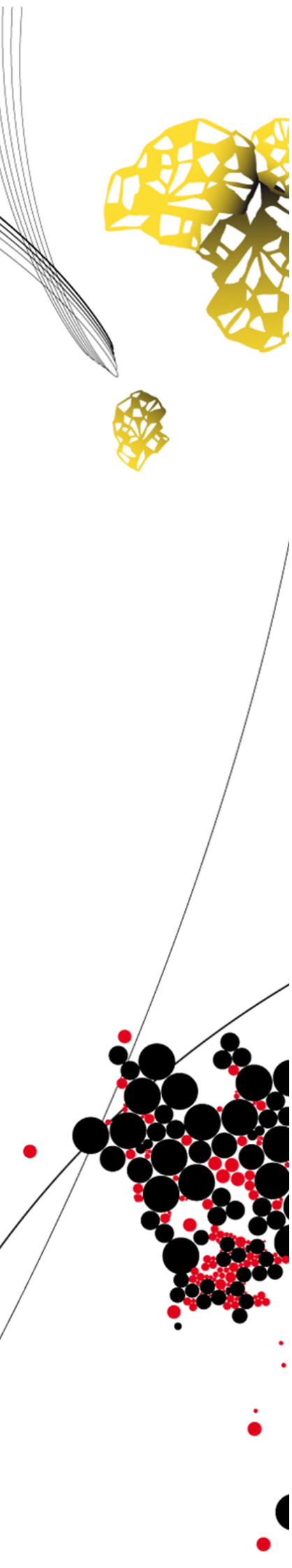

UNIVERSITY OF TWENTE.

**Faculty of Electrical Engineering,
Mathematics & Computer Science**

Modelling of Laser Attack Fault Injections on Field Programmable Gate Arrays

S. Nieuwenhuis

B.Sc. Thesis
June 2019

Supervisors:
dr.ing. D. M. Ziener
M. A. Sheikh
ir. E. Molenkamp
dr.ing. E. Tews bsc

Computer Architecture
for Embedded Systems
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Abstract

These days, software security algorithms are virtually unbreakable. As a result, the trend of tampering with hardware is on the rise. Hardware hacking focuses on bringing faults into a system and comparing the output of that tampered system to the output of the regular system. This Differential Fault Analysis can be used to read data streams inside of a chip. In Field Programmable Gate Arrays (FPGAs), the configuration bitstream that is stored on the chip can be recovered via these attacks. Possibly revealing confidential or sensitive data about the configuration of the FPGA.
The type of hardware tampering focused on in this work is Laser Fault Injection (LFI). These kinds of fault injections make use of the sensitivity of CMOS devices for charged radiation particles. Theoretically, it is possible to target single transistors using this technique, which makes it very effective. Modern technologies able to produce Static Random-Access Memory (SRAM) cells with a surface area lower than the size of a laser dot. As a result, neighbour cells could also be affected.
State-of-the-art laser setups necessary for these attacks are expensive and not readily available. This work will first present an electrical simulation of a laser attack on a single SRAM cell commonly used in the configuration memory of an FPGA. This is followed up by the layout and placement of these cells in a block RAM configuration. In this configuration, the regions sensitive to a bit flip, as well as unintended effects on nearby cells have been simulated and compared to measurements.

Contents

Chapter 1

Introduction

In the past few years, software encryption algorithms become good enough to prevent most attacks imposed by hackers. This development brings other weaknesses to the surface, like the security of the implemented hardware. An example is a number of security leaks found in processors in the past few years. More and more research is proving that electronic devices are vulnerable to all kinds of attacks. Chips like microcontrollers, System-on-Chips (SoCs) and Field-Programmable Gate Arrays (FPGAs) used for cryptography or security purposes can be attacked to reveal the data inside [1] [2] [3].

Such attacks can be performed for instance using side-channel analysis, where physical parameters like power draw are monitored to extract sensitive data [4]. This method however, is hard to aim at a specific area on the device, as usually the entire device is influenced by the attack. With Laser Fault Injection (LFI), an attack can be placed accurately by pointing a small laser dot on the desired part of a chip. The faulty output of the attacked system can be compared to the output of the unaffected circuit to extract sensitive information. This is called Differential Fault Analysis (DFA) [5].

Whilst very effective, LFIs suffer from a few inconveniences, since the silicon of the system must be exposed and the equipment necessary is expensive [1]. Precise laser setups are needed and a lot of preparation is needed. This report covers the development of a simulation of Laser Fault Injections on the configuration memory of a Field-Programmable Gate Array. The purpose of this simulation is to investigate the effect on memory cells surrounding the targeted cell. The simulation will be based on the results of measurements on 90 and 45 nm devices. The report will finish with a prediction on smaller technology nodes.

Chapter 2

Background

2.1 Field-Programmable Gate Array

Field-Programmable Gate Arrays (FPGAs) are configurable logic arrays, able to do logic computations very efficiently with more versatility than an Application Specific Integrated Circuit (ASIC), which is an integrated circuit designed for one application only. FPGAs are in that sense of flexibility more like software microprocessors, yet much faster in signal processing, due to its capability to operations parallel instead of the mostly serial workload on microprocessors.
The implementation of an FPGA is stored in a bitstream. This bitstream and thus the implementation could be confidential. At each start-up of the FPGA, the bitstream is loaded from a memory to configure the chip. This process is vulnerable to outside listeners. These could re-engineer the implementation and copy the system.

2.1.1 Architecture

FPGAs are built using three basic components, with configurable wires connecting all of them together. These components are:

- Configurable Logic Blocks (CLB)
- Block RAM (BRAM)
- Digital Signal Processing (DSP) blocks

Configurable Logic Block

A Configurable Logic Block (CLB) contains the core logic elements of the FPGA. It is a cluster of Basic logic elements (BLEs), which can be addressed individually via a set of multiplexers (MUX). Each BLE contains a Look-Up Table (LUT) or logic arrays for logic operations, a flip-flop for storage, and a MUX that select between the LUT and

the flip-flop [6].
In Xilinx FPGAs, this is implemented a little differently, with the use of so-called slices. In the 7-series and newer, a slice is configured as a set of four logic elements (LUTs in this case), eight storage elements, and some multiplexers and carry logic. These slices are called SLICEL. Xilinx CLBs can also be implemented with a SLICEM. A SLICEM contains a distributed RAM block instead of a LUT. This RAM can be configured as RAM, ROM, shift register or a LUT, whilst a LUT can only be used as ROM [7]. Each CLB contains either a set of SLICELs or a SLICEL and a SLICEM. The LUTs configuration bits are stored in SRAM cells [8].

Block RAM

The Block RAM (BRAM) in a FPGA is the second type of building block. It is placed separate from the CLBs, in the case of Xilinx in columns which span an entire clock region. BRAM is meant for the storage of larger sets of data than the storage elements in CLBs. Xilinx FPGAs since the 5-series come with 36 kb of storage per block, which could be split into two separate RAMS of 18 kb [9].
The individual memory cells in the BRAM are SRAM cells. These cells keep their value as long as power is applied, in contrast to DRAM, which needs to be constantly refreshed.

Digital Signal Processing Blocks

FPGAs are very powerful in parallel computations. This makes them a good fit for real-time digital signal processing (DSP), since different processes can be done side by side. CLBs are not very well-suited for these kinds of operations though. For instance floating point numbers are difficult to implement. This has an effect on the precision of the computations [10].
For the signal processing then, the third building block is a dedicated DSP block. These blocks consist of dedicated adders, multipliers and accumulators.

2.1.2 Design flow

The basic design flow for Xilinx FPGAs is as follows in both ISE and Vivado is as follows [11]:

1. Design an entry
2. Perform a behavioural simulation to check if the functionality is correct
3. Synthesize the design
4. Implement the design, assigning the hardware on the FPGA

5. Run a timing analysis to check for any physical limitations

6. Create the configuration bitstream and upload it to the FPGA

It is this last step that is of interest to attackers, since the bitstream contains the implementation of the FPGA. If it can be read, the configuration of the FGPA can be reverse engineered, leading to possible theft of designs.

2.2 Static Random-Access Memory

An Static Random-Access Memory (SRAM) is a steady state memory, meaning that is does not have to be refreshed to keep its data like Dynamic Random-Access Memory (DRAM) does. This results in a faster memory, but it is built out of more components, so it is more expensive to manufacture. This section touches on the design of such an SRAM cell, starting at the basic MOSFET theory.

2.2.1 MOSFET

Consider the NMOS p-n junction pictured in Figure 2.1. The drain and source are heavily doped n-type semiconductor. They are nested in a p-type substrate. When a positive voltage is applied to the gate relative to the source, a charge will build up in the p-substrate between the drain and source terminals, since the electrons are attracted to the insulator. This will make the p-type semiconductor behave more like an n-type, creating a conductive path between the drain and source terminals. This area is called the depletion region.

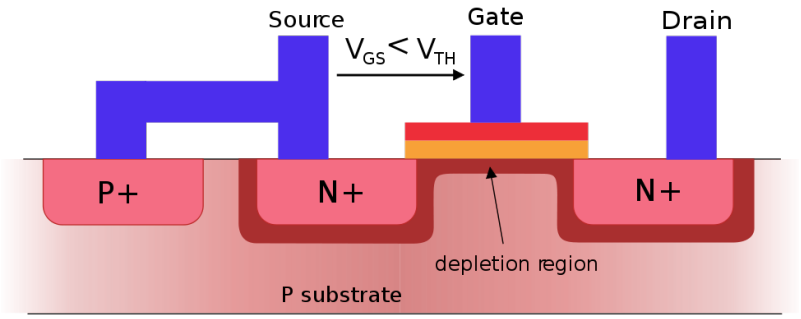

Figure 2.1: Cross-section of a NMOS [12]

2.2.2 CMOS technology

Complementary Metal-Oxide-Semiconductor (CMOS) technology is nowadays the standard for most integrated circuits, both digital and analog. CMOS combines p-type and n-type Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs) on the same p-substrate, sharing the V_{DD} and V_{SS} connections. The result is a circuit that has a very low static power draw compared to discrete p-type or n-type MOSFETs.
The p- and n-type MOSFETs themselves consist of a p-doped and an n-doped semiconductor respectively. P-type semiconductor materials have free flowing holes. N-type materials have free electrons. For NMOS, the source and drain are n-doped, embedded in a p-substrate. For PMOS, it is the other way around. To accommodate the PMOS in CMOS, a dedicated n-well is created in the p-substrate.

2.2.3 Inverters

Two of these CMOSs configured as inverters are the basis of a typical 6 transistor (6T) SRAM cell. Two enable transistors are added, such that the data is not transferred instantly to the output node, but only when the cell is selected. The inverters themselves are cross-coupled, see Figure 2.2.

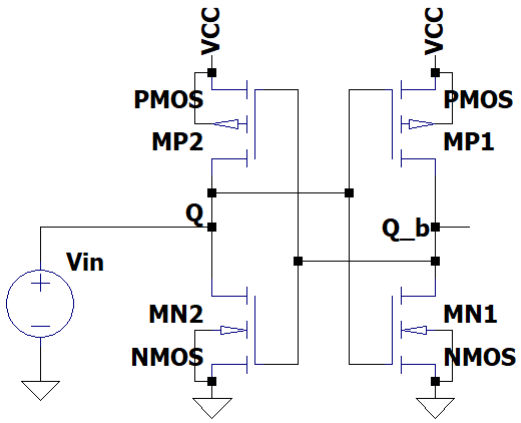

Figure 2.2: Cross-coupled CMOS inverters

Figure 2.3 shows that when a clock signal is applied to the input node (Q), the output (Q_b) changes instantly to the opposite of the input. When several of these cells are placed in a memory, arranged in parallel, the output would be unregulated. Also, the input value would be written to all cells at once. A multiplexer and demultiplexer could fix this, but those components add a lot of transistors - and thus costly surface area - to each cell. Also those might not be able to switch as fast as the cells themselves.

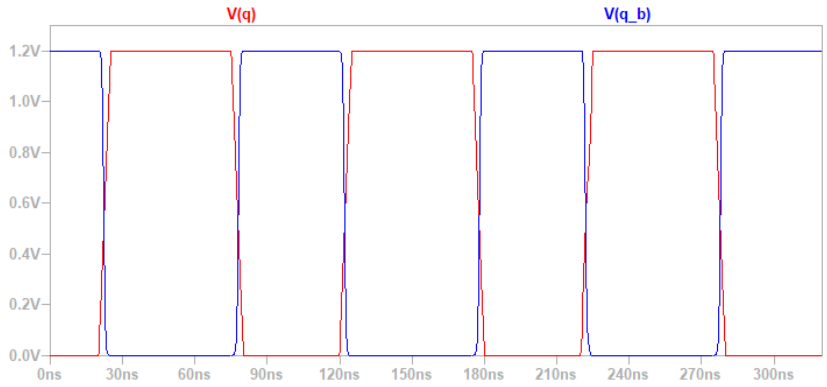

Figure 2.3: Input (red) and output (blue) of the cross-coupled inverters

2.2.4 SRAM cell

A better solution than MUXes is to synchronise the different memory cells by adding two MOSFETs. These word-select transistors are driven by the enable (EN) line. The first MOSFET enables the input, the second the output. This way, each cell can be assigned/read independently. The circuit is shown in Figure 2.4. This simulated the capacitance induced by the readout circuitry. The input has a source added, to assign values to the cell when the enable line (EN) is pulled high. This is shown in Figure 2.5 a) with the input ($V(q)$) pulled high. Figure 2.5 b) for an input pulled low.

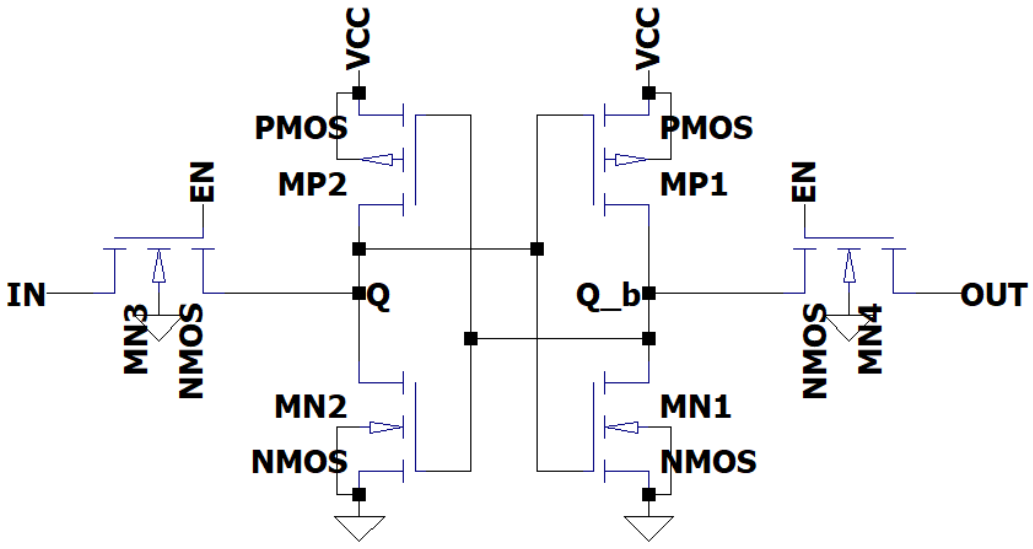

Figure 2.4: Circuit diagram of an 6T SRAM cell

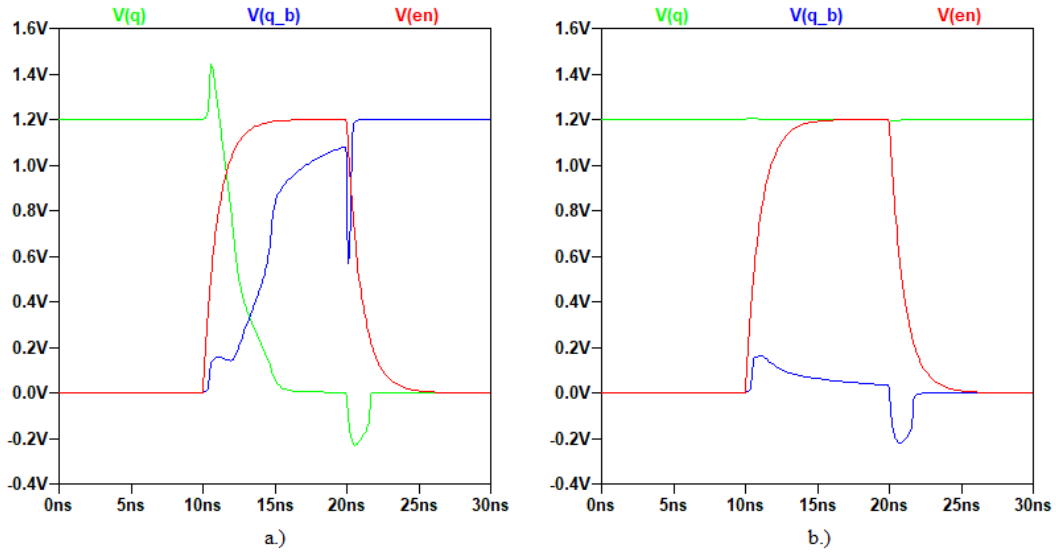

Figure 2.5: SRAM initialisation with values a.) 1 and b.) 0. Values are measured at the output. The output (blue) the inverse value of the input (green) when the enable signal (red) is pulled high.

2.3 Laser Fault Injection

A Laser Fault Injection (LFI) is a type of Single Event Upset (SEU) that is introduced by charged particles entering the depletion region of a MOSFET, thereby flipping the state of the transistor. Compared to other attack methods, LFI has a very high accuracy. Especially in older technology nodes it is possible to target a single MOSFET in, for instance, an SRAM cell. Because of the nanometer scale of MOSFETs nowadays, the surface area of the cell is often smaller than the size of a laser dot. Neighbour cells will be affected as a result. The accuracy is still relatively high compared to other attacks, which affect the entire device. Starting at the definition of a SEU, this section covers the theory behind LFIs.

2.3.1 Single Event Upset

A Single Event Upset is a temporary fault in a piece of hardware because a charged particle travels through a circuit, resulting in a so-called soft fail. A soft failure has no implications on the system after the SEU [13] has occurred. It only affects a device for a moment. This in contrast to hardware failures. Then the circuit is malfunctioning to the extent where it needs to be repaired. There are two kinds of soft errors:

- Transient faults, which are caused by influences outside of the system. A few causes are temperature, pressure, voltage or charge particles.

- Intermittent faults. These are caused by influences of the system itself. Think of bad connections or aging hardware.

This work is about laser attacks, which are transient faults. These kinds of upsets are also called Single Effect Transients (SETs).

2.3.2 Single Effect Transient in a MOSFET

Applying a voltage to the gate is not necessary to create a current in a MOSFET. A charged radiation particle can reach into the silicon [14]. When this particle travels through the p-n junction, it leaves behind a trail of both electrons and holes, as can be seen in Figure 2.6 a.). In the depletion region, these electrons create a charge difference (Figure 2.6 b.)), which decreases as a current starts to flow (Figure 2.6 c.)). The current created is called the photo current. When this photocurrent flows away from the node, it momentarily seems like the gate of the MOSFET is activated.

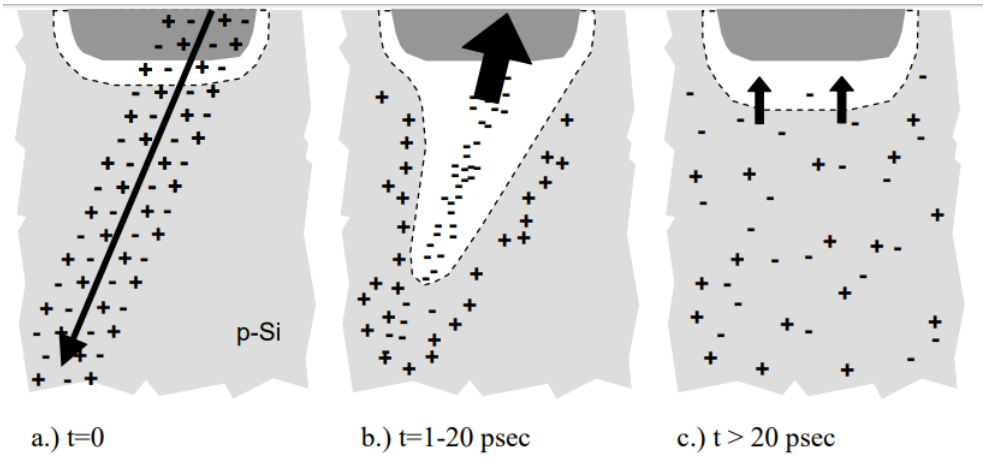

Figure 2.6: Charged particle travelling through a semiconductor node. a.) During strike, b.) charge distributing, c.) leftover charge collection. [14]

Chapter 3

Method

The effects of a laser attack have been simulated by creating two models; an electrical model and a physical model. The electrical model of an SRAM cell has been created in LTSpice, taking advantage of the power of SPICE simulation for electric circuits. The output of this model will be the current needed in each MOSFET to initiate a flip in the memory cell. The physical model contains the information and locations of MOSFETs in cells and cells in memory, as well as a model of the laser beam. This will be used to determine which cells will be affected by an attack.

3.1 Modelling the laser beam

The resulting photocurrent from a laser attack is dependent on the distance from the center of the dot and the size of the dot. These parameters are modelled into an object within MATLAB, to provide an interaction with the memory cells.

3.1.1 Photocurrent

The photocurrent induced by the laser beam crossing a MOSFET is determined by Sarafianos [15]. His experiments show that the photo current I_{ph} can be approximated by a first order polynomial, see Equation 3.1:

$$I_{ph} = a \cdot V + b \tag{3.1}$$

Where V is the bias voltage, and a and b constants depending on the laser power.

By measuring the I_{ph} for a constant voltage and a laser power of 0, 450 and 1300 mW, Sarafianos estimated a and b as [15]:

$$a = p \cdot {P_{laser}}^2 + q \cdot P_{laser} + r \tag{3.2}$$

$$b = s \cdot P_{laser} \tag{3.3}$$

For this work, the equipment to perform these measurements or the data from the measurements is not available. Therefore the values of p, q, r, and s are assumed to be the same as in [16], see Table 3.1. The feature size of the MOSFETs is the same in both this work as in [15] [16]. It is assumed that these findings are general for all 90 nm transistors and these equations can be used in this research as well.

Table 3.1: Values for the coefficients in equation 3.2

Coefficient	Value
p	$4E-9$
q	$-5E-7$
r	$9E-6$
s	$4E-6$

3.1.2 Laser spot

The intensity of the laser spot describes a Gaussian distribution [17], so it is possible to create a mathematical model of the dot, see equation 3.4

$$f(x) = \alpha \exp(-(x-\beta)^2/c^2) \tag{3.4}$$

Where a is a scaling factor, b is always 0 as there is no shift, c is the slope, and x is the distance to the center of the spot. For this simulation, it is desired to specify the spot size and get the intensity at a given distance. This spot size is defined as $x = 1/\exp 2$ [18]. Obtaining the slope of the curve is now a matter of solving equation 3.5 with $\alpha = 1$ and $\beta = 0$:

$$\exp(-2) = \exp(-x^2/c^2) \tag{3.5}$$

3.2 Targeting a memory cell

The memory is modelled in MATLAB with different classes. A class diagram can be found in figure 3.1. The full MATLAB code is available in Appendix A.

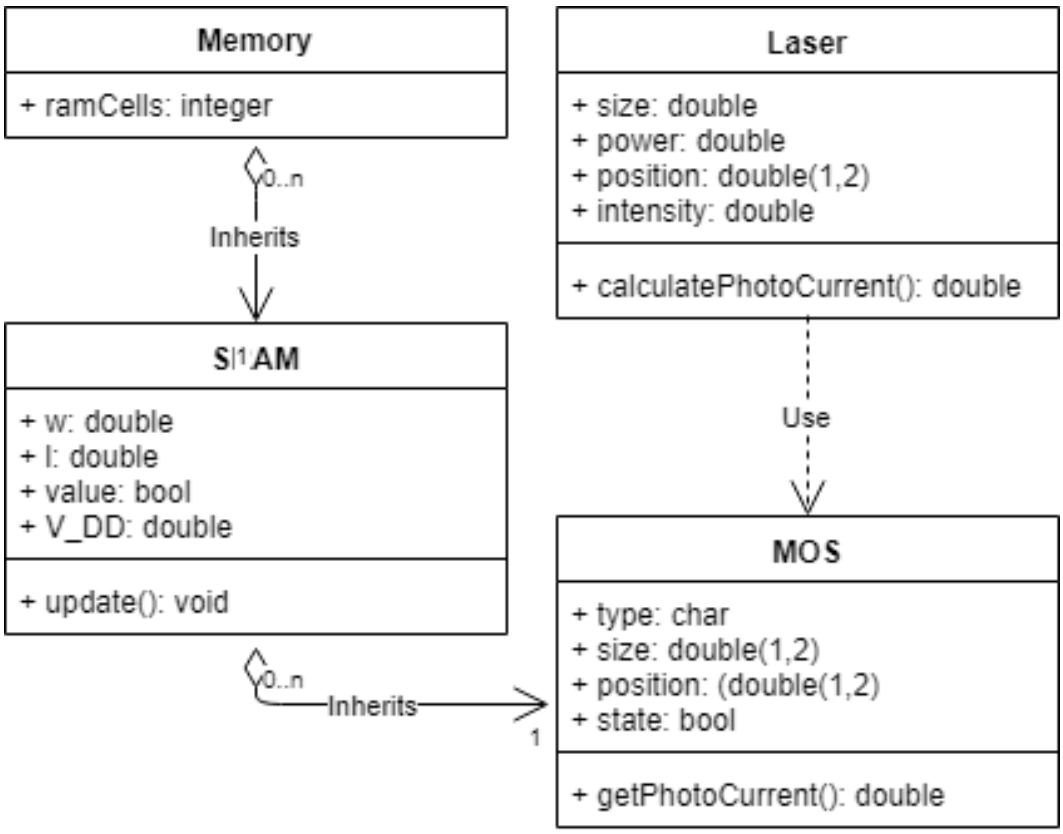

Figure 3.1: Class diagram of the model

3.2.1 Main file

In the main file of the script, a memory cell of a certain size and a laser are created. The `Memory` class then creates the necessary amount of instances of the `SRAM` class.

After the objects are created, a sensitivity map will be created. This done by moving the laser over the memory cells, calculating the photocurrent at certain intervals. If the current in a nearby junction is high enough, a memory flip will be recorded and stored.

3.2.2 Memory class

The `Memory` class does not much more than creating several SRAM cells and giving them a relative position to each other.

3.2.3 SRAM class

The most important variable in the `SRAM` class is `lambda`. According to the λ-design rules, the λ of a MOSFET is defined as half its minimum feature size [19]. Based on

this λ, all dimensions of the MOSFETS and the memory are created. This class also contains an update function, which will assign a value to the cell.

3.2.4 MOS class

Each MOSFET is based on the `MOS` class. It contains dimensions and position, as well as the state (on or off) that the MOSFETs is in. The state is assigned depending on the value of a SRAM cell. The function `getPhotoCurrent()` determines the closest distance to a laser target position and then calls the `Laser` class to calculate the photocurrent through the MOSFET.

3.2.5 Laser class

A dedicated class called Laser has been made in the MATLAB script to model the properties of the laser beam and its interaction with the memory. An object is made by specifying the size of the dot, the power of the laser, and the target position in an x and y coordinate.
The object has one function, `laser.calculatePhotoCurrent(d, V)`. Where `d` is the distance from the observing object to the laser and `V` the supply voltage of the observing object. The function returns the photocurrent the observing object endures.

3.3 Determining a flip

With the relation between the laser power and the resulting photocurrent addressed in Section 3.1. For this section, the focus will be on the amount of current needed and where this current needs to be injected in order to flip a bit.

3.3.1 SPICE simulation

With an expression for the photocurrent when the laser is aimed directly at a MOSFET, the necessary current to induce a flip can be determined. Whilst MOSFETs can be described by equations for the drain current depending on threshold and gate-source voltages, there are too many unknowns and dependability on parasitic components to calculate this by hand.
This is why the choice has been made to simulate an SRAM cell with a Simulation Program with Integrated Circuit Emphasis (SPICE) simulation. This has been done in LTSpice, since it was available and in the time frame of this work, getting familiar with something like Cadence or ELDO would not be possible within the available timeframe. The drawback of LTSpice is that it does not support physical interfaces, so only an electrical model could be made.

3.3.2 Predictive Technology models

The MOSFETs used are models from the ASU Predictive Technology Model (PTM) database [20]. These MOSFET models were used because the TSMC produced transistors used in Xilinx FPGAs are proprietary and not available to the public, like all CMOS components. This will naturally introduce some errors in the model. The length has been set to the minimum feature size of 2λ, in this case 90 nm. The width has been set to 3λ for the PMOS and 6λ for the NMOS, as is common for SRAM cells [21].

3.3.3 Placement

As discussed in Section 2.3.2, the photocurrent is induced in either the source or the drain of a MOSFET. In the case of an NMOS, the n-type semiconductor material will generate an excess of electrons, which will produce a positive charge in the depletion region, resulting a current to the ground node. This can be modelled by a current source between either the drain or the source of the transistor, leading to ground (see Figure 3.2, I4 and I3).
For PMOS, the opposite is true, as the p-type semiconductor in the nodes will result an excess of holes. The resulting current will be pulled from a common node, for instance V_{DD} in Figure 3.2 I2 and I1.
Because an NMOS and a PMOS use electrons and holes as carriers respectively, the width of a PMOS should be about three times the width of an NMOS [22]. This larger gate area results in a lower sensitivity to photocurrents.
While both the drain and source of a MOSFET can be hit by a laser, only the drain allows for a flip [23]. In the SPICE model then, current sources will be modelled only the drains of the transistors.

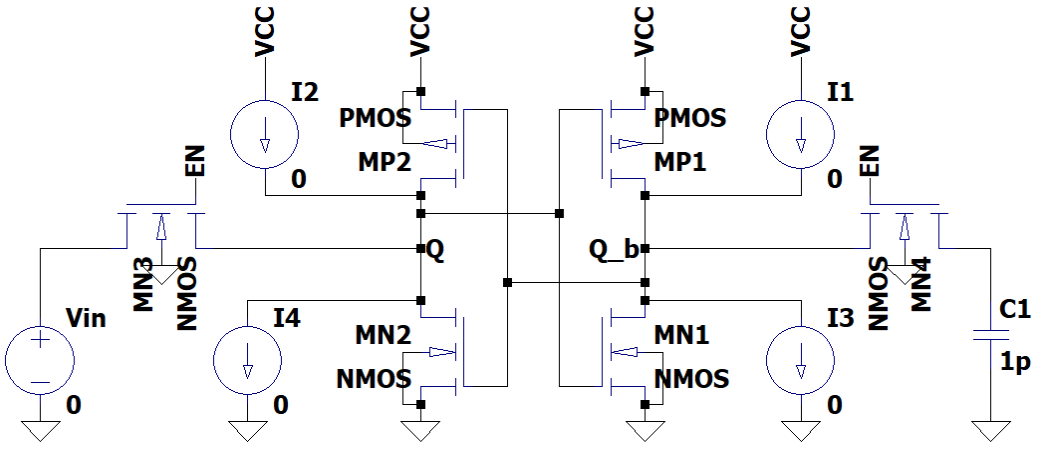

Figure 3.2: Circuit diagram of an SRAM cell with the current sensitive zones indicated by current sources.

3.3.4 Current pulse

As a capacitive charge is built and discharged in the silicon of the target transistor, a current pulse would have an exponential shape. From previous work, a duration of about 50 ns should be sufficient in order to initiate a flip [15]. This is with an amplitude of 100 μA.

3.4 Creating a memory layout

There are many different ways to implement 6T SRAM. The most common type for a gate length $>$ 90 nm is the so-called type 2. For newer processes, type 4 is more commonly used [21].
Assuming the W/L ratio is constant as a MOSFET scales and the polysilicon (red) in Figure 3.3 is the same length as the gate of each MOSFET, the dimensions and the spacing of the transistors can be estimated. These values have been given to the MOS objects in MATLAB.

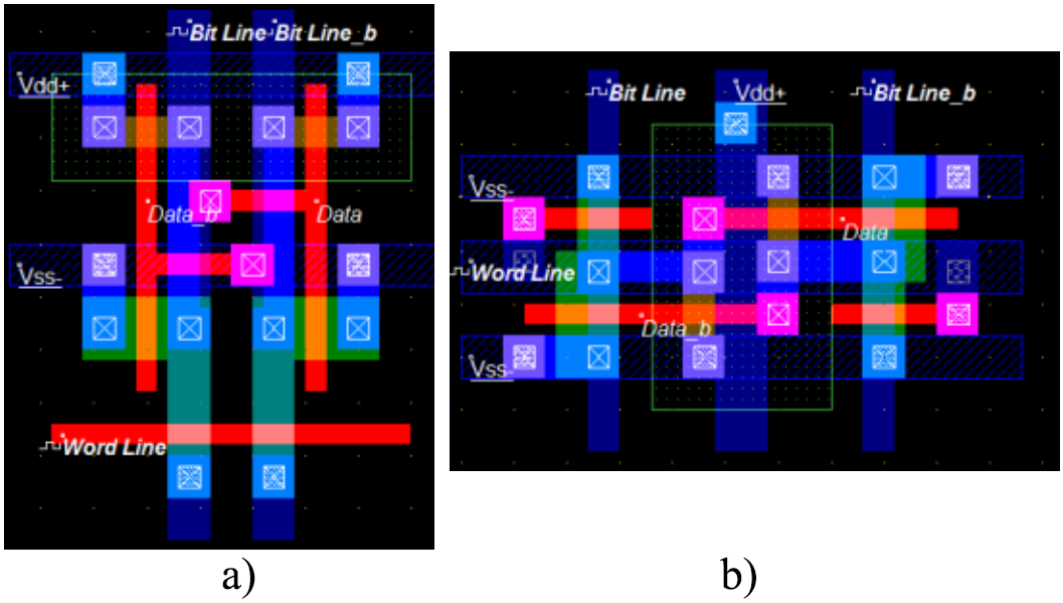

Figure 3.3: a) Type 2 SRAM, b) Type 4 SRAM (adapted from [21])

Chapter 4

Results

With the models, three simulations have been performed. All simulations will be done under the same conditions as the research in [18], so the results of the simulation can be compared to an actual measurement. The first simulation focuses on flipping a single memory cell, finding if the current pulse of the measurements produces results in the simulation as well. The second simulation is about indicating sensitive regions and comparing them with measurements to validate the accuracy of the model. Finally, the model is used to predict the effects for even smaller gate lengths than measured in [18] to try and predict what would happen at 28 nm. The results of those simulations are discussed in this section. When the value of a flipped MOSFET or SRAM cell changed from 0 to 1, it is indicated as a bit set. Similar, 1 to 0 is called a bit reset.

4.1 Flipping a single cell

A transient simulation has been performed in LTSpice with the circuit in Figure 3.2. The duration of the current pulse is 800 ps, as in [18] with a rise- and fall-time of 50 ps to accommodate for the capacitive charge and discharge happening in the depletion region. An amplitude of 200 μA flips the cell at the drain of a PMOS, an NMOS flips at 55 μA. Figures 4.1, 4.2, 4.3, and 4.4 show the results for all four state MOSFETs in a 6T SRAM cell.

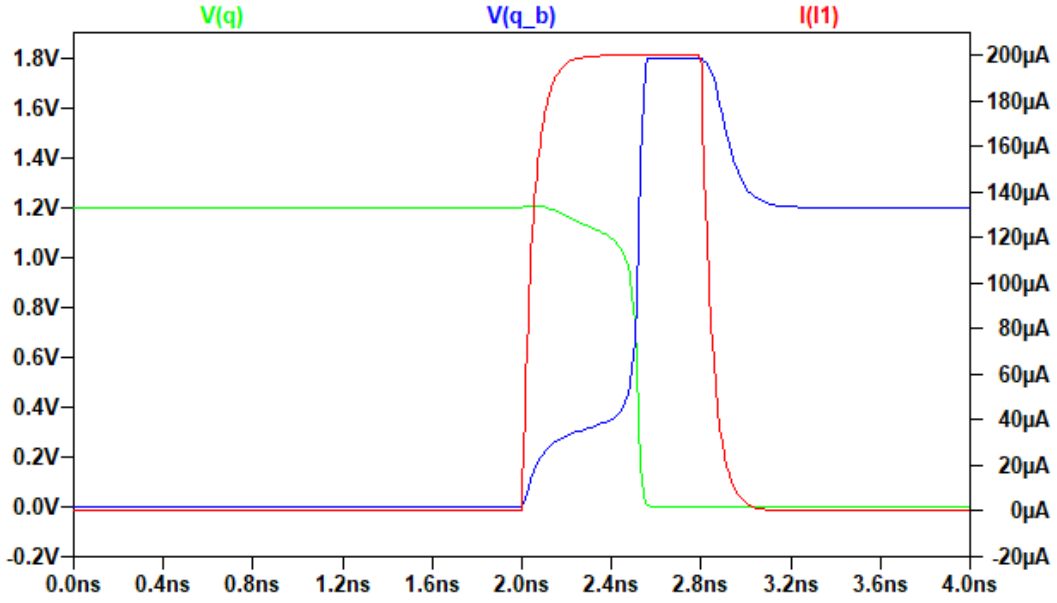

Figure 4.1: Attack on MP1, bit set

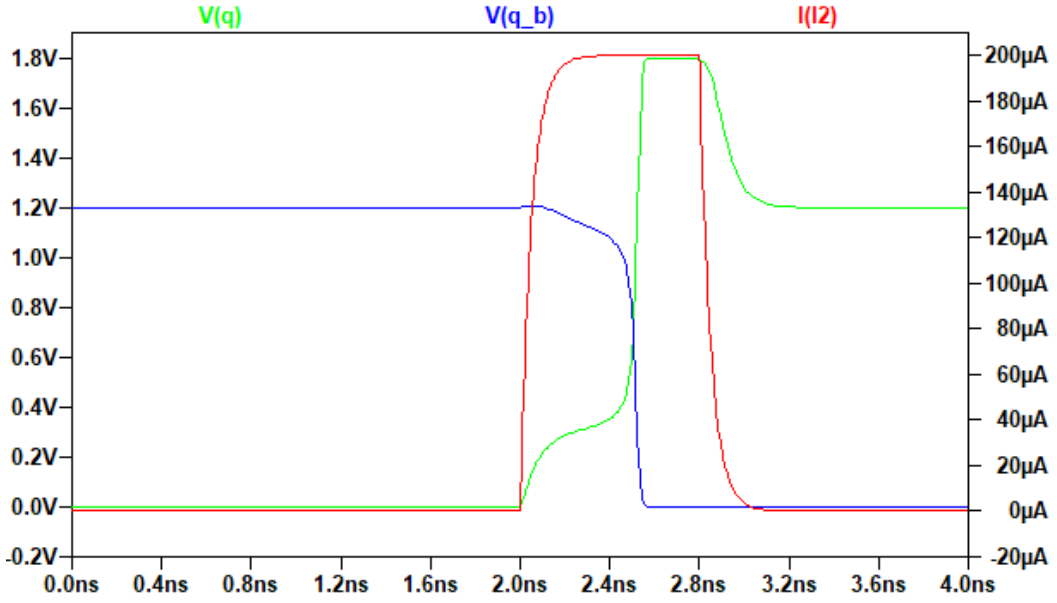

Figure 4.2: Attack on MP2, bit set

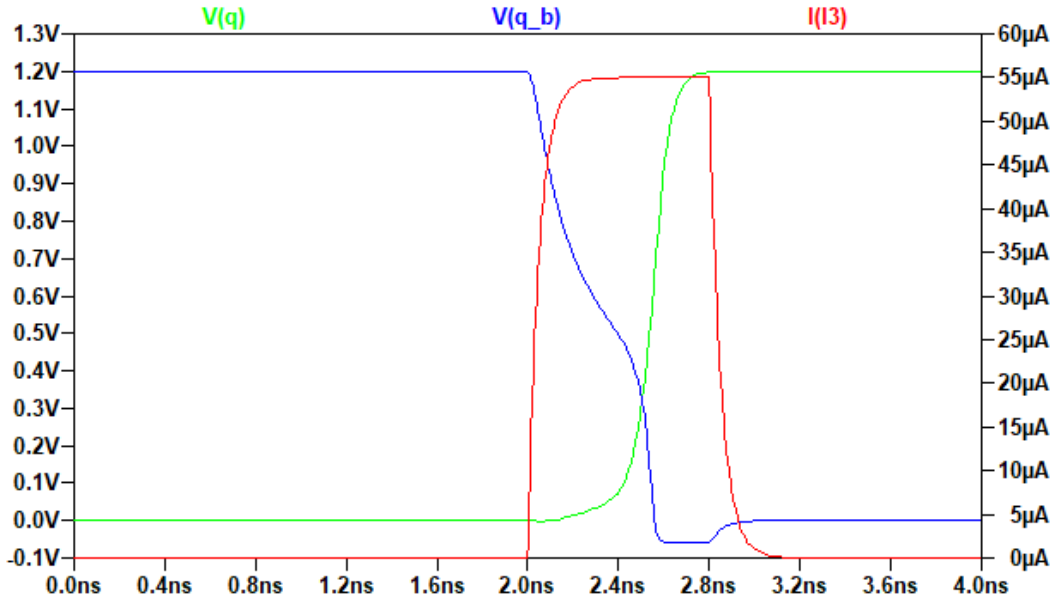

Figure 4.3: Attack on Mn1, bit set

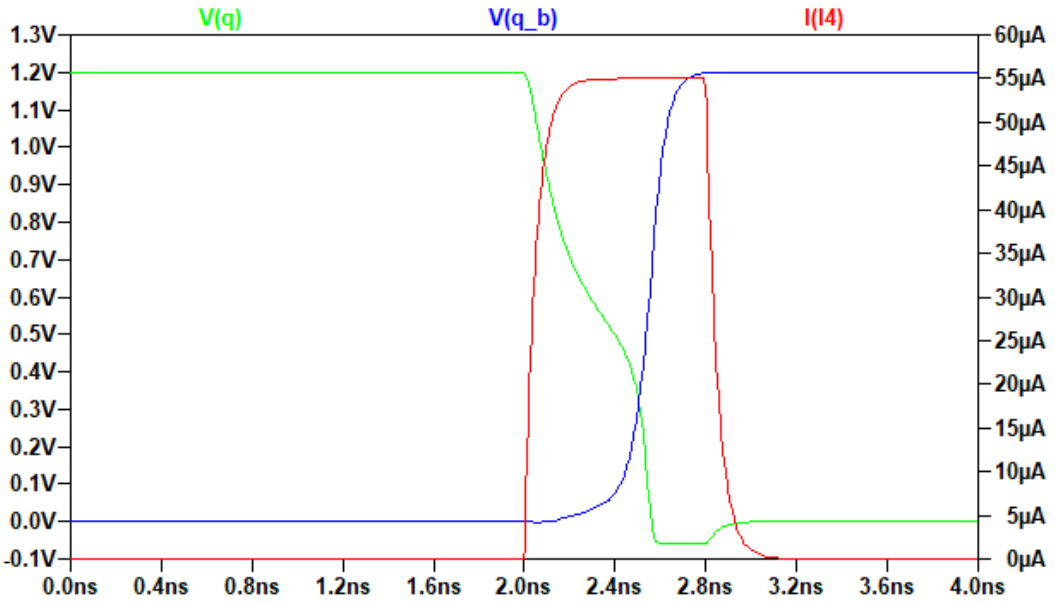

Figure 4.4: Attack on MN2, bit set

When two MOSFETs are hit by an attack, the bit will not flip, see Figure 4.5. This has only been performed on MN1 and MP1 as a proof of concept.

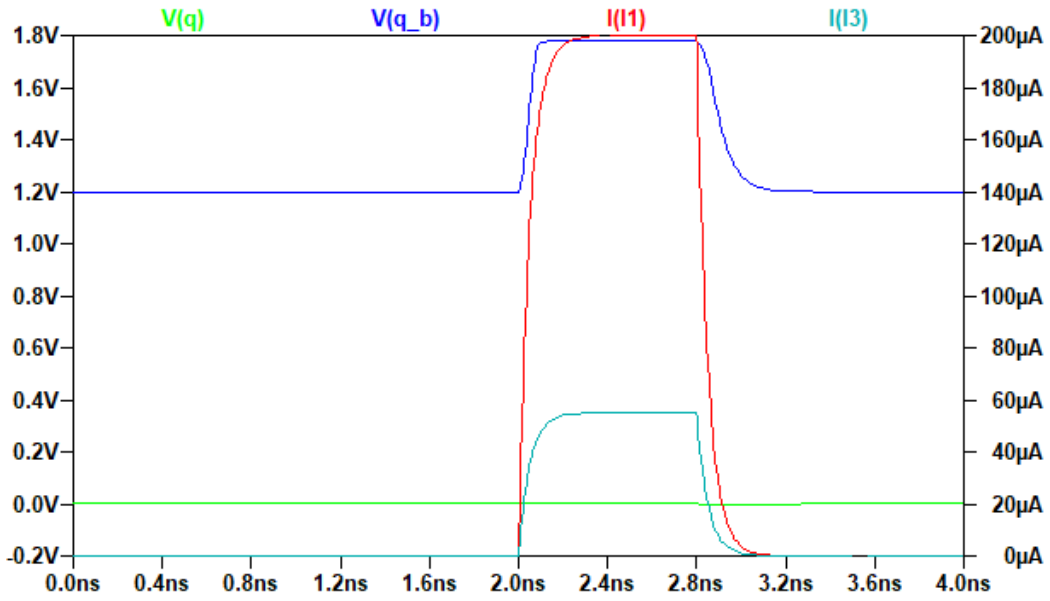

Figure 4.5: Double MOSFET hit by an attack, no flip

4.2 Showing sensitive regions

4.2.1 Measurements

The measurements in [18] show a part of a memory block. This block can be seen in the right plot of Figure 4.7. To obtain these results, the value of each cell has been set to 1. Then the laser would be placed on the chip and an attack would be performed. When the value of the cells has been read and any cell changed from 1 to 0 (also called bit reset), this position for the laser is indicated by a blue color. Then, the laser is moved 200 nm and the same thing is done over and over until the entire area has been scanned. When this is done, the same thing is repeated with all cells set to 0. Now every change in value is indicated in red, which indicates a bit set.

4.2.2 Simulation for 90 nm

The measurements in [18] have been performed on a Xilinx Spartan 3a with a gate length of 90 nm. The second measurements have been done with a Spartan 6, using 45 nm long gates. The sensitive regions of an SRAM cell have been visualised by moving a laser on a grid over a block of memory. Just like in Seciton 4.2.1, at each point is measured whether a cell in the memory has been flipped. The same settings as in Section 4.1. A high-level layout of the grid of cells that has been simulated is given in Figure 4.6

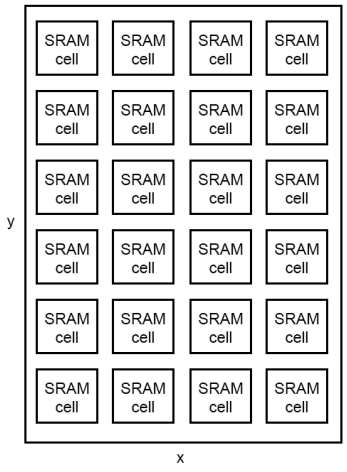

Figure 4.6: High-level layout of the measured array of SRAM cells

There are some differences between the layout of the results in Figure 4.7. The faultless spaces in the measurements are larger than in the measurement, while the faults appear to be closer. Also, two flip zones are located next to each other in the measurement. This indicates that the cells are mirrored next to each other, which

is in line with [21]. Furthermore, the model is less random than the measurement, indicating that some cells are more or less sensitive to laser injections.

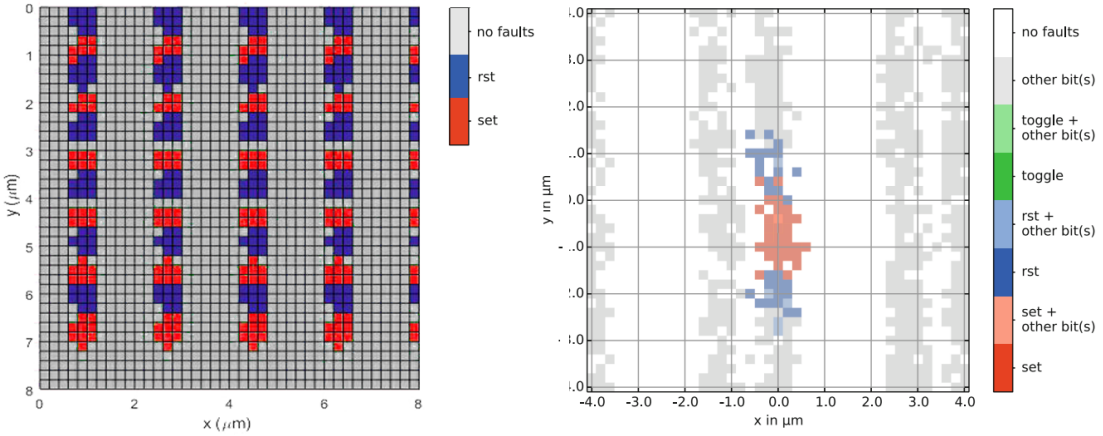

Figure 4.7: Comparison between 90 nm simulation (left) and measurement (right, adjusted from [18])

4.2.3 Simulation for 45 nm

A second simulation has been performed at 45 nm at a Xilinx Spartan 6. From the LTSpice model, only 150 μA is necessary for a flip in a PMOS, and 60 μA is needed for a flip in an NMOS. This is higher than the 90 nm model (see Figure 4.8). The grid consists of 8 by 8 SRAM cells, spanning an area of about 7.5 μm in width and almost 4 μm in height.

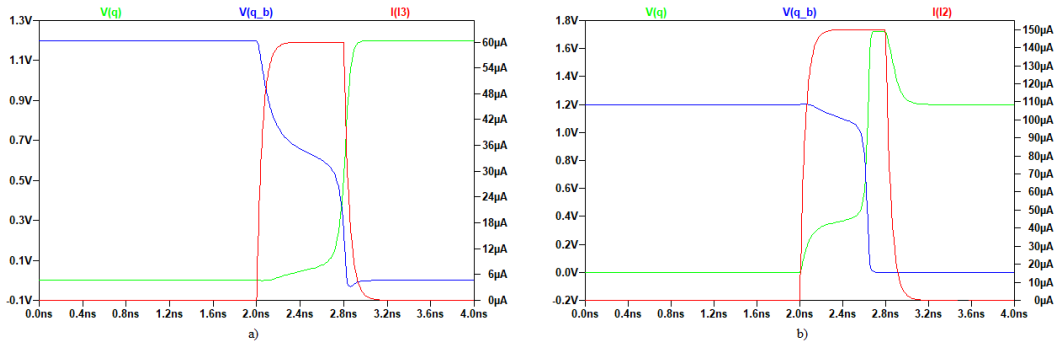

Figure 4.8: a) Bit flip on NMOS, b) Bit flip on PMOS for 45 nm

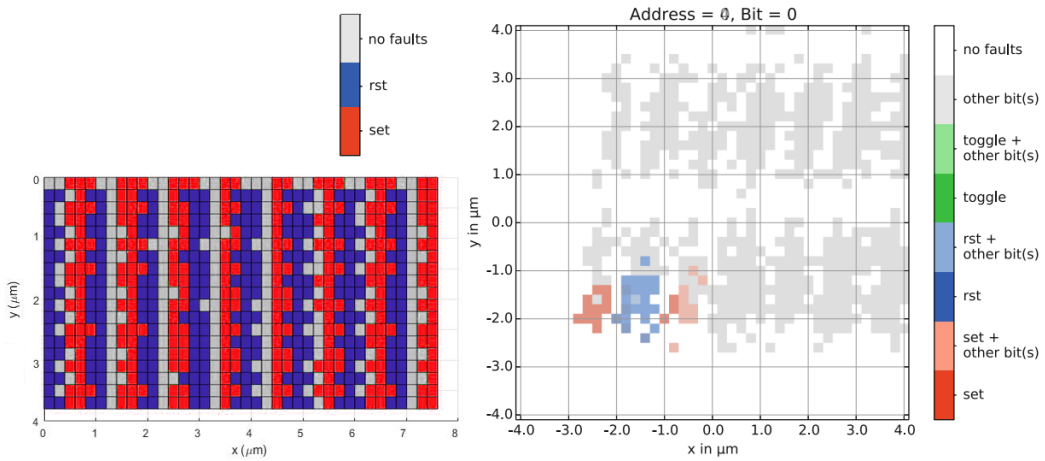

Figure 4.9: Comparison between 45 nm simulation (left) and measurement (right, adjusted from [18])

Figure 4.9 shows that compared to the 90 nm model, the locations where bits are flipped occur more randomly, yet still not as randomized as the measurements. Once again, the cells seem to be mirrored to each other, the white line in the middle of the measurements being the bit select lines.

4.3 Smaller technology nodes

The 22 nm LTSpice models resulted in a bit flip at 20 μA on the NMOS or 50 μA in case of the PMOS. The signals are plotted in Figure 4.10 From Figure 4.11 it can be seen that every point on the chip is sensitive for a laser injections, either set or reset. Notice in Figure 4.11 that an array of 8 by 8 cells is smaller than the 4 μm cross section of the laser dot.

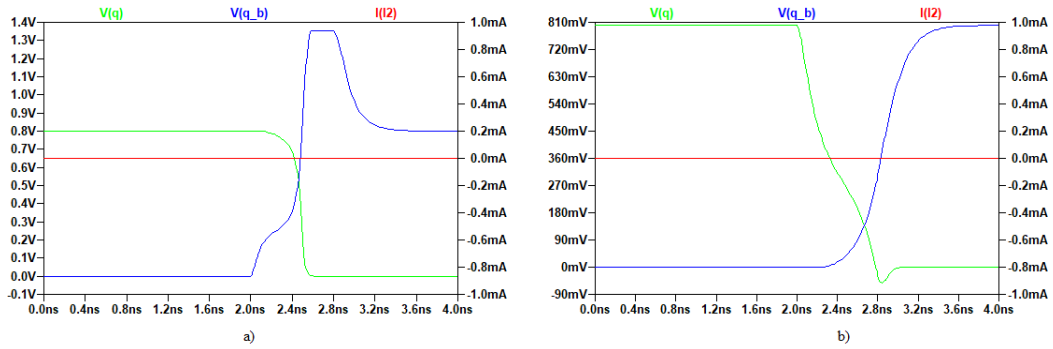

Figure 4.10: For 22 nm CMOS, a) bit set on PMOS, b) bit set on NMOS

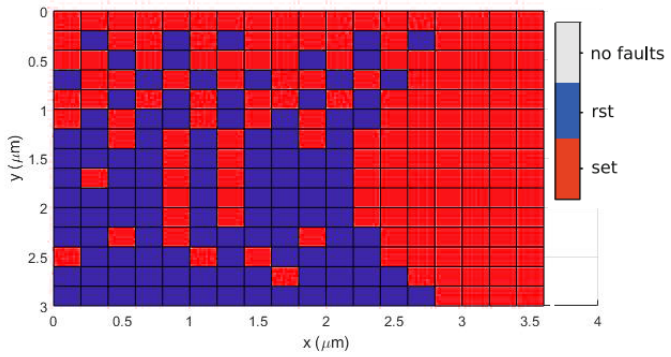

Figure 4.11: Sensitive regions for a 28 nm SRAM cell

Chapter 5

Conclusion

The goal of this research was to simulate the effects of a Laser Fault Injection on the SRAM in FPGAs. This has been implemented using LTSpice and MATLAB, covering the electrical and physical properties of laser tampering respectively.
Starting with the electrical simulation, despite the unavailability of accurate models for the MOSFETS, the PTM models were a proper replacement. The characteristics were similar to other research, even for different technologies. A bridge to a physical model would have been nice, since determining the layout of the memory cells is not ideal in the current implementation.
Given that it is not possible to rotate an SRAM cell in the MATLAB, there are some inaccuracies with the simulation compared to the real-world measurements. Also, it became apparent in Figure 4.7 that there is variation between the sensitivity of the cells. The model does not take these inaccuracies into account. Other than that, the different bit flip locations to resemble the basic behaviour seen in the measurements.
Ideally, the MATLAB and SPICE simulations would be integrated. This would benefit the usability of these kinds of simulations. Now the user has to run the SPICE model, make sure the different signals, voltages, and currents are the same in MATLAB and LTSpice. Due to time constraints this was not possible to implement however.
To conclude, the simulation has very basic functionality. It would certainly benefit if some more time was put into it, to iron out the few imperfections still available.

Chapter 6

Recommendation

For future research, there are several directions to improve these simulations. First of all, the script must be updated such that memory cells can be rotated and mirrored. Adding a random factor to the MOSFET characteristics could produce even better results compared to the measurements.

For the smaller gate lengths, some measurements could be done to verify that the simulation is accurate at those technology nodes as well. For instance, the 22 nm measurement is just an estimation based on larger production nodes.

Finally, a piece of code could be added to manipulate the configuration bitstream of an FPGA. This way, the FPGA would be configured as if it was attacked by a laser. This could open further research tracks for those who do not have a LFI setup available.

Bibliography

[1] S. Skorobogatov and R. Anderson, "Optical fault induction attacks," *Kaliski Jr., B.S., Ko, .K., Paar, C. (eds.) CHES 2002. LNCS*, vol. 2523, pp. 2–12, 2003.

[2] A. Vaselle, H. Thiebeauld, Q. Maouhoub, A. Morisset, and S. Ermeneux, "Laser-induced fault injection on smartphone bypassing the secure boot," *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2017.

[3] D. Ziener, J. Pirkl, and J. Teich, "Configuration tampering of bram-based aes implementations on fpgas," *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2018.

[4] M. Zhao and G. Suh, "Fpga-based remote power side-channel attacks," *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

[5] M. Agoyan, J. Dutertre, A. Mirbaha, D. Naccache, A. Ribotta, and A. Tria, "Single-bit dfa using multiple-byte laser fault injection," *2010 IEEE International Conference on Technologies for Homeland Security (HST)*, 2010.

[6] Xilinx, "7 series fpgas configuration user guide, ug470(v1.13.1)," 2018. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf

[7] edaboard Forum, "What is the distributed ram ?" 2006. [Online]. Available: https://www.edaboard.com/showthread.php?81168-What-is-the-Distributed-RAM

[8] Z. Seifoori, Z. Ebrahimi, B. Khaleghi, and H. Asadi, "Chapter seven - introduction to emerging sram-based fpga architectures in dark silicon era," *Advances in Computers*, vol. 110, pp. 259–294, 2018.

[9] Xilinx, "7 series fpgas memory resources user guide, ug473(v1.13)," 2019. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf

[10] ——, "7 series dsp48e1 slice user guide, ug479(v1.10)," 2018. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

[11] "Fpga basic flow," Accessed 24-06-2019. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx10/isehelp/ise_c_basic_flow.htm

[12] O. Deleage and P. Scott, "Mosfet functioning," 2008. [Online]. Available: https://commons.wikimedia.org/wiki/File:MOSFET_functioning_body.svg

[13] F. Wang and V. Agrawal, "Single event upset: An embedded tutorial," *21st International Conference on VLSI Design (VLSID 2008)*, 2008.

[14] R. Baumann, "Soft errors in commercial integrated circuits," *International Journal of High Speed Electronics and Systems*, vol. 14, pp. 229–309, 2004.

[15] A.Sarafianos, C.Roscian, J.-M.Dutertre, M.Lisart, and A.Triab, "Electrical modeling of the photoelectric effect induced by a pulsed laser applied to an sram cell," *Microelectronics Reliability*, vol. 53, pp. 1300–1305, 2013.

[16] A.Sarafianos, R.Llido, J.M.Dutertre, O.Gagliano, V.Serradeil, M.Lisart, V.Goubier, A.Tria, V.Pouget, and D.Lewis, "Building the electrical model of the photoelectric laser stimulation of a pmos transistor in 90 nm technology," *Microelectronics Reliability*, vol. 52, pp. 2035–2038, 2012.

[17] C.Godlewski, V.Pouget, D.Lewis, and M.Lisart, "Electrical modeling of the effect of beam profile for pulsed laser fault injection," *Microelectronics Reliability*, vol. 49, pp. 1143–1147, 2009.

[18] B. Selmke, S. Brummer, J. Heyszl, and G. Sigl, "Precise laser fault injections into 90 nm and 45 nm sram-cells," *Homma N., Medwed M. (eds) Smart Card Research and Advanced Applications.*, vol. 9514, 2016.

[19] "Nmos cmos inverters and gates lamda base rule." Accessed 23-06-2019. [Online]. Available: http://ggn.dronacharya.info/EEEDept/Downloads/QuestionBank/VISem/VLSI_design/Section-B.pdf

[20] "Ptm - introduction." [Online]. Available: http://ptm.asu.edu/

[21] D. Balobas and N. Konofaos, "Design and evaluation of 6t sram layout designs at modern nanoscale cmos processes," *4th MOCAST Conference*, 2015.

[22] "Ioffe institute database, silicon electrical properties," Accessed 23-06-2019. [Online]. Available: http://www.ioffe.ru/SVA/NSM/Semicond/Si/electric.html

[23] C. Roscian, A. Sarafianos, J. Dutertre, and A. Tria, "Fault model analysis of laser-induced faults in sram memory cells." *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 89–98, 2013.

Appendix A

MATLAB code

A.1 Main script

```
%% Define parameters
res = 0.2;                % Laser change intervals, um

% RAM cell specs
V_DD = 1.2;               % Memory supply voltage, 90/45 : 1.2, 22 : 0.8 (V)
init_value = 1;           % memory inital value
numCells = [8,8];         % number of SRAM cells
gateSize = 0.09;          % in um
min_flipCurrent = 0.055; % 90nm : 0.055, 45nm : 0.060, 22nm : 0.020
max_flipCurrent = 0.2;    % 90nm : 0.200, 45nm : 0.150, 22nm : 0.050

% Laser params
intensity = 5.2;
spread = 4;               % Spot size in um
power = 1250;             % 10 nJ / 800 ps = 1250 (mW)
position = [0,0];         % Target position

%% Create objects
ramMemory = Memory(numCells, V_DD, init_value, gateSize);
laserBeam = Laser(intensity, spread, power, position);

%% Create sensitivity map
flipmap = zeros(floor(ramMemory.ramCells{1,1}.w/res*numCells(2)+1),
        floor(ramMemory.ramCells{1,1}.l/res*numCells(1)+1));
flipmap(:) = init_value;
```

```
% Loop through all memory cells and mosfets, while placing the laser at
% different posistions
for j = 1:length(flipmap(1,:))
    for i = 1:length(flipmap())
        laserBeam.position = [i*res, j*res];
        for n = 1:numCells(1)
            for m = 1:numCells(2)
            for k = 1:4
                % Calculate currents
                I_ph(1) = ramMemory.ramCells{n,m}.MN1.getPhotoCurrent(laserBeam);
                I_ph(2) = ramMemory.ramCells{n,m}.MN2.getPhotoCurrent(laserBeam);
                I_ph(3) = ramMemory.ramCells{n,m}.MP1.getPhotoCurrent(laserBeam);
                I_ph(4) = ramMemory.ramCells{n,m}.MP2.getPhotoCurrent(laserBeam);
                % Double if statement to make sure not 2 MOSFETs in one
                % inverter are targeted
                if I_ph(1) > min_flipCurrent
                    if I_ph(3) < max_flipCurrent
                        flipmap(i,j) = 0; % Bit reset
                    end
                end
                if I_ph(2) > min_flipCurrent
                    if I_ph(4) < max_flipCurrent
                        flipmap(i,j) = 2; % Bit set
                    end
                end
                if I_ph(3) > max_flipCurrent
                    if I_ph(1) < min_flipCurrent
                        flipmap(i,j) = 2; % Bit set
                    end
                end
                if I_ph(4) > max_flipCurrent
                    if I_ph(2) < min_flipCurrent
                        flipmap(i,j) = 0; % Bit reset
                    end
                end
            end
            end
        end
    end
```

```
end
%% Plot Sensitivity map

figure(1)
X = 0:res:length(flipmap(1,:))*res-res;
Y = 0:res:length(flipmap(:,1))*res-res;
surf(X,Y,flipmap)
xlabel('x (\mum)')
ylabel('y (\mum)')
view(0,270)
```

A.2 Laser class

```
classdef Laser
    %LASER

    properties
        intensity;
        size;
        power;
        position = zeros(1,2);
    end

    methods
        function laser = Laser(i, s, P, r)
            %LASER Construct an instance of this class
            laser.intensity = i;
            laser.size = s;
            laser.power = P;
            laser.position = r;
        end

        function I = calculatePhotoCurrent(laser, d, V)

            % Determine slope from spot size. Assuming the light
            % distribution from a laser is Gaussian [y = a*exp((x-b)/c)]
            % and the spot size is given as [1/exp(2)], the coefficients
            % can be determined. a = i, b = 0 and
```

```
            c = sqrt((laser.size/2)^2/2);

            % Coefficients for the power to current conversion, determined from
            % measurements
            p = 4e-9;
            q = -5e-7;
            r = 9e-6;
            s = 4e-6;

            % Calculate current amplitude
            a = p .* laser.power.^2 + q .* laser.power + r;
            b = s .* laser.power;

            I_max = (a * V + b)*laser.intensity;

            % Calculate current distribution over distance
            I = I_max.*exp(-d.^2./c^2);
        end
    end
end
```

A.3 MOS class

```
classdef MOS
    %MOS
    properties
        position = zeros(1,2);
        size = zeros(1,2);
        type;
        id;
        V_DD;
        state = 0;
    end

    methods
        function mos = MOS(id, type, pos, s, v, val)
            %MOS
            mos.id = id;
            mos.type = type;
```

```
            mos.size = s;
            mos.position = pos;
            mos.V_DD = v;
            mos.state = val;
        end

        function I_ph = getPhotoCurrent(mos, laser)

            x_laser = laser.position(1);
            y_laser = laser.position(2);

            x_end = mos.position(1) + mos.size(1);
            y_end = mos.position(2) + mos.size(2);

            % Find shortest path to laser x
            if x_laser < mos.position(1)
                x_mos = mos.position(1);
            elseif x_laser > x_end
                x_mos = x_end;
            else
                x_mos = x_laser;
            end

            % Find shortest path to laser y
            if y_laser < mos.position(2)
                y_mos = mos.position(2);
            elseif y_laser > y_end
                y_mos = y_end;
            else
                y_mos = y_laser;
            end

            % Calculate distance
            d = sqrt(abs(y_mos-y_laser).^2 + abs(x_mos-x_laser).^2);

            % Calculate photocurrent
            I_ph = laser.calculatePhotoCurrent(d, mos.V_DD);
        end
    end
```

```
end
```

A.4 SRAM class

```
classdef SRAM
%SRAM

properties
    lambda = 0;
    w = 0;
    l = 0;
    value = 0;
    V_DD = 0;
    MP1;
    MP2;
    MN1;
    MN2;
    MN3;
    MN4;
end

methods
    function sram = SRAM(v, val, offset, gs)
        %SRAM
        sram.lambda = gs/2;
        sram.value = val;
        sram.V_DD = v;

        % use type 2 layout for nodes larger/equal than 90 nm
        % and type 4 for smaller than 90 nm
        if sram.lambda < 0.09/2
            sram.w = 22*sram.lambda;
            sram.l = 43*sram.lambda;
            sram.MP1 = MOS(1, 'p', [offset(1)*sram.w+18*sram.lambda,
                offset(2)*sram.l+16*sram.lambda], [2*sram.lambda,
                3*sram.lambda], sram.V_DD, val);
            sram.MN1 = MOS(2, 'n', [offset(1)*sram.w+18*sram.lambda,
                offset(2)*sram.l+4*sram.lambda], [2*sram.lambda,
                6*sram.lambda], sram.V_DD, ~val);
```

```
                sram.MP2 = MOS(3, 'p', [offset(1)*sram.w+9*sram.lambda,
                    offset(2)*sram.l+24*sram.lambda], [2*sram.lambda,
                    3*sram.lambda], sram.V_DD, ~val);
                sram.MN2 = MOS(4, 'n', [offset(1)*sram.w+9*sram.lambda,
                    offset(2)*sram.l+33*sram.lambda], [2*sram.lambda,
                    6*sram.lambda], sram.V_DD, val);
            else
                sram.w = 26*sram.lambda;
                sram.l = 40*sram.lambda;
                sram.MP1 = MOS(1, 'p', [offset(1)*sram.w+4*sram.lambda,
                    offset(2)*sram.l+5*sram.lambda], [3*sram.lambda,
                    2*sram.lambda], sram.V_DD, val);
                sram.MN1 = MOS(2, 'n', [offset(1)*sram.w+4*sram.lambda,
                    offset(2)*sram.l+22*sram.lambda], [6*sram.lambda,
                    2*sram.lambda], sram.V_DD, ~val);
                sram.MP2 = MOS(3, 'p', [offset(1)*sram.w+20*sram.lambda,
                    offset(2)*sram.l+5*sram.lambda], [3*sram.lambda,
                    2*sram.lambda], sram.V_DD, ~val);
                sram.MN2 = MOS(4, 'n', [offset(1)*sram.w+20*sram.lambda,
                    offset(2)*sram.l+22*sram.lambda], [6*sram.lambda,
                    2*sram.lambda], sram.V_DD, val);
            end

        end

        function sram = update(sram, val)
            sram.value = val;
            sram.MP1.state = val;
            sram.MN1.state = ~val;
            sram.MP2.state = ~val;
            sram.MN2.state = val;
        end
    end
end
```

A.5 Memory class

```
classdef Memory
    %MEMORY
```

```
    properties
        ramCells;
    end

    methods
        function memory = Memory(numCells, V_DD, init_value, gs)
            %MEMORY Construct an instance of this class
            for n = 1:numCells(1)
                for m = 1:numCells(2)
                    memory.ramCells{n,m} = SRAM(V_DD, init_value, [n-1, m-1], gs);
                end
            end
        end
    end
end
```