductor which serves as the substrate for micro-circuits (DIEs) built in and upon it. At the end of its fabrication each DIE is cut from the others by dicing and then it is packaged as an integrated circuit.
The entire process is shown in the image below [15]:

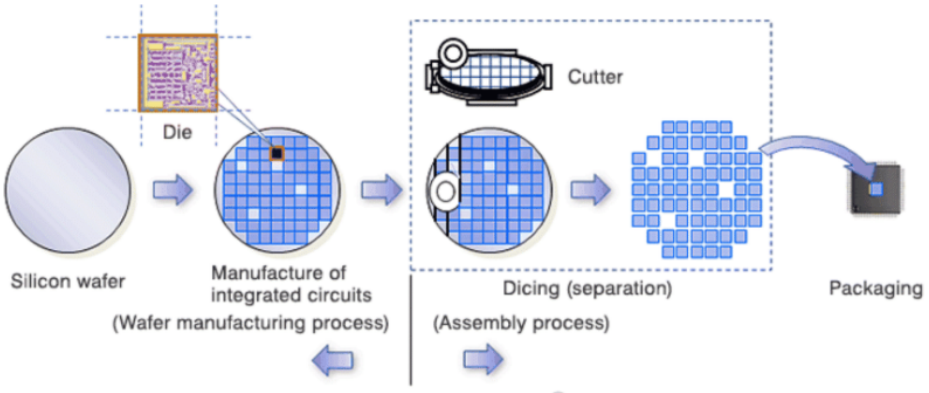

Figure 7.1: Manufacturing process of basic silicon wafers

The wafer fabrication is a sequential process where the wafer is many times exposed to photolithography after every physical process (doping, ion implantation, etching) during which the circuit is gradually created. Photolithography is a process that transfers the geometric pattern from a photomask to a photoresist on the wafer substrate.

7.1 Wafer probe

Before dicing the wafer, the DIEs are tested. Test signals are transmitted to each picopads of the single DIE via a proper probe card and then signals are collected from the device and compared with the excepted ones.
Each probe card is composed by an amount of picoprobes of the same number of the picopads of the DIE we are testing.

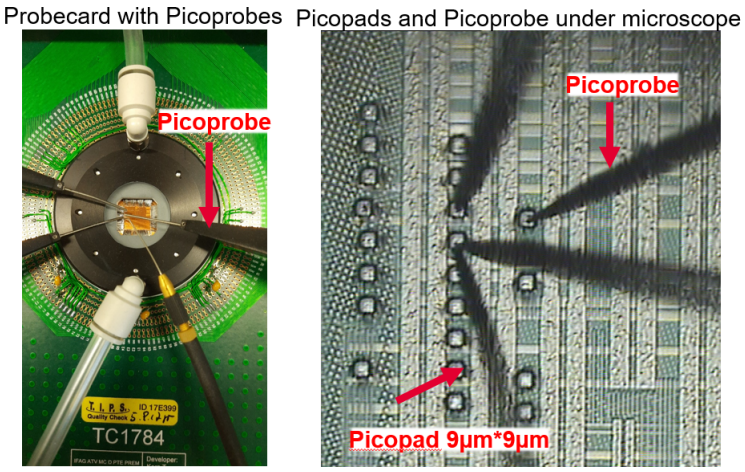

Figure 7.2: Picoprobes and picopads

Once a DIE is tested, the microprobes are shifted on other DIE's picopads until all the DIEs of the wafer have been tested.

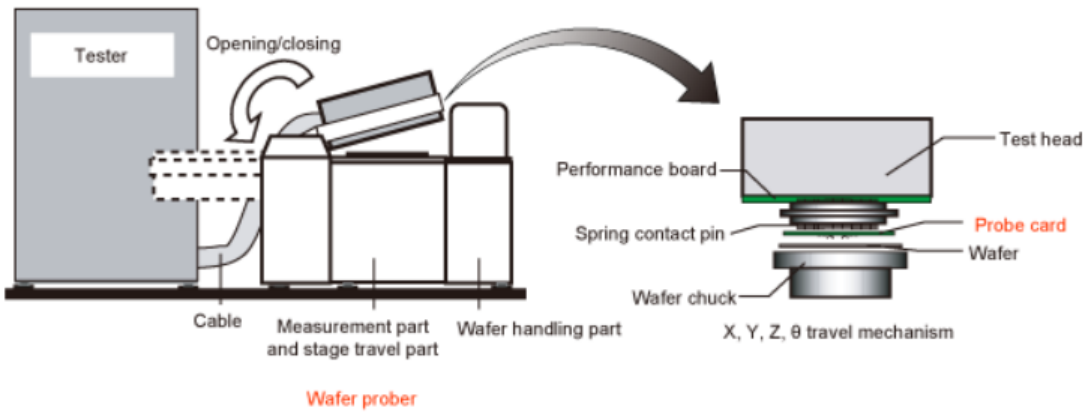

Figure 7.3: Wafer test system

The wafer test system is composed by different parts, as it shown in the figure above, the wafer is allocated on the chuck and then it is docked the probe card on the wafer which serves as a connector between the bonding pads of the DIE and the tester.

The needles of the probe card contact the pads of the DIE to conduct the tests and also with the auxiliary of external microprobes it is possible to choose the pad from which we want to perform the measure in order to observe the behavior of specific signal, in our case Vcore.

7.2 Measurements on the wafer

In this paragraph we want to highlight how the measurements taken on wafer level are worst with respect the one taken at board level due to the microprobes are directly contacting the virgin silicon of the DIE.

To have a reasonable comparison, we have using the same family of devices: A2G

TC38xEVO and we have executed the same test flow (S1) with JAZZ that we have used to test the chip on the board, but this time the executable file (the one obtained with Tasking) is modified a little because we want to avoid the initial peak present at the beginning of the verify test, caused by the PLL locking.

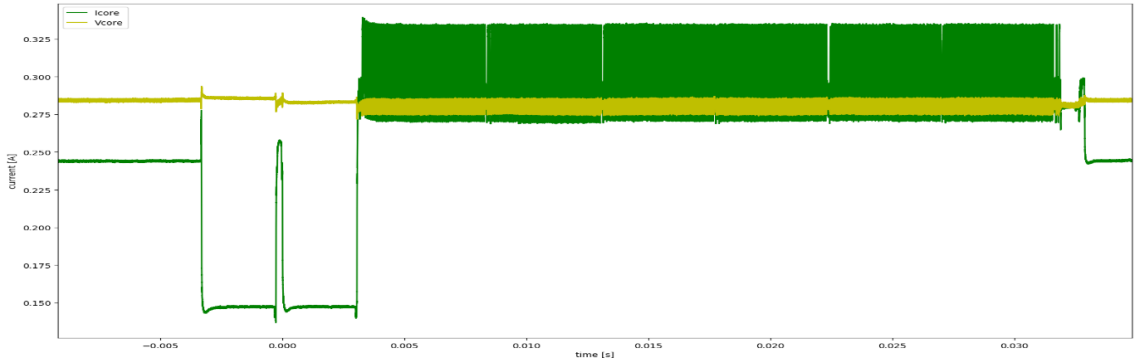

Figure 7.4: Verify w the initial locking of the PLL: Vcore (yellow) and Icore (green)

This peak is due to the PLL is locked/unlocked for every tes. To solve this problem it is specified in the test codes to lock it just one time at the beginning of the test flow and to maintain it locked for all the followed tests.This is a big advantage because avoiding this enormous current jump we avoid also variations of Vcore.

The new verify without the lock/unlock of the PLL has this shape, both for Icore (green) and Vcore (yellow):

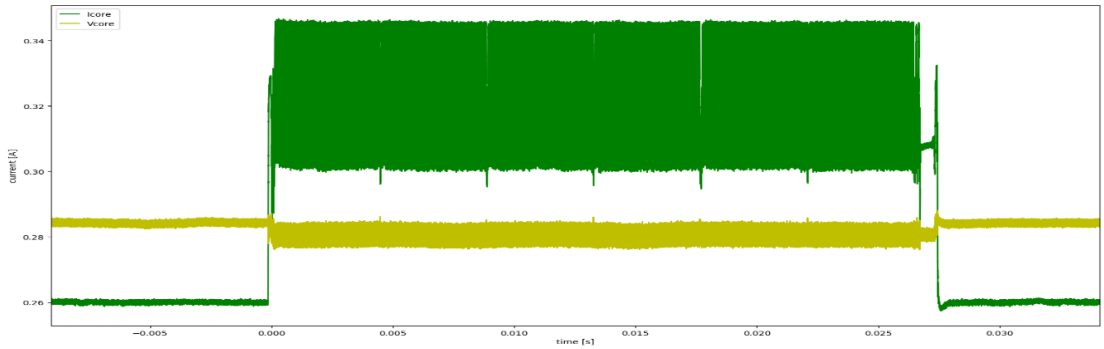

Figure 7.5: Verify w/o locking of the PLL: Vcore (yellow) and Icore (green)

All the next measurements on the wafer are done with this new executable file and therefore they will be compared with the verify in figure Figure 8.4 , where we lock the PLL just at the beginning of the flow and then it remains locked at that frequency for all the next tests.

In this chapter we start comparing the "board-level" with the "wafer-level" worst case scenario and then we improve the setup inserting the feedback loop and also capacitances.

The first comparison is done with the virgin DIE where are connected the micro-probes on the Vcore pins and then with Python are plotted both the Vcore taken from the board (yellow) and the one taken from the virgin DIE (red for Vcore and black for Icore).

Firstly, even if from the power supply are set 1.30V in both cases, the measurements done on the wafer seems to have it at 1.38V, but this shift is not real. This is just due to the fact we are using active micro-probes that a resistance in series at the end of the probe.

The peaks on the wafer level are much higher with respect the ones at the board level, to improve this measurement setup we can solder some capacitances on the

plug of the wafer probe and we have tried to do this experiment firstly with a very small capacitance (6,8nF) and then with a bigger one ($470\mu F$).

To have a more stable voltage supply it is inserted a feedback loop between the power supply and the Vcore's pin of the wafer, but the result is that the amplitude of Vcore remains unchanged while there is a shift of +20mV in its mean value, as shown below:

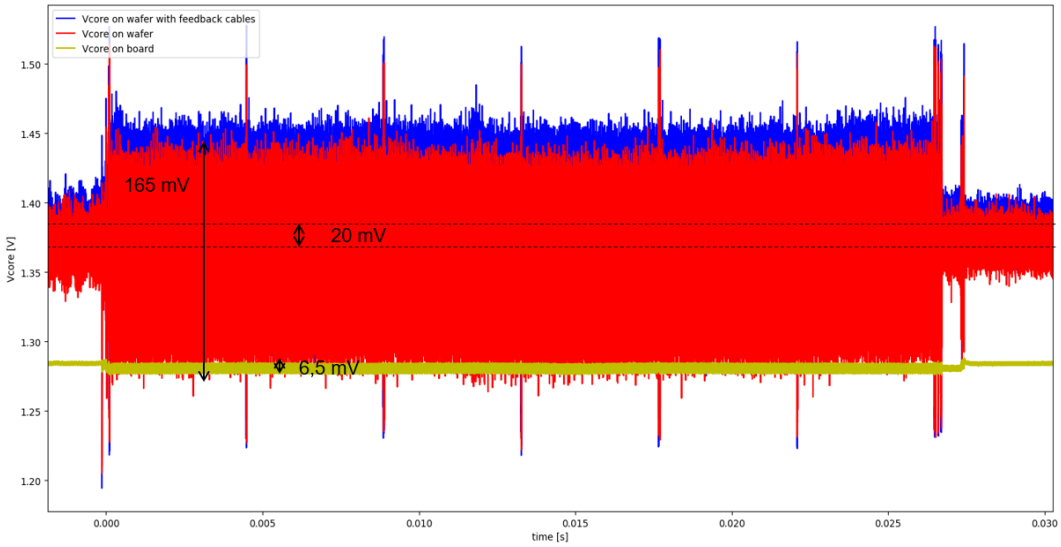

Figure 7.6: Vcore during the verify w/ and w/o feedback measured from the board (yellow) and from the wafer (red and blue)

Soldering the capacitance on the Vcore domain:

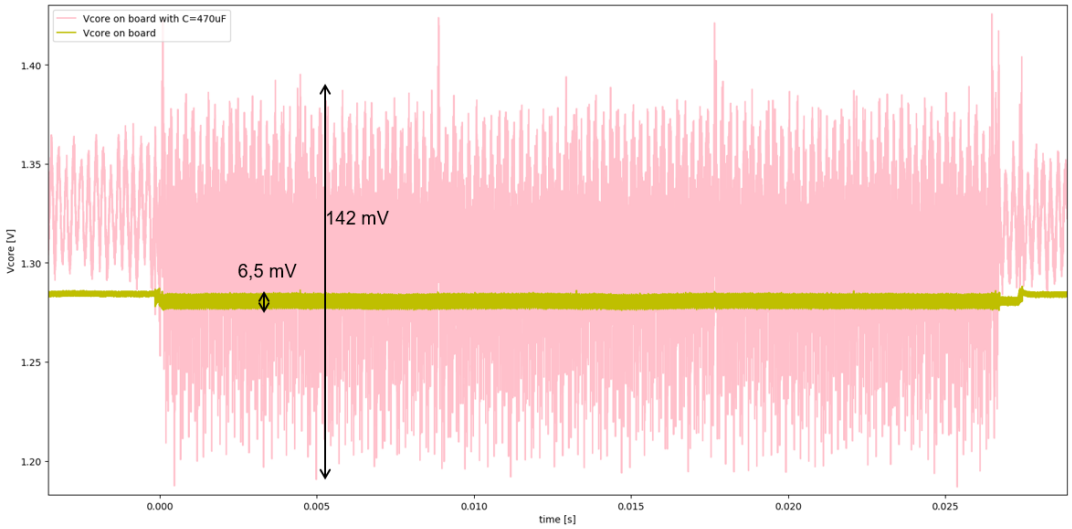

Figure 7.7: Vcore during the verify with $C = 470\mu F$: Vcore from the board (yellow) and from the wafer(pink)

If we focus on the verify of every single minisector, the effect of capacitances:

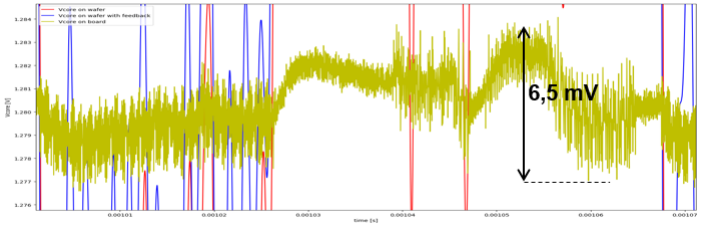

Figure 7.8: Mnsec measured on the board

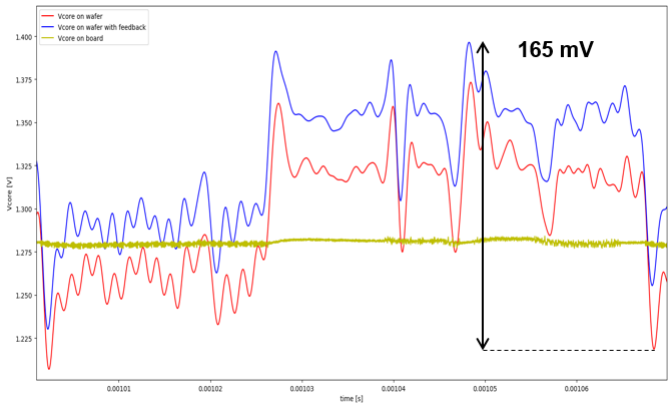

Figure 7.9: Mnsec measured on the wafer w/ and w/o feedback loop

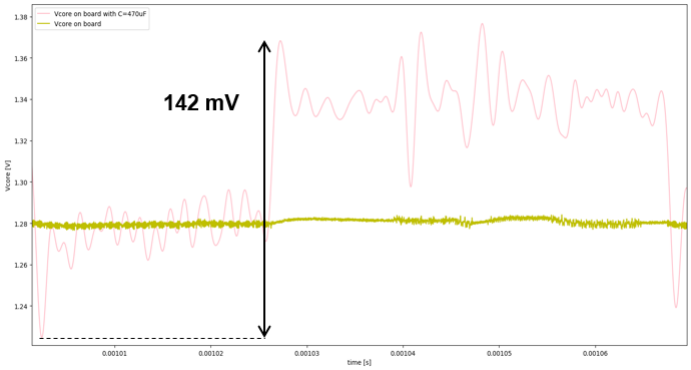

Figure 7.10: Mnsec measured on the wafer with $C = 470\mu F$

To summarize the results obtained at board and wafer level, it is reported below a table filled with the amplitude of the voltage droop that occurs when it is verified a minisector. As it is better highlighted in the table, there is a big variation in terms

	Vcore droops for a mnsec
On board	6,5mV
On wafer	165mV
On wafer with C=6,8nF	155mV
On wafer with C=470μF	142mV

Table 7.1: Vcore droops for verifying a minisector

of amplitude for every voltage droop, even if the test are exactly the same and the

devices used are of the same family (AURIX2G TC38xEVO). This is due to the presence of extra-resistances between picopads and picoprobes. This contribution is big considering also the small dimension we are working with, as a matter of fact each picopads is $9\mu m \cdot 9\mu m$, and the value of a R for an object is indirectly proportional to the dimension of the object itself. The presence of these resistances causes a bigger amplitude for each voltage droop.

Chapter 8

Deepening

In this paragraph is reported a parallel analysis done to further understand the power consumption during the test execution.

8.1 Pump Monitor effect

There is a contribution for the power consumption coming from the Pump Monitor (PM), which has the function to evaluate the state of the pumps and so understands if there are problems in the circuit or not, for example if a pump is working at 100% means that a short-circuit has occurred.
The PM can be enabled/disabled and this has an impact on the power consumed during the execution of the erase and program tests, in particular contrary to what expected if the PM is disabled the current requested increases otherwise decreases. To better understand the reason why this happens it is analyzed the test code more in detailed. In the interface of the different commands there is the function :

$$ftlib\_pump\_init(pinterface->$$
$$analysis.bit.pump\_monitor, params.pumpMonitorScheme):$$

```
VOID ftlib_pump_init(
        BOOL enable,
        FTLIB_PUMP_OPERATION_T operation )
{
    /* Set TestMode Pump Load monitor flag */
    ftlib_testmode_setPumpLoad( enable );
    /* Clear histogram data */
    ftlib_pump_histogramInit();

    if ( enable == TRUE )
    {
        initializePumpOperation( operation );
#if defined( DERIVATIVE_AURIX )
        readPumpLimitFromWorkbook( operation );
        readVDNTicksFromWorkbook();
#elif defined( DERIVATIVE_AURIX2G )
        readPumpLimitFromWorkbook();
#else
#error Not supported derivative - File: ftlib_pump.c
#endif

        FTLIB_PUMP_START_OSC();
        FTLIB_PUMP_START_MEASURE( FTLIB_PUMP_MONITOR_RESTART );
    }
    else
    {
        setInvalidOperation();
    }
    return;
}
```

Figure 8.1: ftlib_pump_init()

At the *ftlib_pump_init(pinterface− >analysis.bit.pump_monitor, params.pumpMonitorScheme)* function is passed the bit of the Pump Monitor (PM) and here it is decided to disable or enable it. To set this bit (0 = disable, 1 = enable) must written in the register "0x70000614" the second LSB equal to '1' or '0' depending if the PM must be enabled or disabled.

If it the PM is disabled we finish in the *"else"* branch, where this function is executed:

```
STATIC INLINE VOID setInvalidOperation( VOID )
{
    /* Set invalid operation  */
    FTLib_PumpParameters.operation = FTLIB_NUM_OF_PUMP_OPERATION;
    /* Reset pointer */
    FTLib_PumpParameters.psetting = NULL;
    /* Disable Pump Log */
    ftlib_testmode_setPumpLoad( FALSE );
    return;
}
```

Figure 8.2: setInvalidOperation()

In FTLib_PumpParameters.psetting is evaluated to decide if the state machine must start evaluating the pump's measures that are: *measureVppVdpPumpMonitor()*, *measureVppVpnPumpMonitor()* , *measureVppVdpPumpMonitorDisturb()* and all of these functions require to read from few SFRs registers.

As written before to disable the PM we must specify at the function : *ftlib_pump_init(pinterface-¿analysis.bit.pump_monitor, params.pumpMonitorScheme)* that the bit.pump_monitor is '0'. In order to do so it is added a new test in the JAZZ flow named: "disabled_PM.spt", it is inserted before the erase test.

The test code is the following:

RW32, 0x70000614, 0x00000000, 0x00000010

Below are reported the measurements of the erase test, in particular the current Icore when the Pump monitor is enabled (green curve) and when it is disabled (yellow curve) Icore:

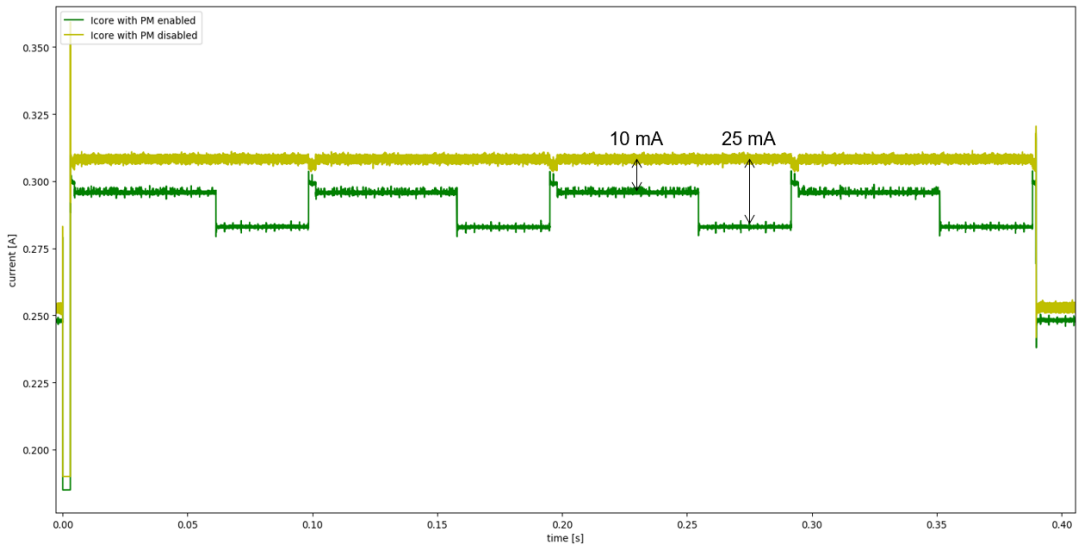

Figure 8.3: Icore on board level with PM enabled(green)/disabled(yellow) during erase test

Disabling the PM causes a variation in the current shape and an increasing in the current consumption of maximum 25mA but this is contrary to what is expected because without the PM the pumps are no more measured.

As anticipated before, also in the program test the disabling/enabling of the pumps implies a change in the current request both concerning its amplitude and shape.

Below it is reported part of the program test executed both with the PM disabled (black curve) and the PM enabled (yellow curve):

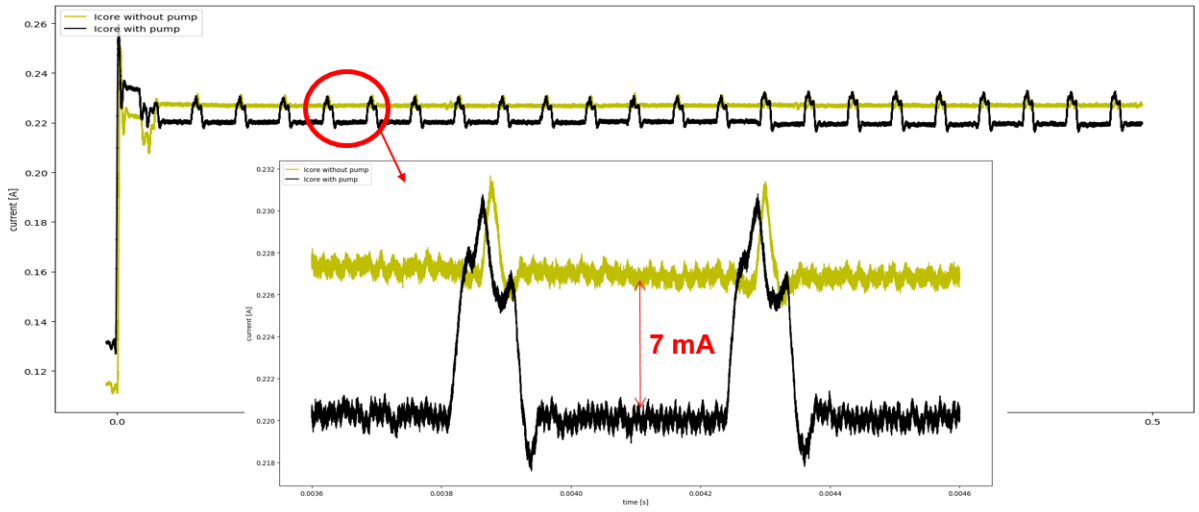

Figure 8.4: Icore on board level with PM enabled (green)/disabled (yellow) during program test

As for the erase test also in the program test we have the same effects: Icore changes shape and increases, of about 7mA.

To be sure that what we have discovered does not depends on some hardware on the ROVACS board, it is done the some analysis also on the wafer.

Below it is shown the erase test performed with the PM enabled and disabled and in both cases without the initial locking of the PLL:

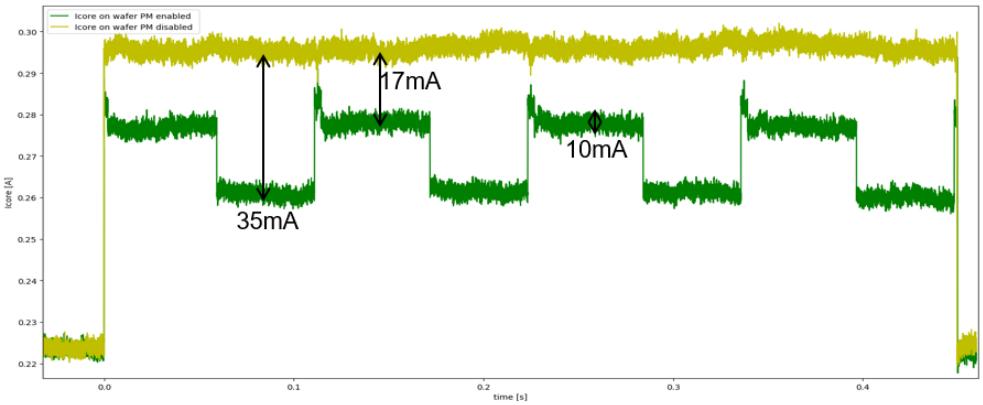

Figure 8.5: Icore on wafer level with PM enabled (green)/disabled (yellow) during erase test

If we compare this figure with the figure9.2 we see that the effect of disabling the PM is the same: the current increases and change shapes, but on the wafer level Icore increases more and on the waveforms there are 10mV of superimposed noise. What is the main cause of this current increasing remain not completely clear, however it could be attributed to the fact that when the Pump Monitor is disabled its circuitries are used for other purposes that consume more.

Chapter 9

Conclusion

In conclusion, testing memories is crucial in order to assure the functionality of an IC. A big variety of faults can affect the memory cell and, in order to test all of them, it is essential to choose a good test algorithm. The test algorithm is implemented in the form of test codes and each test code is characterised by having a certain power profile and TT.
With the aim of characterising each NVM test code, we have found three techniques for decreasing the power consumed by the IC during the test execution. All of them are based on the assumption that, the greater is the number of context switches, the higher is the amount of consumed power. Below it is an overview of the results obtained:

- Direct use of MEM8()

  MEM8() is a low level instruction which is used to load/store data from/to the specified memory location. It causes an important decreasing of the requested current.

- Modified version of ftlib_sfr_write():

  By commenting just two functions inside the original ftlib_sfr_write(), we can observe a decreasing of the current. These two functions are not essential for the tests, but they simply verify if the ftlib_sfr_write() has been done correctly

or not.

- Reduction of the frequency of the polling activity:

  Every time we need to monitor if an operation is finished, so repetitive accesses to the registers are performed. This activity requires a certain amount of power. By decreasing the number of accesses at these registers, also the requested current decreases.

In order to have an immediate idea of the efficiency of these techniques, it is reported a table with the values of the current Icore needed for the verification of a single minisector. Originally, verifying a minisector required 77mA. Instead, by using the above techniques, the requested current decreases according to the type of technique we are implementing.

	Icore required by a mnsec	**Gain**
Original test code (TC)	77mA	
TC with the new_ftlib_sfr_write()	68 mA	9 mA
TC with polling delayed of 100	56,5mA	20,5 mA
TC with MEM8()	14,5 mA	62,5 mA

Table 9.1: Icore peaks for verifying a minisector

As it is better highlighted by the table, the most efficient technique is the one with MEM8(). However, this technique makes the test code dedicated for the specific family of device we are testing and it is not suitable if the aim is having a reusable test code.

Moreover, it has been proved that, by using these techniques, we have advantages both on the power consumption and on the voltage supply droops that could cause failures. These voltage droops can be decreased also by using flat cables instead of

round cables and, furthermore, by soldering capacitance on the power supply.
Additionally, the comparison between the tests executed at the wafer, on the one hand, and at the board level, on the other, shows that in the latter the measurements are more accurate. In every test that has been repeated in both environments, the test time is the same and also the peaks occur at the exact same moment. However, the shape and the amplitude of every current peak change. In particular, from the wafer probe these peaks are much higher than the one measured from the board, on the IC. This result demonstrates how bad it is the wafer probe environment with respect to the board level environment in terms of measurement quality.
Future directions could be structured in the attempt of a better understanding of what is behind the remaining quantity of power. As an example we recall the 14.5mA measured even in the case that the low level instruction MEM8() is directly used.
Another possible step could be the developing of a test code that takes advantage of the usage of MEM8() but remains reusable as much as possible.

Appendix A

Python scripts

Before_test.py

This is the code that must be inserted before the test we want to measure, in order to set properly the oscilloscope basing on the characteristics of that test itself.

```
#BEFORE_TEST.py
import Ic_helper
from pverify.drivers.Scope.lcscope import IIviScope
from pverify.drivers.SimplifiedLabInstruments import SimpleScope

scope = SimpleScope(IIviScope())  #definition of oscilloscope object
resourcename = 'USB0::0x05FF::0x1023::2801N55814::INSTR' #ID of the oscilloscope used
scope.Initialize(ResourceName=resourcename, IdQuery=True, Reset=False, OptionString='')

# Setting sample_rate and time/DIV : ( (XX s/DIV)*10 , MAXIMUM SAMPLE POINTS )
scope.ScopeSetup(0.6, 25e6) #1ms/DIV

#CH2 used as trigger
scope.Trigger_Edge(Level=0.5, Slope='RISE', Position=1, Coupling='DC', ChannelIndex=2)

#SETTING EACH CHANNEL OF THE OSCILLOSCOPE:
scope.ch1 = scope.GetChannel(1)
```

```
scope.ch1.ProbeSetup(Coupling='DC', Bandwidth=20e6, Vrange=0.160,
                     Offset=-0.199, Position=0, Probe_Attn=1,
                     Probe_Type='voltage', Impedance=50)

scope.ch2 = scope.GetChannel(2)
scope.ch2.ProbeSetup(Coupling='DC', Bandwidth=200e6, Vrange=8,
                     Offset=-3.8, Position=0, Probe_Attn=1,
                     Probe_Type='voltage', Impedance=50)

scope.ch3 = scope.GetChannel(3)
scope.ch3.ProbeSetup(Coupling='DC', Bandwidth=200e6, Vrange=0.4,
                     Offset=-1.147, Position=0, Probe_Attn=1,
                     Probe_Type='voltage', Impedance=50)

scope.ch4 = scope.GetChannel(4)
scope.ch4.ProbeSetup(Coupling='DC', Bandwidth=200e6, Vrange=0.160,
                     Offset=-0.101, Position=0, Probe_Attn=1,
                      Probe_Type='voltage', Impedance=50)

# Arm scope waiting for the trigger
scope.Arm(Continuous=True)
```

After_test.py

Below it is shown the code of the script that must be inserted in the JAZZ flow after the test we want to analyse. This script is composed of a first part that simply defines the object "scope" to specify the oscilloscope we are using. Then a second part, in which are defined the two power supplies, one for the Vcore (1.3V) and one for Vext(5V) and Vddp3(3.3V). Depending on the value of the power supply we save the waveform with a different name, for example if we are analysing the situation in which the values on the supplies are maximum, the waveform of the Icore will be named: "ICORE_maximum.npz". Every time we want to save a waveform it is necessary to specify an existing path on the computer and if the waveform already exists with that name, it will be overwritten.

```
#AFTER_TEST.py
from pverify.drivers.Scope.lcscope import IIviScope
from pverify.drivers.SimplifiedLabInstruments import SimpleScope

scope = SimpleScope(IIviScope())  #definition of oscilloscope object
resourcename = 'USB0::0x05FF::0x1023::2801N55814::INSTR' #ID oscilloscope
scope.Initialize(ResourceName=resourcename, IdQuery=True, Reset=False,
                 OptionString='')

#EACH CHANNEL OF THE OSCILLOSCOPE:
scope.ch1 = scope.GetChannel(1)
scope.ch2 = scope.GetChannel(2)
scope.ch3 = scope.GetChannel(3)
scope.ch4 = scope.GetChannel(4)
```

```
dataOfCH1 = scope.ch1.GetProbeWaveform()
dataOfCH1.save_to_file(r'C:\Users\lavignia\Desktop\results\erase\signal1.npz')

dataOfCH2 = scope.ch2.GetProbeWaveform()
dataOfCH2.save_to_file(r'C:\Users\lavignia\Desktop\results\erase\signal2.npz')

dataOfCH3 = scope.ch3.GetProbeWaveform()
dataOfCH3.save_to_file(r'C:\Users\lavignia\Desktop\results\erase\signal3.npz')

dataOfCH4 = scope.ch4.GetProbeWaveform()
dataOfCH4.save_to_file(r'C:\Users\lavignia\Desktop\results\erase\signal4.npz')



#control also on the power supply:
from pverify.drivers.DCPwr.hpe364xa import IIviDCPwr
from pverify.drivers.SimplifiedLabInstruments import SimpleDCSource
# initialization
PWR_CORE = SimpleDCSource(IIviDCPwr()) #definition of pwr supply(1), Vcore
resourcename_core = 'GPIB0::5::INSTR' #ID of pwr1
PWR_CORE.Initialize(ResourceName=resourcename_core, IdQuery=True, Reset=False,
                    OptionString='simulate=False')
PWR_CORE.chcore = PWR_CORE.GetChannel(1)

PWR_ExtVddp = SimpleDCSource(IIviDCPwr()) #definition of pwr supply(2) (Vext and Vddp)
resourcename_ext_ddp3 = 'GPIB0::4::INSTR' #ID of pwr2
PWR_ExtVddp.Initialize(ResourceName=resourcename_ext_ddp3, IdQuery=True, Reset=False,
                       OptionString='simulate=False')
PWR_ExtVddp.ch1 = PWR_ExtVddp.GetChannel(1)
PWR_ExtVddp.ch2 = PWR_ExtVddp.GetChannel(2)
```

```
actualVcore = round(PWR_ExtVddp.ch2.Measure_Voltage(), 1) #only one decimal value after the comma
if actualVcore == 5.0:
  # now, change voltage for next iteration
    PWR_CORE.chcore.Configure_VoltageLevel(Level=1.33, CurrentLimit=1)
    PWR_ExtVddp.ch1.Configure_VoltageLevel(Level=3.63, CurrentLimit=1)
    PWR_ExtVddp.ch2.Configure_VoltageLevel(Level=5.5, CurrentLimit=1)
    PWR_CORE.chcore.Enable(Enabled=True)
    PWR_ExtVddp.ch1.Enable(Enabled=True)
    PWR_ExtVddp.ch2.Enable(Enabled=True)
elif actualVcore == 5.5:
    # save waveform at maximum voltage
    # now, change voltage for next iteration
    PWR_CORE.chcore.Configure_VoltageLevel(Level=1.14, CurrentLimit=1)
    PWR_ExtVddp.ch1.Configure_VoltageLevel(Level=3, CurrentLimit=1)
    PWR_ExtVddp.ch2.Configure_VoltageLevel(Level=4, CurrentLimit=1)
    PWR_CORE.chcore.Enable(Enabled=True)
    PWR_ExtVddp.ch1.Enable(Enabled=True)
    PWR_ExtVddp.ch2.Enable(Enabled=True)
elif actualVcore ==4.0:
#    save waveform at minimum voltage
    # now, set the nominal voltage
    PWR_CORE.chcore.Configure_VoltageLevel(Level=1.25, CurrentLimit=1)
    PWR_ExtVddp.ch1.Configure_VoltageLevel(Level=3.3, CurrentLimit=1)
    PWR_ExtVddp.ch2.Configure_VoltageLevel(Level=5, CurrentLimit=1)
    PWR_CORE.chcore.Enable(Enabled=True)
    PWR_ExtVddp.ch1.Enable(Enabled=True)
    PWR_ExtVddp.ch2.Enable(Enabled=True)
# Arm scope waiting for the trigger
scope.Arm(Continuous=True)
```

Manipulate_data.py

This script is executing a part from JAZZ and it simply opens the waveforms that have been previously saved, then from every waveform extrapolates its data and time and plot again the waveforms.

```
#MANIPULATE_DATA.py
from pverify.postproc.signals import Waveform
import matplotlib.pyplot as plt

wave1 = Waveform.load_from_file(filepath="C:/Users/lavignia/Desktop/results/erase/signal1.npz")
wave2 = Waveform.load_from_file(filepath="C:/Users/lavignia/Desktop/results/erase/signal2.npz")
wave3 = Waveform.load_from_file(filepath="C:/Users/lavignia/Desktop/results/erase/signal3.npz")
wave4 = Waveform.load_from_file(filepath="C:/Users/lavignia/Desktop/results/erase/signal4.npz")

data1 = wave1.data
time1 = wave1.time
data2 = wave2.data
time2 = wave2.time
data3 = wave3.data
time3 = wave3.time
data4 = wave4.data
time4 = wave4.time

plt.plot(time1, data1, 'r', label='Signal 1')
plt.plot(time2, data2, 'y', label='Signal 2')
plt.plot(time3, data3, 'b', label='Signal 3')
plt.plot(time4, data4, 'g', label='Signal 4')
plt.legend(loc=2)
```

```
plt.xlabel('time [s]')

plt.ylabel('[A] or [V]')

plt.show()
```

Bibliography

[1] Microcontroller Division Applications, *INTRODUCTION TO SEMICONDUCTOR TECHNOLOGY*, STMicroelectronics.

[2] Einfochips PES, *Memory Testing: MBIST, BIRA & BISR. An Insight into Algorithms and Self Repair Mechanism*, https://www.einfochips.com/blog/memory-testing-an-insight-into-algorithms-and-self-repair-mechanism/

[3] Michel Linder, *Test Set Optimization for Industrial SRAM Testing*, Master thesis TUM

[4] Akshay Pai, *Testing on Emulators vs Simulators vs Real Devices*, BrowserStack

[5] T. Windbache, *Flash Memory*, https://www.iue.tuwien.ac.at/phd/windbacher/node14.html

[6] Vatajelu Zambelli, *Nonvolatile Memories: Present and Future Challenges* , http://ishare.infineon.com/sites/EFE_Aurix3G/Shared%20Documents/xRAM/Vatajelu_Zambelli_NVM_Challenges_DATE_2014.pdfsearch=flash%20architecture%20cell

[7] Aya Fukami, *NAND Flash Memory Forensic Analysis and the Growing Challenge of Bit Errors*, https://www.semanticscholar.org/paper/NAND-Flash-Memory-Forensic-Analysis-and-the-Growing-Zandwijk-Fukami/311b34befb3c13dc49cc12049f3fc299bc5098fc/figure/0

[8] *FTOS -FSIST Interface*,

https://envm.drs.infineon.com/envm/FTOS/Documents/g1/125933-td.html

[9] *CURRENT PROBE TEKTRONIX MANUAL*, https://ishare.infineon.com/sites/TD-Lab/Manuals/Measurement%20device/English/Tektronix_%20Current%20Probe%20TCP312.pdf

[10] Dan Harmon, *Choose the Right Current-Measurement Technique for Your Application*, https://www.electronicdesign.com/technologies/test-measurement/article/21800806/choose-the-right-currentmeasurement-technique-for-your-application #:∼:text=Indirect%20current-measurement%20techniques%20are, Maxwell%%A2%E2%82%AC%E2%84%A2s%20Equations.&tex

[11] *Thermonics T-2500SE Temperature Forcing Systems*, https://www.atecorp.com/products/thermonics/t-2500se

[12] Konstantin Root, *Charakterisierung und Modellierung*

[13] Dan Harmon, *Choose the Right Current-Measurement Technique for Your Application*, %https://www.atecorp.com/products/thermonics/t-2500se

[14] Martin Huck, *Voltage droop*, http://ishare.infineon.com/sites/PTEam_Site/PTEUniversity/PTE_University/2019_10_08_Voltage_DROOP/2019_10_08_Voltage_DROOP.pptx

[15] Hafiz Zafar Nazir, *Robust adaptive exponentially weighted moving average control charts with applications of manufacturing processes*, https://www.researchgate.net/figure/Manufacturing-process-of-basic-silicon-wafers_fig4_335174577

[16] ELINFOR, *The Differences Between Flash Memory and Memory*, https://www.elinfor.com/knowledge/the-differences-between-flash-memory-and-memory-p-11191

[17] Paul Zandbergen, *What is Random-Access Memory?*, https://study.com/academy/lesson/what-is-random-access-memory-ram-definition-history-quiz.html#:~:text=RAM%20is%20considered%20volatile%20memory,not%20store%20any%20information%20permanently

[18] Paul Zandbergen, *What is Random-Access Memory?*, https://www.geeksforgeeks.org/classification-and-programming-of-read-only-memory-rom/:~:text=It%20consists%20of%20two%20basic,will%20represent%20its%20decimal%20equivalent%20.

[19] *MBIST (Memory Built-In Self Test)*, https://vlsiuniverse.blogspot.com/2013/05/mbist.html

[20] Arunkumar Krishnan, *SRAM and DRAM*, https://medium.com/@emailarunkumar/sram-and-dram-sdram-9b6d01f09eb7