LELAPE

GHADIR Group
Universidad Complutense de Madrid

March 1, 2022

Contents

Chapter 1

Introduction

Probably, the reason of reading this document is that you are a researcher that tests electronica devices under radiation. Even more, perhaps you are not interested in any electronic devices but, at this moment, only in commercial-off-the-shelf (COTS) memories.

Under the umbrella of memories, different devices are comprised. Let us enumerate them:

- Static Random Access Memories (SRAMs)

- Dynamic Random Access Memories (DRAMs)

- Non-volatile memories (Flash, PRAM, MRAM, etc.)

- Configuration memory in FPGAs

- Cache memory in microprocessors and microcontrollers

- ...

It is well known that these elements will show bitflips if they are exposed to protons, neutrons, heavy ions, ... The common procedure is to write a pattern in the memory and look for errors during read-back. These errors will be labeled indicating the word address and the flipped-bit position in the word.

Unfortunately, this is the so-called *phisical address* and it is impossible to relate it to the exact physical position on the integrated circuit. And this leads to a serious problem at the time of interpreting results. Nearby bitflips are probably caused by pernicious multiple cell upsets (MCUs) but they will not discovered unless the researcher has somehow got the information to relate any logical address to its physical address (an X, Y pair on the silicon surface).

However, the presence of MCUs will leave a signature in the set of the logical addresses of bitflips that can be used to group pairs of related bitflips and classify them in single or multiple events. No matter are the multiple events hidden among the rest of bitflips, we can track and locate them.

LELAPE is the Spanish acronym for *Listas de Eventos Localizando Anomalías al Preparar Estadísticas* , which is equivalent in English to LAELAPS (*Lists of All Events Locating Anomalies at Preparing Statistics*). In Greek mythology, LELAPE, or LAELAPS, is Zeus' hound, with the magic skill to track and hunt any prey however hidden it may be. In a similar way, LELAPE is a software tool able to inspect sets of apparently random logical addresses of bitflips and discover those that are members of the same multiple event.

Chapter 2

Theoretical Background
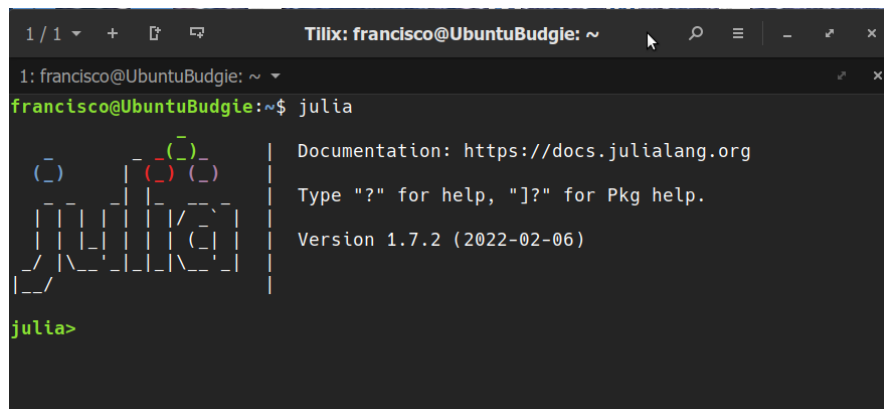
TBD

Chapter 3

How to install LELAPE

3.1 Why Julia?

The Julia language (https://julialang.org] was released in 2012 as a possible solution the classical "Two Languages' Problem"": a language easy to learn and develop in is usually slow and vice versa, so too often algorithms must be developped in one language and rewritten in another more efficient one. This language was specifically built to achieve speed, efficiency and clarity.

3.2 How to install Julia

Julia, an open software released with MIT license, can be downloaded and installed from https://julialang.org/downloads/ for different operating systems and architectures. Depending on the system, the binary files will be installed (Microsoft Windows, macOS) or just uncompressed (GNU/Linux, FreeBSD).
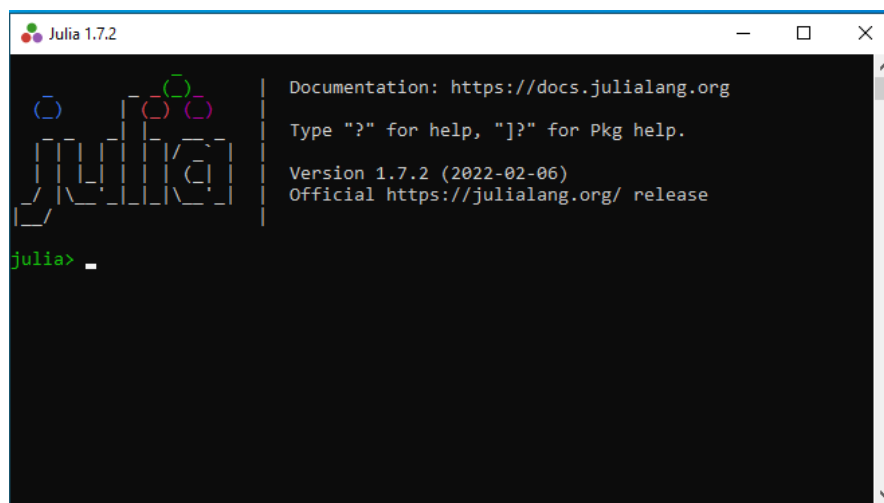
If no additional tools is installed, Julia will be executed in a REPL as it is shown in Fig. 3.1. However, most of the users prefer to use the language in conjunction with IDEs or notebooks such as:`

- **Visual Studio Code**: Popular IDE developed by Microsoft with plugins for Julia. It can be downloaded from https://code.visualstudio.com/ and the plugins installed through the "Extensions" tool.

- **Atom**: Just like VS Code, it is a general-purpose IDE with plugins for Julia.

- **Jupyter**: Although it is typically used for Python, it is also appropriate for Julia. Indeed, Jupyther is an acronym for **Ju**lia-**Pyth**on-**R**. There are different ways of installing Jupyter. In systems with Microsoft Windows OS, the most simple way is to install Anaconda https://www.anaconda.com/products/individual. Despite the fact of being designed for Python, Julia can be used as calculation engine. In GNU/Linux there are smaller packages to install Jupyter. For example, in Ubuntu the simple instruction `sudo apt install jupyter-notebook` will install the software in your computer. Later, it is necessary to install an additional package inside Julia but this will be studied a bit later.

- **Pluto**: A notebook following the philosophy of Jupyter but specifically developed for Julia and easily extensible with JavaScript. Unlike the previous tools, it is installed inside Julia, not along with it.

(a)

(b)

Figure 3.1: Example of the REPL's welcome screen for Julia on a machine running Ubuntu Budgie (a) and Microsoft Windows (b).

Figure 3.2: If Jupyter is not detected, Julia installs a minimal version the first time IJulia is used.

Atom, Jupyter and Pluto require the installation of additional packages. As Jupyter is probably the most popular tool, it is installed inside Julia REPL with the following instructions:

```
using Pkg; Pkg.add("IJulia")
```

This adds the package. However, a previous step is to search for Jupyter on your computer. In the case of not finding it, Julia suggests to dowinload a minimal version of Conda, thus installing Jupyter, as shown in Fig. 3.2.Finally, Jupyter inside Julia is launched as follows:

```
using IJulia; notebook()
```

Pluto package is installed following a similar procedure.

```
using Pkg; Pkg.add("Pluto");
```

and launched with:

```
using Pluto; Pluto.run()
```

There are other options if you prefer cloud computing. For example, in spite of the fact that its primary use is running Python code, Google Colab is compatible with Julia language. For further information (and also learning a little Julia), you can read Julia for Pythonists and use the Julia Colab Template. However, this solution is not recommended due to some problems at installing external packages as well as at loading data files. Perhaps this flaw can be fixed in the future, but, nowadays, the tool is not as powerful as the others.

3.3 Installing LELAPE

LELAPE is built as a module. In Julia, a module is a set of elements such as variables, functions, etc. that can be loaded at will. The procedure is the following:

Figure 3.3: How to indicate Julia where LELAPE is installed, and how to load it.

1. Download the ZIP system from the website and decompress it. Also, you can clone the site with git.

2. Find the folder where a file called LELAPE.jl is located.

3. Copy the full path pointing to this folder (PATH_TO_FOLDER) and execute in REPL, Jupyter or the notebook you use the following command:

```
push!(LOAD_PATH, "PATH_TO_FOLDER")
```

For example, if LELAPE.jl is found in /home/johndoe/Download/LELAPE/src/, the instruction is:

```
push!(LOAD_PATH, "/home/johndoe/Download/LELAPE/src")
```
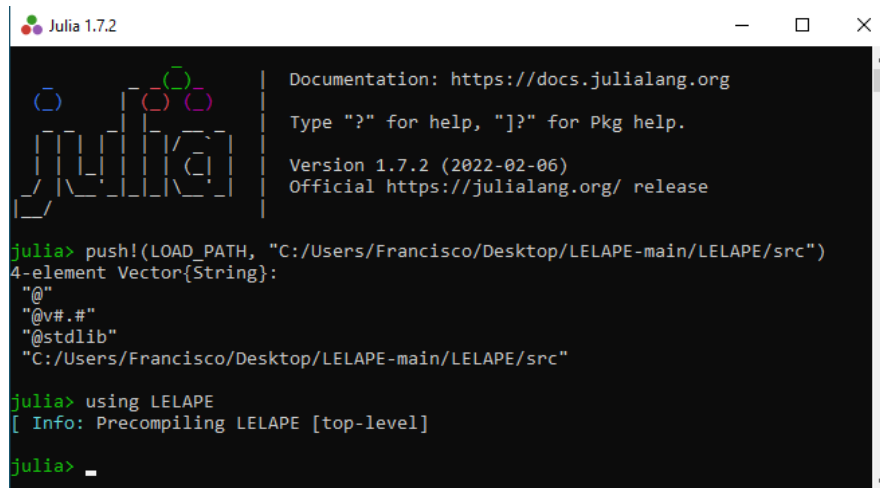
Thus, Julia knows where to find the module. LOAD_PATH is a string vector that contains the list of folder where Julia must look up external libraries. push! is a function that adds a new element at the end of any vector, keeping the name. Therefore, we have just added a new entry to the original list. Fig. 3.3 is a snapshot of the Julia terminal in GNU/Linux.

A warning for users of Microsoft Windows: in this operating system, folders in the path are marked with the symbol \. For technical reasons, this is not recognized in Julia, so the path to LELAPE must be modified with one of the following tips:

- Replacing \ with /, emulating the Unix style (GNU/Linux and Mac OS X).
- Replacing \ with \\.

Fig. 3.4 shows how to sucessfully add a new element to LOAD_PATH with the first option.

4. Now, just launch LELAPE with the following instruction:

Figure 3.4: The folder containing LELAPE is added to the admitted paths in Microsoft Windows.

$$using\ LELAPE$$

After a few seconds to precompile the library, the functions are loaded. Fig. 3.3 shows a practical example.

Jupyter users should be aware of a detail. Even if it is loaded on the terminal, it does not inherit loaded packages or modules, so they must be loaded again and independently in Jupyter. Even more, the `LOAD_PATH` variable is initialized with its default value, with only three elements, so the instruction `push!(LOAD_PATH, "PATH_TO_FOLDER")` must be executed again before loading LELAPE.

3.4 Recommended packages

There are many Julia packages at the user's disposal that can be found on https://juliapackages.com/. Some packages are extremely popular:

- **Revise**: Useful for code developers since it allows reloading user functions without restarting Julia and losing information.

- **OhMyREPL**: Inteligent highlighting of elements in REPL. Figs. 3.1 & 3.3 are using this package to show function names and strings.
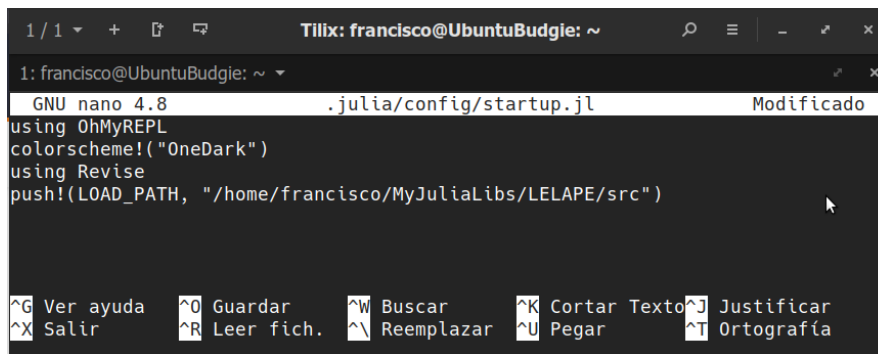
Both are installed with `Pkg.add()`.

Another interesting package to have is **DelimitedFiles**, which allows reading and writing CSV files. It is an essential package in Julia, available in a fresh installation, but not loaded by default. It is not necessary to fetch it and it is loaded as:

$$using\ DelimitedFiles$$

This package is necessary to use the illustrative Jupyter notebooks that are provided along with LELAPE.

A tip for new users: after working with Julia for a long time, you may eventually discover that only a few packages are frequently used and you may get bored of loading them every time you start a new session. Thus, the typical solution is to create the following folder and text file in your home directory:

Figure 3.5: Example of `startup.jl` file in GNU/Linux.

- In GNU/Linux & Mac OS X: `/home/<USER>/.julia/config/startup.jl`

- In Microsoft Windows: `C:\Users\<USER>\.julia\config\startup.jl`

Fig. 3.5 is an example of the `startup.jl`. In this file, one can see that LELAPE placement is automatically loaded when the session begins. However, if you wish to load these packages in Jupyter, it is necessary to create a new file, `startup_ijulia.jl`, with identical information. However, a soft link to `startup.jl` is enough.

Chapter 4

How to use LELAPE

4.1   Formatting input data

You are supposed to have performed experiments on some memory element. Tests were performed as:

- **Static**: The device was written, sent to standby mode, irradiated and eventually read. The content after the irradiation was compared to the initially writing.

- **Pseudostatic**: Similar to static ones, but standby intervals are shorter than the irradiation time and the memory is read several times during the irradiation. Usually, flipped bits are corrected on the flight.

In both cases, the researcher saves information about the bitfilps: Word address, read content, In order to use LELAPE, radiation test data must be converted to a matrix with three or four columns. The meaning of the columns is the following:

- First Column: Word Address where bitflips were observed.

- Second Column: Content in the word address after the radiation tests.

- Third Column: Content in the word address before the irradiation.

- Fourth Column: In pseudostatic tests, cycle in which the bitflip was observed. This column can be omitted in the static tests or replaced by a column full of ones.

LELAPE needs this elements to be converted to a UInt32 matrix[1] so it is important that all the elements of the matrix, including the cycle label, are in this format or, at least, in some kind of integer. This makes dangerous labelling the fourth column with words or letters.

A simple solution consists in grouping the data results in CSV format and read this text file with `readdlm`, included in the DelimitedFiles package. In the Jupyter folder, you can see some examples that can guide you to adapt your own data. One advantage of this function is that it can automatically convert the data to the required format, as shown in the Jupyter notebooks.

---

[1]In Julia, typing variable is optional but encouraged to speed up calculations.

4.2 Setting up the analysis

Before starting the analysis, we must define some additional variables to indicate the software how to proceed. These variables are the followong:

- **LA**: Variable in integer format. It indicates the memory size in words (not in bits!). It is often a power of 2.

- **WordWidth**: Also an integer, it indicates the number of bits per word. Typically 8, 16, 32 but other values are possible.

- **Operation**: A string variable to indicate the mathematical operation used to create the DV set. So far, only two options are implemented:

    - *XOR*: Addresses are xored bit to bit. This mode is set with the ``XOR'' value.
    - *Positive subtraction*: The absolute value of the difference of addresses is returned. It is marked with `"POS"`.

    In practice, we have observed that the former is appropriate for SRAMs and the latter for FPGAs. However, this idea may be erroneous due to the use of few and partial radiation test data.

- **UsePseudoAddress**: The pseudoaddress is defined as follows: let us suppose that we have observed a bitflip in the k-th position of the NWA-th word address, The word width is W and $k = 0$ corresponds to the least significant bit, $W-1$ to the most significant one. Hence, the pseudoaddress of the bitflip is:

$$PSA = NWA \cdot W + k$$

    This value is full of meaning in FPGAs since just returns the position of the cell in the bitflips. It is completly artificial in some SRAMs but somehow analysis using the pseudoaddress instead of the word address are more accurate and efficient.

    The researcher can set this variable to `true` or `false` at will.

- **KeepCycles**: In pseudostatic tests, this boolean variable indicates that the system must use the information about the cycles (fourth column) or just using the set of data as a whole.

- **TraceRuleLength**: This an integer variable with 1,2 or 3 as allowed values. LELAPE looks for anomalously repeated elements in the DV set with very few ones in binary representation. The user can decide if looks for elements with 1, 2 or 3 ones or less and include them as candidates to detect pairs.

- ε: A float number always positive but close to 0. If the expected number of elements repeated k times in the DV set is lower than ε, we must consider this number of repetitions impossible. If higher, we determine than at least an element can appear k times just due to randomness. Default value is 0.05.

    A very low value of ε will exclude false positive but also genuine values relating pair of addresses in an MCU. On the contrary, if it is chosen too low, false positives might be taken as good ones.

- **LargestMCUSize**: During the search of critical DV values, LELAPE starts to group addresses in provisional MCUs that grow large and large as new possible critical DV values are tested. Unfortunately, sometimes this process does not find a stable solution and goes on looking for it despite being unrealistic. This parameter is used to stop the calculation since it informs the software of not considering MCUs with more than *LargestMCUSize* addresses. By default, it is set to 200.

4.3   Functions in the module

ConvertToPseudoADD

- **Arguments**:

  - *Method 1:* **DATA**::Array{UInt32}, **WordWidth**::Int
  - *Method 2:* **DATA**::Array{UInt32}, **WordWidth**::Int, **KeepCycle**::Bool

- **Output**: Array{UInt32, 2}, or Matrix{UInt32}.

- This function looks for the flipped bits between words in the same row but in the second and third columns of the DATA matrix. It does not matter if there are several bitflips, since they are independently coundted. The pseudoaddress of each bitflip, defined as

$$WORDADDRESS \times Wordwith + Bitposition$$

  is returned as the first column of the output.

  If there is information about the different cycles, it can be kept in the optional second column in the output with the condition of declaring **KeepCycle** true. If cycle information is absent, the second column is filled with 1s.

ExtractFlippedBits

- **Arguments**:

  - *Method 1:* **WORD**::UInt32, **PATTERN**::UInt32, **Wordwidth**::Int
  - *Method 2:* **WORD**::UInt16, **PATTERN**::UInt16, **Wordwidth**::Int
  - *Method 3:* **WORD**::UInt8, **PATTERN**::UInt8, **Wordwidth**::Int

- **Output**: :Array{Int,1}, or Vector{Int}

- This function allows discovering the position of different bits between WORD and PATTERN. It also verifies that both values are coherent with the **Wordwidth**, meaning that neither of them are higher than $2^{Wordwidth} - 1$. A vector, never larger than **Wordwidth** is returned. If **WORD** and **PATTERN** are equal, the output is a void vector.

CheckMBUs

TBD

DetectAnomalies_SelfConsis

- **Arguments**:

    - *Method 1:* **DATA**::Array{UInt32, 2}, **WordWidth**::Int, **LN0**::Int, **Operation**::String, **UsePseudoADD**::Bool, **KeepCycle**::Bool, ε::AbstractFloat, **LargestMCUSize**::Int

    - *Method 2:* **DATA**::Array{UInt32, 2}, **WordWidth**::Int, **LN0**::Int, **Operation**::String, **UsePseudoADD**::Bool, **KeepCycle**::Bool, ε::AbstractFloat

    - *Method 3:* **DATA**::ArrayUInt32, 2, **WordWidth**::Int, **LN0**::Int, **Operation**::String, **UsePseudoADD**::Bool, **KeepCycle**::Bool

- **Output:** Array{UInt32, 2}

- This function will calculate the anomalies in the set of addresses using the Self-Consistency principle.

    First of all, let us know the inputs:

    - **DATA**: A matrix with 3 or 4 columns.
        * The first one contains the word addresses in UInt32 format.
        * The second one shows the content read in the memory after the irradiation.
        * The third one, the pattern that should be inside.
        * The fourth one is optional and shows the namber of the read cycle if the memory was read and corrected several times during the irradiation.
    - **WordWidth**: The size of each word in bits, usually 8, 16. 32, etc.
    - **LN0**: The memory size in words (not in bits!!!). In many cases, 2^N.
    - **Operation**: A string variable to indicate the preferred operation to calculate the DVSET. Only two operations are allowed:
        * "XOR": XORing bit to bit.
        * "POS": abs(a-b)
    - **UsePseudoADD**: A boolean variable. It allows to indicate that the user wants to user word addresses (false). If true, a pseudoaddress is assigned to each bit and calculated as $\text{WORADDRESS} \times \text{WordWidth} + \text{BitPosition}$. Full of sense in FPGA since it is just the position of the bit in the bitstream, it has not physical interpretation in memories BUT works!!!!
    - **KeepCycle**: If true, the function looks for the fourth column and uses it to calculate the DVSet.
    - ε: A small positive integer number to determine the threshold that defines when a number of repetitions are impossible to occur. Set by default to 0.05 if not provided among the input arguments.
    - **LargestMCUSize**: This value indicates the largest possible size for MCUs. It has not physical sense and is only used to stop the program if unreallistic events occur. Set to 200 if not given as an input.

        The funcion return an $N \times 2$ UInt32 matrix. The first column contains the anomalously repeated values of the DV SET compatible with the SelfConsistency test. The second one contains the number of times they appear

in the DV set. Due to format integrity reasons, this column is expressed in unnatural UInt32 format.

It is advisable a latter conversion into Int to make this column more readable.

DetectAnomalies_Shuffle_Rule

TBD

DetectAnomalies_Trace_Rule

TBD

DetectAnomalies_MCU_Rule

TBD

DetectAnomalies_FullCheck

TBD

MCU_Indexes

- **Arguments**:

    - *Method 1:* **DATA**::Matrix{UInt32}, **OPERATION**::String, **Markers**:: Vector{UInt32}, **UsePseudoADD**::Bool, **WordWidth**::Int, **LimitMCUSize**:: Int

    - *Method 2:* **DATA**::Matrix{UInt32}, **OPERATION**::String, **Markers**:: Vector{UInt32}, **UsePseudoADD**::Bool, **WordWidth**::Int

    - *Method 3:* **DATA**:Matrix{UInt32}, **OPERATION**::String, **Markers**:: Vector{UInt32}, **UsePseudoADD**::Bool

    - *Method 4:* **DATA**::Matrix{UInt32}, **OPERATION**::String, **Markers**:: Vector{UInt32},

- **Output**: Matrix{Int}

- This functions uses the **DATA** set to look for pairs of addresses which treated with **OPERATION** yield one of the **MARKERS**. If an **ADDRESS** is related to other two addresses, a 3–bit MCU appears (and so on.) The rest of parameters are used to provide necessary information to use the **PSEUDOADDRESS** instead of the **WORD ADDRESS**.

    More information about the inputs:

    - DATA: A matrix with 3 or 4 columns.
        * The first one contains the word addresses in UInt32 formata.
        * The second one shows the content read in the memory after the irradiation.

∗ The third one, the pattern that should be inside.
　　　　∗ The fourth one is optional and shows the namber of the read cycle if the memory was read and corrected several times during the irradiation.
　　– **OPERATION**: A string variable to indicate the preferred operation to calculate the DVSET. Only two operations are allowed:
　　　　∗ "XOR": XORing bit to bit.
　　　　∗ "POS": abs(a-b)
　　– **UsePseudoADD**: A boolean variable. It allows to indicate that the user wants to user word addresses (false). If true, a pseudoaddress is assigned to each bit and calculated as

$$\text{WORDADDRESS} \times \text{WordWidth} + \text{BitPosition}.$$

　　Full of sense in FPGA since it is just the position of the bit in the bitstream, it has not physical interpretation in memories BUT works!!!!
　　– **WordWidth**: The size of each word in bits, usually 8, 16. 32, etc.
　　– **LargestMCUSize**: This value indicates the largest possible size for MCUs. It has not physical sense and is only used to stop the program if unreallistic events occur. Initially set to 200.

Concerning the OUTPUT: It provides an integer NMCU×LMCU matrix, NMCU being the number of detected MCUs and LMCU the size of the largest reconstructed MCU. Every value different than 0 must be determined as follows:

1. UsePseudoADD = false: The index indicates the row in DATA with the address in the MCU.

2. UsePseudoADD = true: It provides the index in the PSEUDOADDRESS derived SET. If the exact position of the bitcell is required, DATA should be treated with ConvertToPseudoADD() and the index used in the resulting matrix.

In both cases, if the size of the MCU is smaller than LMCU, the row will be filled with zeros until reaching the desired length. For example, if the content of a row is [5 7 9 0 0], it must be interpreted as a 3-bit MCU involving addresses indexed with 5, 7 & 9 in an experiment in which at least a 5-bit MCU (and nothing larger) was observed.

Finally, if the index of an address does not appear in the returned matrix, it should be interpreted as isolated and belonging to an SBU.

Classify_Addresses_in_MCU

- **Arguments**:

　　– *Method 1*: **DATA**:: Matrix{UInt32}, **Indexes**:: Matrix{Int}, **UsePseudoADD**:: Bool, **WordWidth**:: Int
　　– *Method 2*: **DATA**:: Matrix{UInt32}, **Indexes**:: Matrix{Int}, **UsePseudoADD**:: Bool
　　– *Method 3*: **DATA**:: Matrix{UInt32}, **Indexes**:: Matrix{Int}

- **Output**: Vector:: {Any}

- The purpose of this function is to classify the addresses (or pseudoaddresses) with bitflips transform the matrix of INDEXES got from MCU_Indexes() into a Vector of matrices, called SOLUTION, which is eventually returned as OUTPUT.

  The length of SOLUTION is the size of the largest observed MCU(s), NLMCU. Thus, SOLUTION[1] is a N x NLMCU matrix in which every row contains the addresses or pseudoaddresses of the NLMCU bitflips involved in this MCU. N is the number of observdd NLMCU-bit MCUs.

  SOLUTION[2] is devoted to events with M = NLMCU-1 bits. As before, it is a matrix with NLMCU-1 rows and an undetermined number of rows.

  Finally, SOLUTION[NLMCU] is just a simple vector with the addresses not involved in MCUs. Obviously, these are the SBUs.

TheoAbundance_XOR

- **Arguments:**

  - *Method 1*: **NR**: :Int, **NB**::Int, **LN**::Int, **UsingDV**::Bool
  - *Method 2*: **NR**: :Int, **NB**::Int, **LN**::Int

- **Output:** AbstractFloat

- This function allows calculating the expected number of values repeated **NR** times after **NB** bitflips in a memory with **LA** words with **W** bits per word with XOR operation.

  NR must be an integer number higher or equal than 0; **NB** is an integer number supposed to be higher than 1. **LN** is an integer number, and indicates the size of the elements in the set where elements are chosen.

  Sometimes, the user provides directly the DV set so an additional boolean input is provided to take into account this fact. It is usually disabled. In this case, NB PLAYS THE ROLE OF NDV.

  Equations were got from Eq.12 of the Appendix.C in F. J. Franco et al., *"Statistical Deviations From the Theoretical Only-SBU Model to Estimate MCU Rates in SRAMs,"* in IEEE Transactions on Nuclear Science, vol. 64, no. 8, pp. 2152-2160, Aug. 2017, doi: 10.1109/TNS.2017.2726938.

  Lawfully avalaible for free on https://eprints.ucm.es/id/eprint/43874/

TheoAbundance_POS

- **Arguments**:

  - *Method 1*: **NR**: :Int, **NB**::Int, **LN**::Int, **UsingDV**::Bool
  - *Method 2*: **NR**: :Int, **NB**::Int, **LN**::Int

- **Output:** AbstractFloat

- This function allows calculating the expected number of values repeated NR times after NB bitflip in a memory with LA words with W bits per word supposing to have used the POSITIVE subtraction.

  NR must be an integer number higher or equeal than 0 **NB** is an integer number supposed to be higher than 1. **LN** is an integer number, and indicates the size of the elements in the set where elements are chosen.

  Sometimes, the user provides directly the DV set so an additional boolean input is provided to take into account this fact. It is usually disabled. In this case, NB PLAYS THE ROLE OF NDV.

  Equations were got from Eq.2 of the Appendix in J. C. Fabero et al., *"Single Event Upsets Under 14-MeV Neutrons in a 28-nm SRAM-Based FPGA in Static Mode,"* in IEEE Transactions on Nuclear Science, vol. 67, no. 7, pp. 1461-1469, July 2020, doi: 10.1109/TNS.2020.2977874.

  Lawfully avalaible for free download on https://eprints.ucm.es/id/eprint/59496/

TheoAbundance

- **Arguments**:

  - *Method 1:* **NR**::Int, **NB**::Int, **LN**::Int, **Operation**:: String, **UsingDV**::Bool
  - *Method 1:* **NR**::Int, **NB**::Int, **LN**::Int, **Operation**:: String

- **Output**: AbstractFloat

- This is an Alias for TheoAbundance_POS() or TheoAbundance_XOR().

  NR must be an integer number higher or equeal than 0 **NB** is an integer number supposed to be higher than 1. **LN** is an integer number, and indicates the size of the elements in the set where elements are chosen. **Operation** is "XOR" or "POS".

  Sometimes, the user provides directly the DV set so an additional boolean input is provided to take into account this fact. It is usually disabled. In this case, NB PLAYS THE ROLE OF NDV. If **UsingDV** is not provided, it is supposed to be false.

MaxExpectedRepetitions

- **Arguments**:

  - *Method 1*: **NDV**::Int, **LN**::Int, **Operation**:: String, ε:: AbstractFloat
  - *Method 2*: **NDV**::Int, **LN**::Int, **Operation**::String

- **Output**: Int

- The purpose of this funcion is to determina the maximum number of expected repetitions. in a DV set taken from a memory with size equal to **LN**. In general, it is the first integer such that its theoretical abundance is lower than ε. If this threshold is not provided, it is assumed to be 0.01.

CorrectNBitFlips

- **Arguments**: **NBF**::Int, **LN**::Int

- **Output**: Float64

- This function tries to correct the number of bitflips to compensate cells hit twice that escape from inspection.

    - **NBF**: Number of bitflips. Theoretically, SBUs but they are impossible to be distinguished from other kinds of bitflips. Therefore, a simple approach is taken.
    - **LN**: Memory size in BITS!!!!!

    Expression is taken from Eq. 6 in F. J. Franco, J. A. Clemente, H. Mecha and R. Velazco, *"Influence of Randomness During the Interpretation of Results From Single-Event Experiments on SRAMs,"* IEEE Transactions on Device and Materials Reliability, vol. 19, no. 1, pp. 104-111, March 2019, doi: 10.1109/TDMR.2018. 2886358.

NF2BitMCUs

- **Arguments**:

    - *Method 1*: **NSBU**::Int, **LA**::Int, **METHOD**::String, **D**::Int, **WordWidth**::Int, **UsePseudoAddress**::Bool
    - *Method 2*: **NSBU**::Int, **LA**::Int, **METHOD**::String, **D**::Int, **WordWidth**::Int

- **Output**: Float64

- It indicates the expected number of false 2-bit MCUs that will occur in a memory with **LA** words with **WORDWIDTH** bits each in which **NSBU** SBUs have occurred. In this analysis,

    MCUS are sought using some grouping method (**METHOD**) with a generalized distance **D**.

    Admitted values for **METHOD** and **D** are the following:

    1. METHOD: "MBU" –> Only MBUs are sought. In this case, D is the Wordwidth.
    2. METHOD: "MHD" –> Only possible if the user has been able to place the ExtractFlippedBits cell in the XY plane. Two cells are related if |x1-x2|+|y1-y2| <= D. This is the Manhattan distance.
    3. METHOD: "IND" –> Only possible if the user has been able to place the ExtractFlippedBits cell in the XY mplane. Two cells are related if max(|x1-x2|,|y1-y2|) <= D. In mathematics, this is the "infinite distance".
    4. METHOD: "THD" –> Only valid if pairs of bitflips are located in a linear bit-stream and if the distance between cells is smaller than D: |x1-x2| <= D.
    5. METHOD: "XOR" –> Related pairs are got by means of statistical deviations. Addresses are XORed and only if the value is one of the D possible critical values. If the WORD Addresses is used instead of PSEUDOADDRESS, the memory size must be expressed in WORDs, LA = LN/WordWidth. IF SO, THE WORDWIDTH MUST BE PROVIDED.

6. METHOD: "POS" –> Identical to the previous one but with positive subtraction instead of XOR.

Everything can be found in Eq. 11 of F. J. Franco, J. A. Clemente, G. Korkian, J. C. Fabero, H. Mecha and R. Velazco, *"Inherent Uncertainty in the Determination of Multiple Event Cross Sections in Radiation Tests,"* IEEE Transactions on Nuclear Science, vol. 67, no. 7, pp. 1547-1554, July 2020, doi: 10.1109/TNS.2020.2977698.

NF3BitMCUs

- **Arguments**:

  - *Method 1*: **NSBU**::Int, **NMU2**::Int, **LN**::Int, **METHOD**::String, **D**::Int, WordWidth::Int
  - *Method 2*: **NSBU**::Int, **NMU2**::Int, **LN**::Int, **METHOD**::String, **D**::Int

- **Output**: TupleFloat64, Float64

- It indicates the expected number of false 3-bit MCUs that will occur in a memory with LN bits in which NSBU SBUs and NMU2 2-bit MCUs have occurred. In this analysis, MCUS are sought using some grouping –method (METHOD) with a generalized distance D.

  MCUS are sought using some grouping method (**METHOD**) with a generalized distance **D**.

  Admitted values for **METHOD** and **D** are the following:

  1. METHOD: "MBU" –> Only MBUs are sought. In this case, D is the Wordwidth.
  2. METHOD: "MHD" –> Only possible if the user has been able to place the ExtractFlippedBits cell in the XY plane. Two cells are related if |x1-x2|+|y1-y2| <= D. This is the Manhattan distance.
  3. METHOD: "IND" –> Only possible if the user has been able to place the ExtractFlippedBits cell in the XY mplane. Two cells are related if max(|x1-x2|,|y1-y2|) <= D. In mathematics, this is the "infinite distance".
  4. METHOD: "THD" –> Only valid if pairs of bitflips are located in a linear bitstream and if the distace between cells is smaller than D: |x1-x2| <= D.
  5. METHOD: "XOR" –> Related pairs are got by means of statistical deviations. Addresses are XORed and only if the value is one of the D possible critical values. If the WORD Addresses is used instead of PSEUDOADDRESS, the memory size must be expressed in WORDs, LA = LN/WordWidth. IF SO, THE WORDWIDTH MUST BE PROVIDED.
  6. METHOD: "POS" –> Identical to the previous one but with positive subtraction instead of XOR.

Everything can be found in Eq. 11 of F. J. Franco, J. A. Clemente, G. Korkian, J. C. Fabero, H. Mecha and R. Velazco, *"Inherent Uncertainty in the Determination of Multiple Event Cross Sections in Radiation Tests,"* IEEE Transactions on Nuclear Science, vol. 67, no. 7, pp. 1547-1554, July 2020, doi: 10.1109/TNS.2020.2977698.

In this paper, it was demonstrated that it is mathematically impossible to get an exact value. Therefore, optimistic and pessimistic results are provided.