FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Negative Voltage Fault Injection Attacks on Microcontrollers

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Hardware and Software Security

eingereicht von

Christian Kudera, BSc.
Matrikelnummer 0926721

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Dipl.-Ing. Dr.techn. Markus Kammerstetter, BSc.

Wien, 20. Februar 2018

Christian Kudera	Wolfgang Kastner

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Negative Voltage Fault Injection Attacks on Microcontrollers

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Hardware and Software Security

by

Christian Kudera, BSc.
Registration Number 0926721

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Dipl.-Ing. Dr.techn. Markus Kammerstetter, BSc.

Vienna, 20th February, 2018

Christian Kudera	Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Christian Kudera, BSc.
Altgasse 3/24, 1130 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. Februar 2018

Christian Kudera

Danksagung

Ganz besonders möchte ich Herrn Prof. Wolfgang Kastner für die Betreuung und die Möglichkeit zur Erstellung der Arbeit danken.

Ein besonderer Dank gilt meinen Freunden und Kollegen am Vienna Seclab. Allen voran Dr. Markus Kammerstetter, der mich bei der Erstellung der Arbeit maßgeblich begleitet hat und mir immer mit Rat zur Seite stand. Daniel Burian, Markus Müllner und Viktor Ullmann, die immer ein offenes Ohr für meine Probleme hatten und mich mit viel Geduld unterstützt haben.

Ein großer Dank gebührt Daniel A. Maierhofer, der mit seinem Fachwissen im Bereich der Elektrotechnik eine große Unterstützung im analogen Schaltungsdesign war.

Der größte Dank gebührt aber meinen Eltern Edith und Paul Kudera, die mich von früher Kindheit an bei allen meinen Vorhaben unterstützt und begleitet haben.

Acknowledgements

I would particularly like to thank Prof. Wolfgang Kastner for the support and the opportunity to create this master thesis.

Special thanks to my friends and colleagues at the Vienna Seclab. Above all, Dr. Markus Kammerstetter, who was instrumental in the creation of the work and was always on hand with advice. Daniel Burian, Markus Müllner and Viktor Ullmann, who always had an open ear for my problems and supported me with a lot of patience.

Many thanks go to Daniel A. Maierhofer, who was a great supporter of analogue circuit design with his expertise in the field of electrical engineering.

The biggest thanks deserve my parents Edith and Paul Kudera, who have supported and accompanied me since my early childhood.

Kurzfassung

Fault Angriffe sind eine wohl bekannte Angriffsform im Bereich der Hardware Security. Eine verbreitete Art der Fault Injection ist das kurzfristige Variieren der Versorgungsspannung, wodurch ein dafür anfälliger Prozessor Instruktionen falsch interpretiert oder überspringt. Glücklicherweise erkennen immer mehr Hersteller von Mikrocontrollern die Wichtigkeit von gehärteter Hardware und implementieren Gegenmaßnahmen gegen Fault Angriffe. In dieser Arbeit wird eine neue Methode der Fault Injection Angriffe vorgestellt. Während die Spannungsversorgung bei herkömmlichen Angriffen lediglich in Richtung GND gezogen wird, wird bei der neuen Methode auch der negative Spannungsbereich ausgenutzt. Die Hypothese dieser Arbeit ist, dass dadurch kürzere Glitches und eine schnellere kapazitive Entladung erreicht werden. Durch die Nutzung von negativen Spannungen wird eine höhere Flankensteilheit erwartet, da Schaltungsimplementierungen innerhalb und außerhalb von Mikrocontrollern schneller entladen werden können. Im Rahmen der Arbeit wurde ein Prototyp für die neue Methode implementiert und evaluiert, um diese Hypothese zu überprüfen. Die Ergebisse zeigen, dass insbesondere in Gegenwart von höheren kapazitiven Lasten Fault Angriffe nicht nur vereinfacht, sondern überhaupt erst ermöglicht werden. Gegenüber klassischen Angriffen konnten zudem kürzere Glitches erreicht werden, sodass auch Controller mit höheren Taktraten angegriffen werden können.

Abstract

Fault attacks are a well known physical attack type in the area of hardware security. A common fault injection technique is a short term variation of the supply voltage causing a vulnerable processor to misinterpret or skip instructions. Fortunately, an increasing number of microcontroller manufacturers recognize the importance of hardened hardware and implement countermeasures against fault attacks into their products. In this work, we present a new fault injection attack method. While conventional attacks pull the power supply rail to GND, in the new method we pull into the negative voltage supply range instead. The hypothesis of this work is that negative voltage fault injection attacks provide advantages over their conventional counterparts with respect to shorter glitch durations in presence of capacitive charges. Utilizing negative voltage during the generation of a fault, we expect higher slew rates due to faster discharging of the circuit implementations outside and within microcontrollers. Within this work, we implemented and evaluated a negative voltage fault injection prototype to test this hypothesis. The results show that especially in presence of higher capacitive loads, fault injection attacks are not only simplified, but they become feasible in the first place. In contrast to classical attacks, shorter glitches were achieved opening the attack vector even to controllers with higher clock rates.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Today, microcontrollers are essential to our way of life and they are used in a wide range of embedded systems. Gartner, Inc. forecasts that 20.8 billion connected things will be in worldwide use in 2020 [1]. Data in embedded devices is often highly confidential and sensitive, so privacy and security expectations must be fulfilled. Furthermore, the highly distributed nature of embedded devices allows malicious attackers physical access to the systems. As a result, they must be hardened against any kind of physical attacks just the same.

Fault attacks are well known physical attacks in the area of hardware security. One of the most common fault injection technique is a short term variation in supply voltage, which may cause a processor to misinterpret or skip instructions [2].

Unfortunately, many manufacturers of embedded devices do not consider fault attacks during system development, although the dangers of fault attacks have been known for over 15 years. The first academic fault attack [3] was published in 2001 and describes a number of methods for attacking public key algorithms. A more recent real-world attack was the Xbox360 reset glitch attack [4] in 2012. The focus of the attack was to execute unsigned code to circumvent Microsoft's security concept. In a nutshell, the processor of the console was attacked by sending a short reset pulse that changed the behavior of the `memcmp` function. In presence of the fault attack, the function `memcmp`, used to check the bootloader SHA hash digest against a stored one, returned with the result that there was no difference. The attack thus circumvented the copy protection of the game console and made playing pirated games possible. The attack was impemented in the form of so called *mod chips* to be useable for everyday consumers. Mod chips are a mass market today.

Fortunately, more and more microcontroller manufacturers recognize the importance of hardened hardware and implement countermeasures into their products. However, since microcontrollers with countermeasures against fault attacks are more expensive in comparison to their unprotected counterparts, fault attacks are still feasible on many embedded devices. On these devices, fault injection attacks can also be utilized by independent security researchers to obtain the firmware in the presence of readout protections such as security fuses or bootloader protections. Without access to the firmware, independent in-depth security audits are not feasible and thus increase *security-by-obscurity* scenarios where security critical firmware can not be scrutinized for vulnerabilities.

1.2 State of the Art

Recently, two commercial solutions for voltage fault injection attacks have been released: The VC Glitcher [5] including the Glitch Amplifier [6] by Riscure[1] and the ChipWhisperer [7] with the VC Glitch add-on by NewAE Technology Inc.[2]. While the Riscure solution allows negative voltages to some extent, both approaches primarily focus on conventional voltage fault injection attacks.

In 2000, Sergei P. Skorobogatov released a summary [8] of possible attack vectors on common microcontrollers. Even though the summary was released over a decade ago, most of the microcontrollers covered are still in use today. Voltage glitching was one of the described attack vectors, but negative voltage fault attacks are not covered.

In 2006, Bar-El et al. described different fault injection attacks on cryptographic implementations in their paper [2]. However, negative voltage fault attacks are not mentioned in their publication.

In 2014, Carpi et al. published a paper [9] which summarized a novel methodology for choosing multiple parameters required for effective faults on smart cards. Since their search space for the glitch voltage was between -5.0 V and -0.05 V, they handled negative voltage fault injection attacks, but only for low power smart cards and not for microcontrollers or larger controllers in general.

In the same year, Zussa et al. released a paper [10] where they analyzed positive and negative voltage fault attacks on Field Programmable Gate Arrays (FPGAs) with an on-chip voltmeter. Although they used negative voltage to inject the glitch, they didn't compare conventional voltage fault injection attacks against negative voltage fault injection attacks.

1.3 Problem Statement

The hypothesis of this work is that negative voltage fault injection attacks provide advantages over their conventional counterparts with respect to shorter glitch durations

[1] https://www.riscure.com
[2] https://newae.com/

in presence of capacitive and inductive charges. Utilizing negative voltage during the generation of a fault, higher slew rates are expected due to the faster discharging of the circuit implementations outside and within microcontrollers. The aim of this work is thus to design and implement a negative voltage fault injection prototype to generate and evaluate these attacks against their conventional counterparts.

1.4 Methodology

The methodological approach consists of three steps.

In the first step, a literature survey is conducted to obtain background knowledge on fault injection attacks. In existing work, different fault injection methods are identified and analyzed with regard to their usability for negative voltage fault injection attacks.

In the second step, different approaches and ideas for negative voltage glitch generation are explored and evaluated in Simulation Program with Integrated Circuit Emphasis (SPICE) simulations. Based on the results of these simulations, the requirements for electronic components are specified and a hardware prototype for negative voltage fault injection attacks is implemented. During prototyping, printed circuit boards are constructed, manufactured and assembled. Measurements conducted with the prototype are subsequently compared with the simulation results and circuit improvements are be conducted if necessary.

In the third step, the prototype and the negative fault injection approach are comprehensively evaluated on a real-world microcontroller in different configurations. During the evaluation, both conventional and negative voltage fault injection attacks are tested on the printed circuit prototype. The obtained results are then compared to determine whether negative voltage fault injection attacks provide advantages over conventional voltage fault injection attacks with respect to glitch duration, voltages and success rates.

1.5 Outline

This work is structured as follows. Chapter 2 describes the fundamentals of microcontrollers to understand how fault injection attacks work in general. Chapter 3 provides an overview of voltage fault injection attacks and introduces the two well-known non-invasive glitching techniques *clock glitching* and *voltage glitching*. Chapter 4 highlights the limitations of conventional voltage fault injection attacks. Furthermore, the hypothesis of *negative voltage fault injection attacks* is provided. Chapter 5 presents the hardware prototype implementation. In a first step, the requirements are specified. Different approaches and ideas for negative voltage glitch generation are explored and evaluated in electronic SPICE simulations. Based on the results of these simulations, the requirements for electronic components are specified and hardware prototypes for negative voltage fault injection attacks are implemented. The different prototypes are evaluated and the most promising approach is selected. Chapter 6 describes the evaluation process and utilized

test setups to validate the hypothesis of this work. Chapter 7 presents and discusses the results of the evaluation. Chapter 8 summarizes the findings and presents a conclusion of the results. Finally, we provide an outlook and ideas for further work in Chapter 9.

CHAPTER 2

Microcontrollers

This chapter describes the fundamentals of microcontrollers, which are important to understand how fault injection attacks work.

2.1 Overview

Microcontrollers are integrated circuits containing a CPU, memories and accompanying peripherals within a single chip. According to Stan Augarten [11], the TMS 1000 was the fist microcontroller, which became commercially available in 1974. It combined a processor, a clock and memory on one chip and was targeted towards embedded systems.

Today, there are countless different types of microcontrollers on the market. They are typically customized to their area of application to minimize costs and required space. For example, the Atmel ATtiny4 [12] comes in a SOT-23 package with only 6 pins where, depending on the configuration, each pin except VCC and GND can be used for multiple functions. This specific type is low-cost, requires little space and has a low power consumption. At the same time, a compromise is made between performance and usability. In contrast, there are highly integrated microcontrollers with more than 200 pins and the border between microcontrollers and Systems-on-Chips (SoCs) becomes increasingly blurred. The larger controller types can be used for a wide range of applications and it is not uncommon that a full blown operating system such as Linux is used on them. The Raspberry Pi is a well known example for that, although it utilizes external non-volatile memory and, hence, no longer falls into the microcontroller definition where internal memory is used instead.

2.2 Semiconductors

Semiconductors are a group of chemical elements with an electrical conductivity value between that of electrical conductors and that of non-conductors. By introducing foreign

elements (doping) into a semiconductor compound material, the conductivity and the conduction characteristics (electron and hole conduction) can be specifically influenced. The combination of differently doped regions (i.e., with a lack of electrons (*p-type*) or an excess of free electrons (*n-type*)), for example in the case of a p-n junction, allows electronic components with a direction-dependent conductivity (e.g., a diode) or with a switching function (e.g., a transistor) [13].

2.2.1 MOSFET

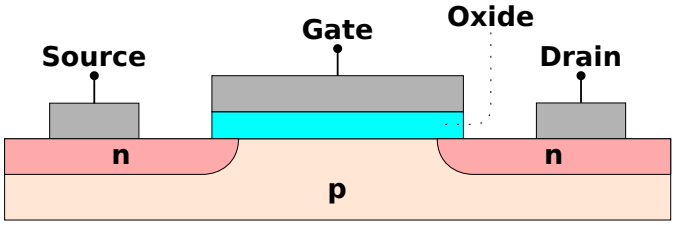

Figure 2.1: Cross Section of an n-type MOSFET

Metal–oxide–semiconductor field-effect transistors (MOSFETs) are field-effect transistors (FETs) with an insulated gate. They can be either enhancement mode MOSFETs, meaning that they are *OFF* at zero gate-source voltage, or depletion mode MOSFETs, where they are *ON* at zero gate-source voltage. Due to the relevance for complementary metal-oxide-semiconductor (CMOS) technology, enhancement mode MOSFETs will be further discussed in Section 2.2.3.

MOSFETs can be distinguished in n-type and p-type MOSFETs. Figure 2.1 shows a cross section of a n-type MOSFET. Source and drain are n regions and the body is a p region. If the voltage U_{GS} between gate and source is lower the threshold U_{TH}, drain and source are not connected. If the voltage U_{GS} between gate and source is *positive* and *higher* as the threshold U_{TH} ($U_{TH} > 0V$), a channel between drain and source forms due to electrostatic attraction of n-carriers and current can flow from drain to source.

In contrast, in p-type MOSFETs, source and drain are p regions and the body is an n region. If the voltage U_{GS} between gate and source is *negative* and *lower* as the threshold U_{TH} ($U_{TH} < 0V$), current can flow from source to drain [14, 13].

2.2.2 Floating-Gate MOSFET

A floating-gate MOSFET (FGMOS) (Figure 2.2) is a MOSFET with a electrically isolated gate (*floating-gate*) and a secondary gate (*control gate*) deposited above the floating-gate. The floating-gate is surrounded by highly resistive, dielectric material (*oxide*) so that the contained charge remains stored even if the FGMOS is not biased. The floating-gate can be charged by applying a high voltage to control gate and drain, while source is connected to the ground. In modern FGMOS, the floating-gate is positioned over the n-channel of the drain. The charge can be discharged by applying high voltage to the

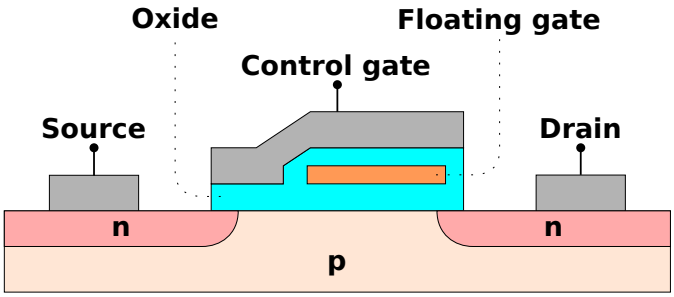

Figure 2.2: Cross Section of a Floating-Gate MOSFET

drain, while the control gate and source are connected to ground. In older FGMOS types, the electrical discharge was often not feasible. If there is no charge stored in the floating-gate, the behavior is similar to a normal MOSFET. However, if there is charge stored in the floating-gate, the transistor is effectively blocked since the charge in the floating-gate prevents the formation of a channel between drain and source [15]. Due to this property, the FGMOS can be used to store a bit. FGMOSs are thus utilized to form non-volatile memories which are described in more detail in Section 2.3.2.

2.2.3 CMOS

Complementary metal-oxide-semiconductor (CMOS) is a term for semiconductor devices where both n-type MOSFETs and p-type MOSFETs are used on a common substrate. It has a high noise immunity, a low static power consumption and the operating voltage can be between 0.75 V and 15.0 V. CMOS is the most widely used technology in microprocessors, microcontrollers, memory devices and other digital logic circuits [14].

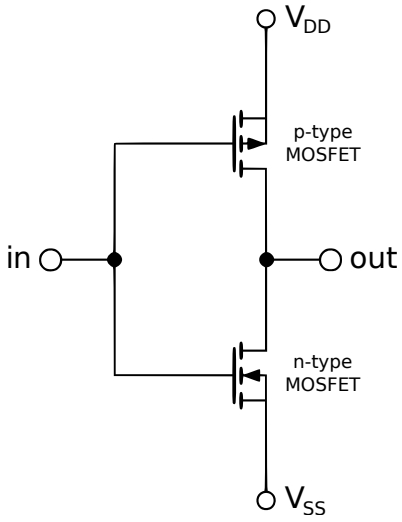

Figure 2.3: CMOS Inverter

Figure 2.3 shows a CMOS inverter, which outputs a voltage representing the opposite logic-level to its input. If the p-type MOSFET is active and the n-type MOSFET is inactive, the output is connected to V_{DD}. Vice versa, if the p-type MOSFET is inactive and the n-type MOSFET is active, the output is connected to V_{SS} [14].

Table 2.1 provides an overview of typical logic levels at 5 V operating voltage. It should be noted that although the acceptable input logic level voltage range is relatively wide, the output level always resides in a small voltage range.

	Input	Output
LOW	0.0 V – 1.5 V	0.0 V – 0.05 V
HIGH	3.5 V – 5.0 V	4.95 V – 5.0 V

Table 2.1: CMOS Logic Levels for 5 V Operating Voltage [14]

Table 2.2 provides an overview on different CMOS technologies and their characteristics. Among other factors, the speed of microcontrollers is primarily limited by the gate delay.

Technology	Abbr.	Power loss per gate	Processing time per gate
Standard CMOS	C	0.3 µW	90 ns
High speed CMOS	HC	0.5 µW	10 ns
Advanced CMOS	AC	0.8 µW	3 ns

Table 2.2: CMOS Characteristics [14]

2.3 Components

This section explains the central components within a microcontroller. Since the implementation details of microcontrollers differ significantly between manufacturers, only a rough overview is provided.

2.3.1 Processor Core

The processor core executes a program by processing instructions (e.g., logic, arithmetic, control and input/output instructions). The components of the core are explained on the basis of the Micro16 educational reference architecture introduced by Schildt et al. [16].

Registers

A register is a circuit comprising several flip-flop gates and stores a certain number of bits. Typically, the size of a standard register is equally large as the bus width within the core. In general, there are different types of registers depending on their application. *Data registers* hold data values such as integers and are used to store operands or the results of calculations. *Address registers* are used to address memory areas. *Status registers* (also known as *status flags*) are used to store truth values such as whether the result of a

previous calculation was zero (zero flag). In addition, there are *special purpose registers* which are explained in the following.

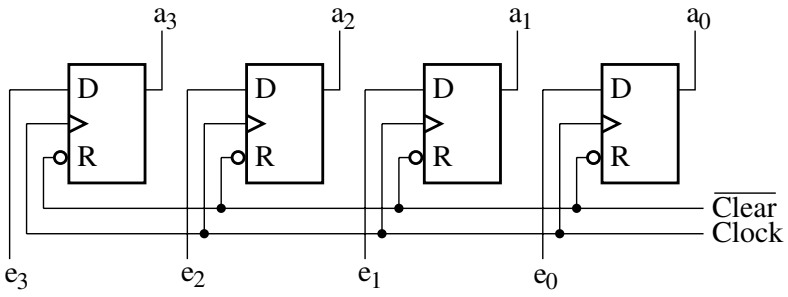

Figure 2.4: A 4-bit Register [16]

Figure 2.4 illustrates a 4-bit register built from four flip-flops. A flip-flop is a basic logic circuit storing one bit of data and it is typically manufactured in CMOS technology. The illustrated flip-flops have a data (D), a clock (>) and a reset (R) input and a one output. The register comprises 4 flip-flops and can thus store a 4-bit word. By holding R low for one clock cycle, the register is erased. When e_3 and e_1 are high and e_2 and e_0 are held low for one clock cycle, the word (1010) is stored. The current value of the register can be accessed through the outputs of the register.

Arithmetic Logic Unit (ALU)

An ALU performs the arithmetic and logic operations. The Micro16 ALU visible in Figure 2.5 supports the following functions:

- Map a 16-bit data word received from register A to an output register without changing the value.
- Addition of two 16-bit data words received from registers A and B. The results are written to the output data and flag registers.
- Bitwise AND operation of two 16-bit data words received from registers A and B. The results are written to the output data and flag registers.
- Bitwise NOT operation of one 16-bit data word received from register A. The results are written to the output data and flag registers.

Additionally, the result of the four functions can be bitwise shifted to the left or to the right. The Micro16 ALU has a 2-bit control input (F_0F_1) to select which function should be executed and a 2-bit control input (S_0S_1) to select whether the result should be shifted (either left or right) or not. Furthermore, there are the *status flags* ZERO (Z) and NEGATIVE (N) which are set if the result of the operation is either zero or negative.

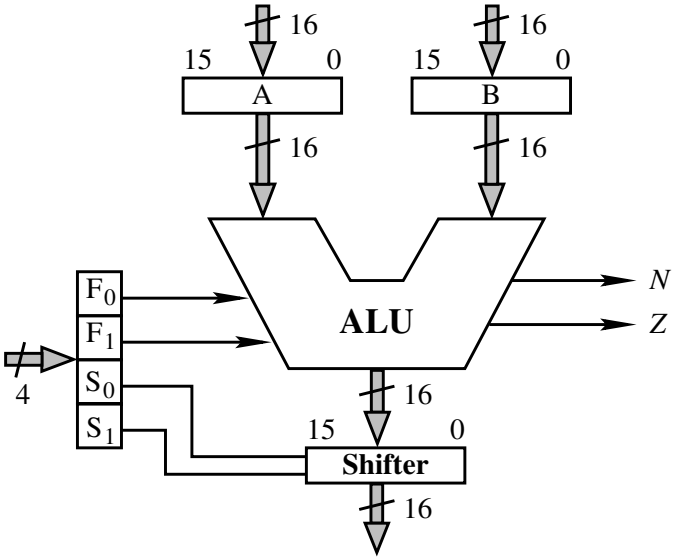

Figure 2.5: The Micro16 ALU [16]

micro instruction ($F_0F_1S_0S_1$)	symbolic	micro instruction ($F_0F_1S_0S_1$)	symbolic
(0000)	A	(1000)	A ∧ B
(0001)	lsh(A)	(1001)	lsh(A ∧ B)
(0010)	rsh(A)	(1010)	rsh(A ∧ B)
(0100)	A+B	(1100)	¬A
(0101)	lsh(A+B)	(1101)	lsh(¬A)
(0110)	rsh(A+B)	(1110)	rsh(¬A)

Table 2.3: Micro Instructions of the Micro16 ALU [16]

Table 2.3 shows the possible *micro instructions* and corresponding functions of the Micro16 architecture.

Overall, the registers A and B are connected to thirteen *data registers* ($R_0 \ldots R_{12}$) and three read-only *data registers*, containing the constants 0, - 1 , + 1. The registers are connected via buses that are described in the following section.

Bus

A bus is an electrical connection between several components of a system and is used to share information between them. There are three types of buses: The *data bus*, the *address bus* and the *control bus*. The transmission of the bits can either be in parallel (e.g., simultaneously) or in serial (e.g., one bit after another). In the Micro16 architecture, the transmission is implemented in parallel so that a distinct bus wire is required for

each bit.

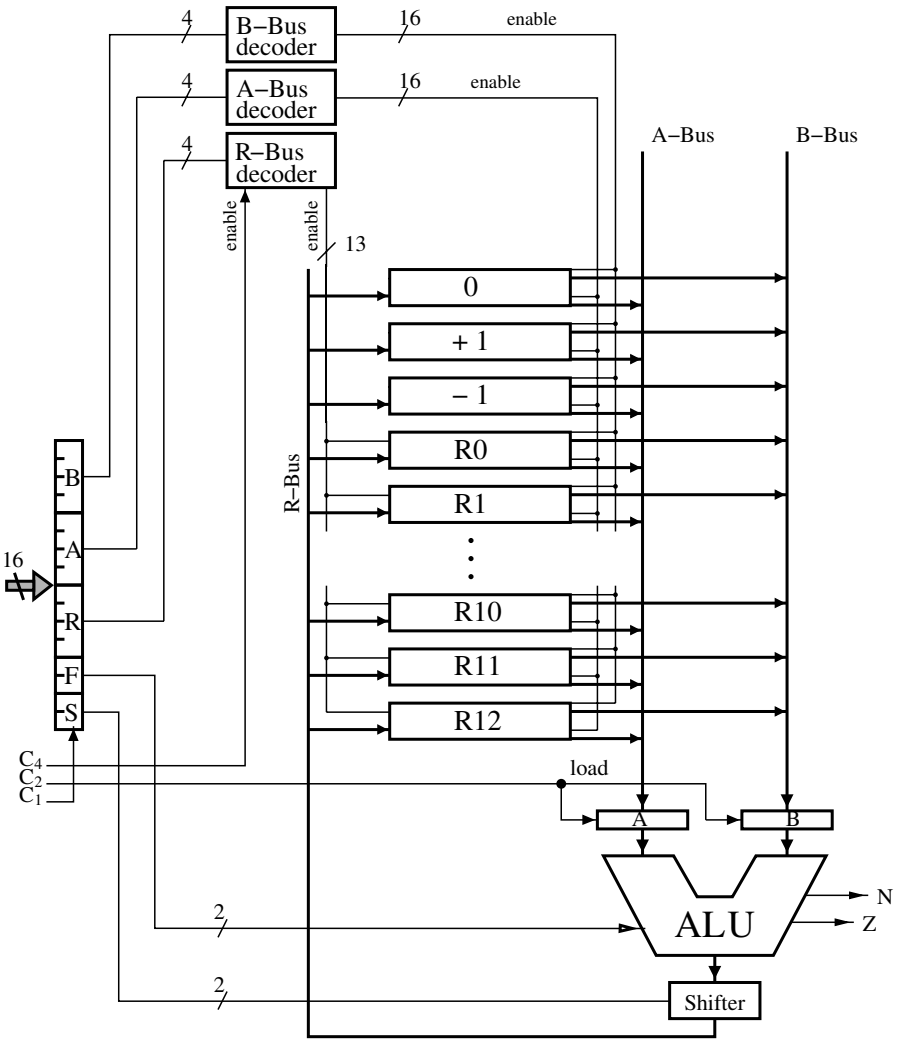

Figure 2.6: Micro16 ALU with Registers and Buses [16]

Figure 2.6 illustrates the bus connections within the Micro16 architecture. The thirteen registers ($R_0 \ldots R_{12}$) can write a value to buses A and B and they can read a value from bus R into the corresponding register. The three constant registers (0, - 1, + 1) can only write their static value to the buses A and B. Each register thus has either two or three enable inputs. To minimize the length of the micro instructions, the selection of the register is encoded in 4-bits and must be decoded in a bus decoder. For instance, to enable writing from register R_0 to bus A, the value $(0100)_2$ is transmitted to the A-bus decoder. Furthermore, the ALU registers A and B each have an input enabling reading from the buses A or B. The figure shows the timing signals C_1, C_2, C_4 as well. These signals are used to coordinate the proper behavior of the processor core and they are be

explained in more detail in the following sections.

Memory Connection

To access the memory, the Micro16 architecture has two registers: The Memory Address Register (MAR) and the Memory Buffer Register (MBR) (Figure 2.7). The MAR addresses the location of the memory and is connected to an address bus. The MBR buffers data before it is written to the memory or after it is read from the memory. The MBR is connected to the data bus and can be switched between read and write mode via a read/write signal. The memory select signal triggers the memory to read or write a value from or to the MBR. Since the MAR is 16-bit long, addresses from 0 to $2^{16} - 1$ can be addressed.

The reading process is divided into the following steps:

1. The address of the memory cell is written to MAR and the signals read/write and memory select are asserted.
2. The memory access time has to pass.
3. The MBR can read the value from the data bus into the register.

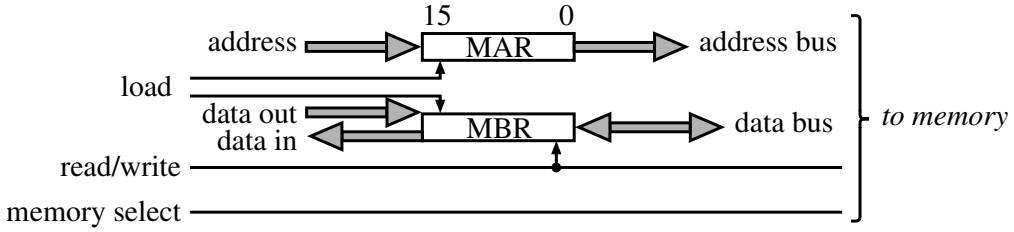

Figure 2.7: Micro16 Memory Connection [16]

Similarly, the writing process is divided into the following steps:

1. The address of the memory cell is written to MAR and the signal memory select is activated.
2. The memory reads the value from the MBR and writes it to the address defined in the MAR.

Figure 2.8 illustrates how the buses are connected to the MBR and MAR. With the signal A_0, a multiplexer switches the A input of the ALU between bus A and the MBR. Since the micro instruction function (0000) maps a 16-bit data word from the ALU input to the output, the value of the MBR can be saved in each data register. The MAR is connected to bus B, so that the memory address can be stored in the data registers. The output of the ALU is written to the MBR, which is required at the writing process.

Program Memory and Micro Sequence Logic

Figure 2.10 shows the Micro16 architecture including new components that are described in the following. The program is stored in a 256x32 sized *program memory*. Since all micro instructions of the Micro16 architecture are 32-bits long, a program can have up to 256 instructions at most. The address of the current instruction is stored in the *Micro Instruction Counter (MIC)* also known as *program counter* in other architectures. The *Micro Instruction Register (MIR)* contains the current micro instruction.

The *micro sequence logic*, which enables conditional and unconditional jumps to an arbitrary position in the program, is connected to the MIC and to the status flags of the ALU. The micro instruction is extended by eight bit (*ADR* for address) representing

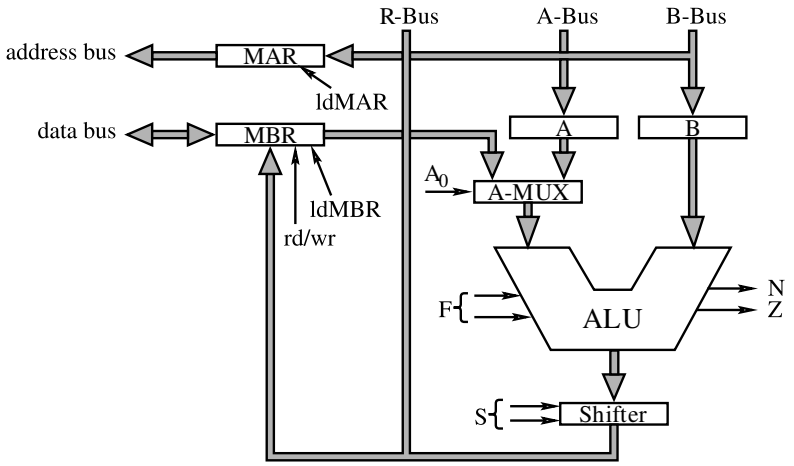

Figure 2.8: Micro16 ALU, Buses and Memory Connection [16]

the destination of the jump. In addition, the micro instruction is extended by two bits (*COND* for condition) to select the jump type. The type can be one of the following:

- $(00)_2$: Do nothing
- $(01)_2$: Jump, if the status flag N is one
- $(10)_2$: Jump, if the staugs flag Z is one
- $(11)_2$: Jump without any condition

Control Unit

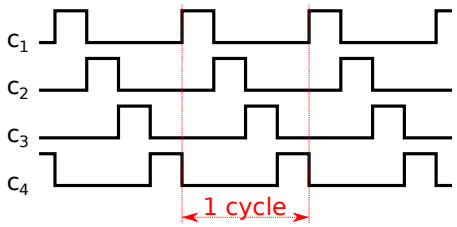

Figure 2.9: Micro16 Clock Timing Diagram

The control unit is a integrated logic block contained within the processor core. It coordinates the execution of a program by interpreting the instructions to provide timing and control signals for the other processor core units such as the ALU and the memory block. One important part is the clock component (*4 phase clock*). Based on a reference clock signal provided by an external crystal or an internal oscillator, it generates four successive clock signals (i.e., *clock cycles*) (Figure 2.9). The following describes functions of each clock cycle:

- C_1: The micro instruction in the MIR is executed. The *A-Mux* multiplexer either selects the MBR or the A register as input for the ALU. The behavior of the *micro sequence logic* is selected and the ALU function is chosen. The MBR and MAR are either enabled or not. If the memory select bit is set, the memory is enabled and according to the RD/WR signal the function is selected. Values are written to the buses A and B. If the ENR bit is set the register, which will store the value from the R bus in C_4, is selected.
- C_2: The registers A and B read the current value from the buses A and B.
- C_3: The MAR and MBR are enabled.
- C_4: The ALU result is stored in the selected register. The next micro instruction is selected by the MIC.

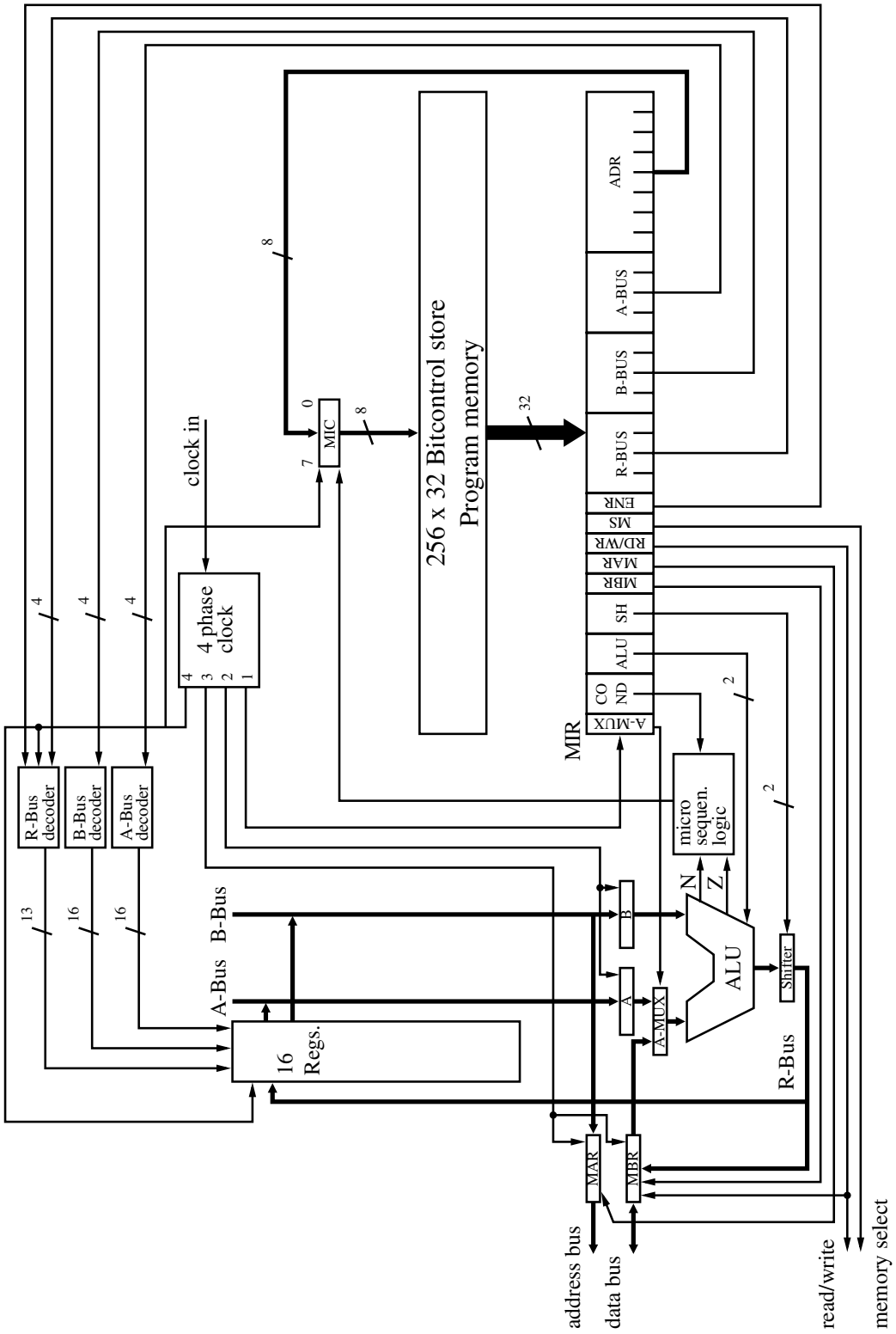

Figure 2.10: Complete Micro16 Architecture [16]

2.3.2 Memory

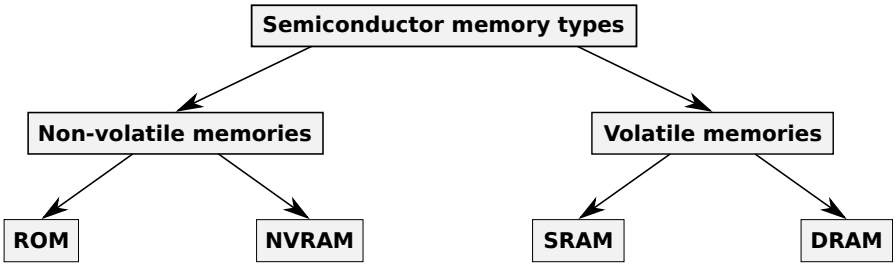

Figure 2.11: Different Types of Semiconductor Memory

Memory is used to store data and is implemented in microcontrollers as integrated circuit logic block. Figure 2.11 provides an overview of different memory types. In general, memory can be divided into volatile and non-volatile types [17]. Volatile memory loses the stored data if the power is turned off. In contrast, stored data can be retrieved from non-volatile memory even if the supply voltage has been turned off and back on again. Non-volatile memory can be further divided into read-only memory (ROM) and non-volatile random-access memory (NVRAM) while volatile memory can be divided into static random-access memory (SRAM) and dynamic random-access memory (DRAM).

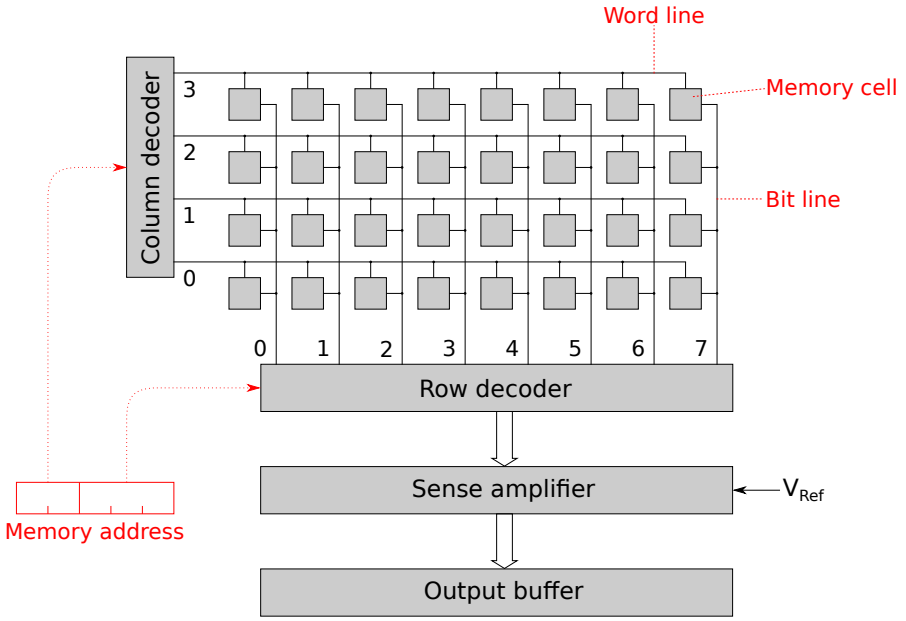

Figure 2.12: Typical Memory Layout

Although there are different types of memory, their fundamental on-chip design (Figure 2.12) is similar. A single bit is stored in a *memory cell*. Memory cells are laid out in an array on the surface of the chip, where each row represents a data word. Horizontally there are the word lines and vertically there are the bit lines. Usually, the two basic

operations performed by a memory are *read*, where a data word is read from the memory, and *write*, where a data word is written to a memory. The data word is accessed by the *memory address*, which is divided into a column and a row part. The memory address is decoded by a *column decoder* and a *row decoder* to select the proper data word. Since due to physical memory technology properties, the memory is typically not operating on the same logic levels, a sense amplifier is required [18]. The sense amplifier shifts the charge of each bit to a range which is comprehensible for the rest of the microcontroller. To decide whether a sensed charge level represents a 0 or a 1 bit, a voltage reference V_{Ref} is required for comparison purposes. After the sense amplifier, the addressed data word is finally stored in an output buffer. The following sections provide an overview of the different types of memory.

Static random-access memory (SRAM)

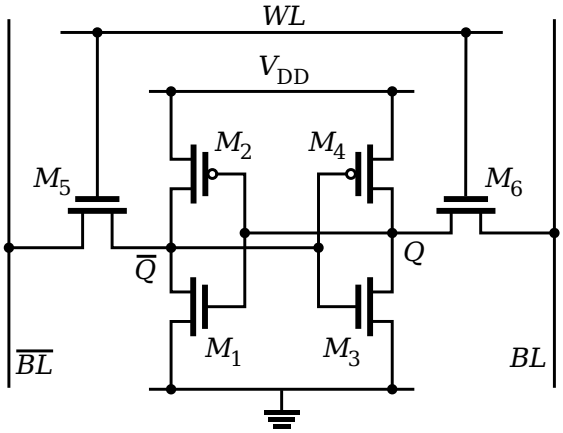

Figure 2.13: Circuit Diagram of an SRAM Cell, Built With Six MOSFETs [19]

An SRAM is a volatile memory that uses a bistable latching circuit to store a bit in a memory cell. The term static means that the SRAM does not need to be refreshed periodically to hold its content.

A typical SRAM memory cell, illustrated in Figure 2.13, is made up of six MOSFETs, known as *6T SRAM cell*. The MOSFETs M_1, M_2, M_3 and M_4 form two cross-coupled inverters. Therefore, the memory cell has two stable states to store a bit. The MOSFETs M_5 and M_6 are used to control the access during read and write operations.

The advantages of SRAMs are low power consumption, no need of a refresh circuit and fast access time. However, the disadvantage is the required space due to the high amount of transistors. For that reason, SRAM is mainly used for areas where fast access time are important (e.g., caches).

Dynamic random-access memory (DRAM)

A DRAM is a volatile memory storing each bit of data in a capacitor within an memory cell. However, the capacitor looses its charge over time so that the memory cell must be refreshed periodically. The advantages of the DRAM are fast access time, structural simplicity, low space requirements and low manufacturing costs. On the other hand, due to its dynamic nature, DRAM consumes large amounts of power and it requires logic for refreshing.

Read-only memory (ROM)

A ROM is a non-volatile memory type. Depending on whether the ROM is reprogrammable, once programmed memory content can be re-programmed at a later point of time. In microcontrollers, a ROM is mainly used to store the firmware and bootloader code. In the following, the different types are described, whereby the chronological order reflects the time of development.

Mask ROM (MROM): In MROMs, the content of the memory is hardwired in the circuit layout of the integrated circuit. For a memory cell which should contain a logical 1, the word line is connected to the bit line, otherwise they are not connected and the memory cell contains a logical 0. The connection can be realized with a diode or a transistor. The advantage of MROMs is that they need less space compared to other ROM technologies and they are feasible at low cost for mass production. However, the content can not be modified and the ROM type is only cost efficient at high quantities.

Programmable ROM (PROM): A PROM can be programmed by a user with a PROM programmer only once. Each memory cell contains a connection between the word line and the bit line. The connection is achieved with a diode and a fuse or a floating-gate MOSFET, where the charge of the floating-gate can not be discharged. The programmer addresses the memory cells which should contain a logic 0. By applying high voltage to the addressed memory cells, the fuses are burned or the floating-gates of the floating-gate MOSFETs are charged as described in Section 2.2.2. As a result, for the addressed memory cells, the connections between word lines and bit lines are destroyed. The advantage of PROMs is that they can be programmed after chip fabrication. The disadvantage is that the PROM needs more space compared to the MROM and, hence, it is more expensive to manufacture.

Erasable programmable ROM (EPROM): The EPROM can be programmed and erased by the user a limited number of times. The structure of the EPROM is similar to the PROM. However, the connection in the memory cells is solely achieved with floating-gate MOSFETs where the charge of the floating-gates can not be discharged with high voltage. The programming works in the same manner as for PROMs. To erase the content of the memory cells, the memory gets exposed to strong ultraviolet light that discharges the floating-gates. The package of the memory thus contains a quartz window that is transparent to ultraviolet light. The advantage of EPROMs is that they can

be programmed and erased, but the the number of reprogramming is limited and the package of the memory needs a quartz window, which increases the manufacture costs.

Electrical erasable programmable ROM (EEPROM): Today, EEPROMs, which can be programmed and erased by the user up to 10^4 – 10^6 times, have a great significance in the semiconductor technology [20]. The structure of the EEPROM is similar to the EPROM, but the floating-gate MOSFET can be erased with high voltage, as described in Section 2.2.2. A special variant of EEPROM is *flash memory*. Flash memory cells differ from EEPROM memory cells in the thickness of the surrounding floating-gate oxide. Since the oxide is thinner, a lower voltage is required for programming and erasing the memory cells. Therefore, the voltage converter and programming logic can be contained in microcontrollers. Considering write operations, typical EEPROM implementations allow individual addressing of words whereas Flash memories only allow addressing in blocks to achieve higher memory densities.

Non-volatile random-access memory (NVRAM)

NVRAMs combine the advantages of SRAMs and ROMs. They contain an SRAM to achieve fast access times and they contain an EEPROM, where the data is stored if the power is turned off. The disadvantages are high manufacturing costs and large space requirements.

2.3.3 Other Features

Reset

In a microcontroller, the reset sets all peripherals including the CPU to a predefined initial state. Once the microcontroller's CPU is ready to execute instructions, execution starts at a hard-coded start address in the memory. This location is commonly known as *reset vector*.

The reset can be triggered by the following scenarios:

- External reset through an electrical signal at the reset pin
- Power-on reset (PoR), which generates a reset signal if power is applied to a microcontroller to ensure that the microcontroller starts operating in a known state
- Internal reset if a error is detected by the logic of the microcontroller
- Internal reset triggered by a program

Bootloader

A bootloader handles the booting process after the reset procedure has finished. It extends the reset procedure and simplifies tasks such as firmware updates, loading a program from an external source into the DRAM or SRAM or encrypting and decrypting

the user program. In general, there are two types of bootloaders. The first type is implemented by the manufacturer ("Mask-ROM bootloader") and is not available on all microcontrollers. It can not be modified by the user and gets directly executed after the reset procedure. The second type is a user bootloader that is implemented by the user and stored in the program memory. It gets executed either directly after the reset or after the execution of the manufacturer bootloader.

2.4 Architectures

The architecture of a microcontroller can be distinguished by the complexity of their instruction set or by the linkage of the instruction memory and data memory to the processor core. This following section provides an overview of these concepts.

2.4.1 Complexity of Instruction Set

Reduced Instruction Set Computer (RISC)

In comparison to the Complex Instruction Set Computer (CISC) architecture, the RISC architecture has fewer and only rudimentary instructions that typically only need a single clock cycle to execute. The advantage of the RISC architecture is that the execution of instructions is very fast, but at the same time the disadvantage is that the instruction set is rather simple and hence the amount of necessary instructions to solve a complex task becomes larger. The software for a microcontroller is typically written in high-level programming languages and compiled through a compiler. As a result, the high level programming language abstracts the simplified instruction set from the developer. Mainly due to faster execution times per instruction and less ALU implementation complexity, the RISC architecture is widely used in microcontroller architectures [14].

RISC systems have a large on-chip register file. Due to the high number of registers, operands intermediate results and data can be kept in the registers. As a result, the traffic between the memory and the processor is reduced, which increases the speed of operation. However, a large register file requires a more complicated decoding logic, which increases the access time to any register.

Due to the small number of instructions, the RISC architecture is normally a load/store architecture. A load/store architecture is a computer architecture whose instruction set allows data memory accesses only with special *load* and *store* instructions.

Complex Instruction Set Computer (CISC)

In comparison to RISC, the CISC architecture has a more complex instruction set. To avoid ALU implementation complexities, many implementations internally make use of microcode where a CISC instruction is represented by a number of less complex RISC instructions. Due to the internal microcode architecture, it is common for an instruction to take multiple clock cycles to execute. Hence, in comparison to RISC, the CISC

architecture has a more powerful instruction set resulting in less code size, but at the cost of more execution cycles per instruction [14].

Since the instructions at the CISC architecture can directly operate on memory, a small number of general purpose registers is required. Therefore, the register file is typically small at this architecture. However, the architecture is characterized by several special purpose registers for the stack pointer, interrupt handling, and so on.

2.4.2 Linkage of the Processor and Data Memory

Von Neumann Architecture

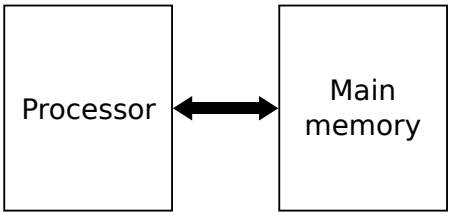

Figure 2.14: Von Neumann Architecture

In the Von Neumann architecture, instructions and data are stored in the same memory (Figure 2.14) and, as a result, only one bus between the processor and the memory is required. While the advantage is that less hardware is needed, the disadvantage is that data and instructions can not be accessed at the same time and processing delays ("hazards") might be introduced [14].

Harvard Architecture

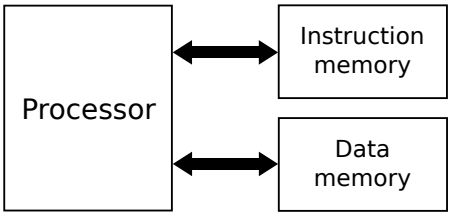

Figure 2.15: Harvard Architecture

In the Harvard architecture, the instructions and data are stored in separate memories (Figure 2.15). Although two buses are required, the architecture is used more frequently due to its advantage that there are no additional delays when accessing instructions or data in memory at the same time [14].

2.5 Software Development

This section explains the fundamentals of software development for microcontrollers. The first part describes how a program is developed, followed by a description of how the program is downloaded into the microcontroller. Different programming and debugging standards are mentioned as well.

2.5.1 Programming

As described in Section 2.3.1, *micro instructions* are defined sequences of bits that control the data flow and instruction execution of the processor core. A *machine instruction* is either directly mapped to a micro instruction or to a series of micro instructions. For instance, the Micro16 architecture contains a left shift micro instruction. Therefore, a left shift machine instruction can directly use this micro instruction. However, the architecture does not contain a multiply micro instruction so that it can be only implemented as a series of micro instructions. An *instruction set*, with its *instruction set architecture (ISA)*, is the interface between a microcontroller's software and its hardware. It defines the valid machine instructions the microcontroller can execute. The *assembly language* is a low-level programming language with a one-to-one mapping between the language and the instruction set. To simplify the programming, the assembly language uses *mnemonic codes* to refer to the machine instructions rather than using the instructions numeric values (*opcodes*) directly. The assembly program has to be translated to a *binary* form, also called *object file*, first. The result contains the machine code (*text segment*) and data such as global variables that typically reside in the (*data segment*). The translation itself from the mnemonic representation to the binary opcodes is done by the *assembler*. Normally, a program consists not only of one object file but of several object files and additional libraries. For this reason, the so-called linker merges the individual object files and libraries into a program. One of the disadvantages of the assembly language is that it has no abstraction from the instruction set. Hence, a developer has to know the assembly language for the specific architecture he is working on. For typical developers it is thus more convenient to a high-level programming language with a strong abstraction from the actual instruction set architecture instead. To translate a program written in a high-level programming language to a low-level assembler program, a *compiler* is used. The compiler needs to know the target architecture to generate the assembly code. Subsequently, the assembler can transform the generated assembly code into its binary opcode form.

2.5.2 Download

After a program has been assembled, the resulting binary must be downloaded into the microcontroller's memory. On the host computer side, this is usually done via the serial, parallel or usb interface. On the microcontroller side both standard and manufacturer proprietary programming interfaces exist. However, even though there are some standards such as JTAG [21] or SWD [22], the programming interfaces are

often proprietary solutions developed by the microcontroller manufacturers. Another possibility is the usage of a bootloader, which was described in Section 2.3.3.

Programming Interfaces

As described above, there are different programming interface types. As an example, the Atmel ATmega in-system programming (ISP) standard is described in the following.

In-system means, that the microcontroller can be programmed while it is already mounted to a printed circuit board. The programming wiring must fulfill the requirements defined in the respective datasheet [23]. The six microcontroller pins *MISO*, *SCK*, *RESET*, *MOSI*, *+5V*, *GND* are connected to a pin header. A *programming adapter* is connected to the pin header and on the other side, it is typically connected to the host computer via a USB interface. The adapter is controlled from the computer with a *programming software*. To enable the programming mode, the programmer pulls the RESET pin to low and transmits a "programming enable" command over the MOSI pin. If the programming mode is entered, the microcontroller responds with an acknowledge message over the MISO pin. Once the programming mode has been entered, further instructions like "write to program memory", "erase program memory" and "read program memory" are available. After programming is finished, the reset pin is released.

2.5.3 Debugging

A *debugger* can be used to search for errors in the developed program. State-of-the-art debuggers offers at least the following features:

- Breakpoints: Defined points in the code where the program execution should be stopped. After the stop, the values of the memory, variables and registers can be inspected.
- Single Stepping: One instruction is executed at a time. The execution is paused afterwards.
- Programming: Since the debugger has access to the program memory, it can be used to program the microcontroller as well.

To debug a microcontroller, the debugger program is executed on the host computer. There are different debugging interface standards. The most common ones are Joint Test Action Group (JTAG) and Serial Wire Debug (SWD).

2.6 Protection Mechanisms

Microcontrollers provide different protection mechanism to protect the program code from unauthorized readouts. Although the implementation of the mechanisms depend on the

manufacturer, the common key concepts are either the use of locking bits or bootloader passwords and/or keys. The basic idea of a locking bit (also known as security fuse bit) is to disable reading the memory through the programming interface. If the protection bit is disabled, the memory is erased as well. With bootloader protections, the functionality to read the memory via bootloader functions is protected either with a cryptographic key, a signature or a password. If the user does not have the necessary credentials, the respective bootloader functions are not available. The big difference between those protection mechanism types is that security fuse bits are commonly implemented in hardware while the bootloader protection is implemented in software (i.e., the bootloader comprises of instructions that are executed by the same CPU that also executes the user's program).

CHAPTER 3

Fault Injection Attacks

A fault injection attack is a procedure to intentionally introduce an error in a system to alter its execution to the attackers advantage. Fault injection effects have been known for a long time.

In 1978, May et al. published a paper [24] which describes accidental faults caused by radioactive particles produced by elements naturally present in IC packaging material. These particles caused bit flips in sensitive chip areas and hence led to undesired errors.

According to Bar-El et al. [2], faults can be divided into *provisional faults*, where the system recovers itself after a reset or when the fault's stimulus ceases, and *destructive faults*, where the system is permanently affected or even destroyed. For fault injection attacks, only the first type is of interest.

A special form of fault attacks are glitch attacks, where a pulse outside the normal operating specification is injected into the system. Glitches can be divided into *instruction glitches*, *data glitches* and *clock-signal glitches* [2, 25].

The aim of instruction glitches is to replace a single critical machine instruction like conditional jumps or the test instruction preceding them to another one. They can be used to extend the runtime of loops, for example in serial output routines to see more of the memory after the output buffer [26]. Anderson et al. described an attack on a cryptographic cipher function, where the rounds of the cipher were reduced to a single-round [27]. The aim of data glitching is to modify the data values that are used by the processor during program execution. Clock-signal glitches temporarily increase the clock frequency for one or more half cycles. As a consequence, some flipflops in the register-transfer-logic (RTL) sample their input before the new state has actually reached them. Logic blocks with low complexity may thus operate normally while more complex blocks do not finish in time and either output their previous or an intermediate state.

In general, there are two well-known non-invasive glitching techniques for creating reliable faults: *clock glitching* and *voltage glitching*. The following sections give an overview over the two techniques.

3.1 Clock Glitching

Since most common microcontrollers are based on synchronous logic, they require a system clock (see Section 2.3.1). To perform a clock glitch, the clock frequency is temporarily increased for one or more half cycles as illustrated in Figure 3.1.

In 2011, Balasch et al. published a paper [28] in which they analyzed how clock glitches affect commercial low-cost microcontrollers. They divided their results into effects of clock glitching on the program flow and effects of clock glitching on the data flow. According to the authors, the fetching of the next opcode can be affected such that it is replaced by another instruction. Skipping an instruction, which is sometimes mentioned in analyses of fault models, was not possible. The best and most stable results for effects on the data flow were obtained for multi-cycle instructions with memory access. Depending on the glitch period, the authors were able to prevent a given number of bits on the data bus from flipping.

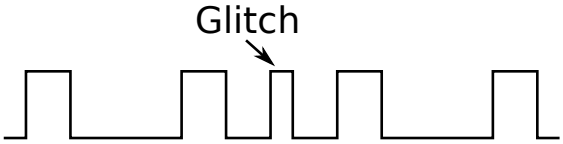

Figure 3.1: Glitch in the Clock Signal

However, non-invasive clock glitching attacks can only be applied to microcontrollers that use an external clock signal. In contrast, many of today's microcontrollers rely on an internal RC oscillator instead so that non-invasive clock glitching is often not feasible.

3.2 Voltage Glitching

Variations in supply voltage during execution may causes a processor to misinterpret or skip instructions [2]. This attack is known as *voltage glitching*.

The use of voltage glitching has been extensively reported over the last decade [29, 30, 31]. However, only a few papers have investigated the underlying fault injection mechanisms. In 2006, Djellid-Ouar et al. published a paper [32] which summarized the effects of voltage glitches on CMOS circuits. They showed that voltage fault injection attacks cannot induce faults into flip-flops. Furthermore, they showed that faults occur because of timing constraint violations, which are caused by an increase of the combinatorial logic propagation delays. However, the results described by Djellid-Ouar et al. were only achieved by simulations. Consecutively, Zussa et al. contributed an experimental proof

[33] that voltage fault injection attacks lead to timing constraint violations in a similar way as clock glitches do.

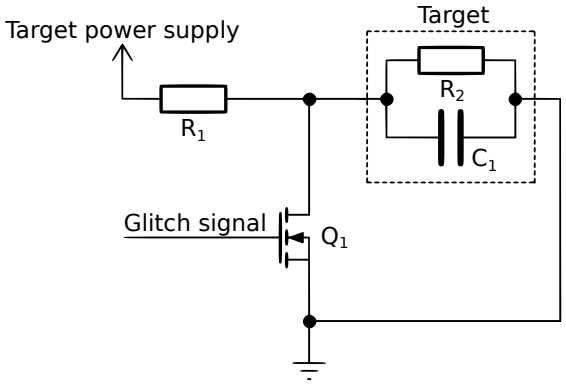

Figure 3.2: Method to Create a Glitch in the Power Supply Line of a Target

One of the easiest methods to create a glitch in the power supply line of a target is illustrated in Figure 3.2. NewAE Technology Inc.[1] uses this method [7, 34] in their commercial solution. If the logical glitch signal is high, the MOSFET Q_1 shorts the power supply line of the target to GND. The capacitor C_1 and resistor R_2 build an equivalent circuit diagram for a microcontroller. The resistor R_1 is required to prevent a short circuit between the power supply and GND.

[1] https://www.newae.com/

CHAPTER 4

Negative Voltage Fault Injection Attacks

While Section 3.2 provided an overview of voltage fault injection attacks that are hereafter referred to as *conventional voltage faults attacks* or *conventional voltage glitching*, this chapter highlights the limitations of conventional voltage fault injection attacks. Furthermore, the hypothesis of *negative voltage fault injection attacks* is described.

4.1 Basic Terms of Voltage Fault Injection Attacks

In this section, we first define a few basic terms to better understand and describe voltage fault injection attacks. Figure 4.1 provides an overview of a typical voltage glitch signal. The dotted waveform shows the logical glitch signal (i.e., the trigger for the glitch). If the logic level is high, a glitch should be inserted in the power supply line of a target. The second waveform illustrates the power supply line of a target with an inserted glitch.

The voltage levels and time periods are explained in the following.

- Glitch Signal Voltage High ($V_{GS_{High}}$): High logic level of the glitch signal.
- Glitch Signal Voltage Low ($V_{GS_{Low}}$): Low logic level of the glitch signal.
- Glitch Signal Width (GS_{Width}): The time period of the rectangular pulsed glitch signal from the moment it rises from $V_{GS_{Low}}$ until reaching $V_{GS_{Low}}$ again.
- Glitch Signal Rise Time (GS_{Rise}): The time period needed to rise from $V_{GS_{Low}}$ to $V_{GS_{High}}$.
- Glitch Signal Fall Time (GS_{Fall}): The time period needed to fall from $V_{GS_{High}}$ to $V_{GS_{Low}}$.

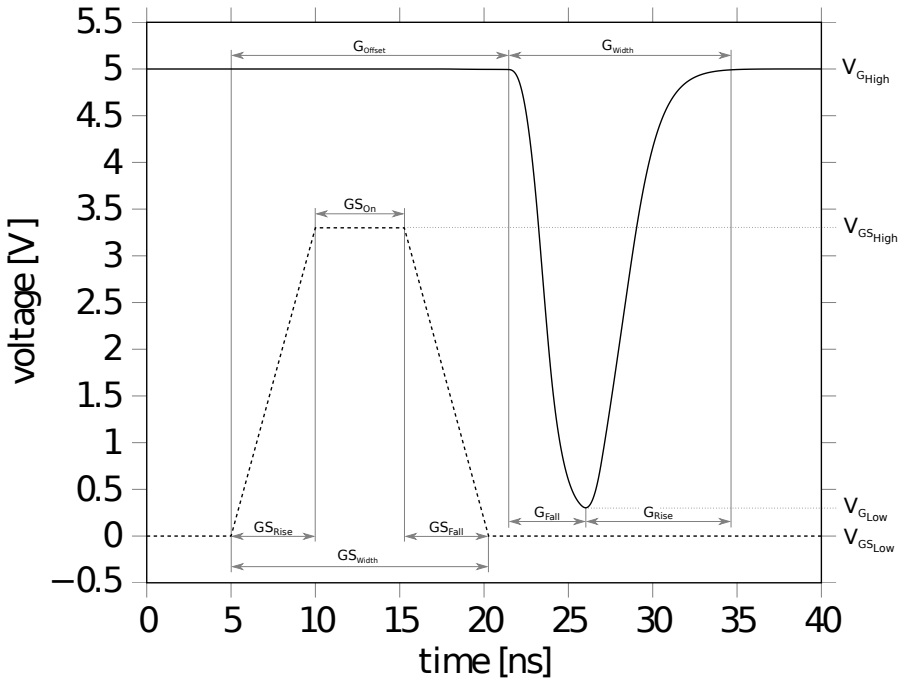

Figure 4.1: Glitch Signal (Dotted Waveform) and Power Supply Line of a Target with Inserted Glitch

- Glitch Signal On Time (GS_{On}): The time period how long the rectangular pulsed glitch signal is at $V_{GS_{High}}$.

- Glitch Voltage High ($V_{G_{High}}$): Voltage level when no glitch is inserted. Normally, this voltage is the required power supply line voltage of a target according to its datasheet.

- Glitch Voltage Low ($V_{G_{Low}}$): The lowest voltage level of an inserted glitch.

- Glitch Offset (G_{Offset}): The time period between the moment the glitch signal rises from $V_{GS_{Low}}$ to the moment the voltage level of the power supply line voltage falls from $V_{G_{High}}$.

- Glitch Width (G_{Width}): The time period of the inserted glitch from the moment the voltage falls from $V_{G_{High}}$ until the moment it reaches $V_{G_{High}}$ again.

- Glitch Fall Time (G_{Fall}): The time period needed to fall from $V_{G_{High}}$ to $V_{G_{Low}}$.

- Glitch Rise Time (G_{Rise}): The time period needed to rise from $V_{G_{Low}}$ to $V_{G_{High}}$.

4.2 Limitations of Conventional Voltage Fault Injection Attacks

The limitations of the conventional voltage fault injection attacks are explained in this section. Several scientific publications have shown that the success of a glitch depends on the glitch duration G_{Width} and the glitch depth $V_{G_{Low}}$ [35, 36]. It may even be necessary that the glitch duration G_{Width} should not be longer than one clock period. Especially with faster microcontrollers at higher clock speeds, the necessarily shorter glitch durations are becoming increasingly challenging. For a successful glitch, the parameters glitch duration G_{Width} and glitch depth $V_{G_{Low}}$ must therefore be as freely adjustable as possible.

In the following, we show that the selection of glitch parameters for the conventional glitch generation method is significantly limited through capacitive charging and discharging effects in the glitch target device. Taking the exemplary schematic for the conventional method (Figure 3.2), the limitations and the resulting tradeoff can be described as follows. Resistor R_1 is necessary to avoid a short-circuit during glitch generation. The glitch target device includes capacitive effects so that during a glitch, the target device needs to be charged or discharged until it reaches the glitch voltage ($V_{G_{Low}}$). If a low resistance is chosen for R_1, pulling down the voltage will lead to a high current flow where the MOSFET and the resistor build a voltage divider. The advantage of using a low resistance at R_1 is the higher supply current that can be used to reach $V_{G_{High}}$ after a glitch. The necessary discharging (G_{Fall}) and charging time (G_{Rise}) is thus reduced. Unfortunately, this advantage comes at the cost that due to the voltage divider, the achievable glitch voltage ($V_{G_{Low}}$) will be significantly higher. In the worst case, the lowest possible glitch voltage will not be sufficient to produce a successful glitch. On the other hand, if a high resistance is chosen for R_1, the achievable glitch voltage ($V_{G_{Low}}$) will be lower but at the cost of a significantly longer charging time (G_{Rise}). The minimum length of the glitch (G_{Width}) is thus significantly limited. In the worst case, the achievable glitch length will be longer than a clock period and hence a single instruction can no longer be targeted with a glitch. For this reason, obtaining a short glitch (G_{Width}) and a low glitch voltage ($V_{G_{Low}}$) at the same time is not feasible in presence of capacitive effects. Instead, the relationship between these parameters is a tradeoff and the attacker can only optimize one of those parameters at a time.

To demonstrate this tradeoff, we conducted two SPICE simulations with the LTSpice[1] tool: Simulation A represents the case where a higher resistance is chosen for R_1 so that a lower glitch voltage ($V_{G_{Low}}$) can be reached at the cost of a higher charging time (G_{Rise}). In contrast, in Simulation B a lower resistance is chosen for R_1 to achieve a faster charging time at the cost of a higher glitch voltage ($V_{G_{Low}}$). Table 4.1 provides an overview of the used components and their values for these two test cases.

In Figure 4.2, the solid waveform shows the simulation results for Simulation A with an R_1 value of 0.5 Ω while the dotted waveform shows the results of Simulation B with a

[1] `http://www.linear.com/designtools/software/`

Component	**Simulation A**	**Simulation B**
Resistor R_1	0.5 Ω	0.1 Ω
Resistor R_2	10 kΩ	10 kΩ
Capacitor C_1	560 pF	560 pF
MOSFET Q_1	IRF7821 [37]	IRF7821 [37]
Power supply voltage level	3.3 V	3.3 V
Glitch signal voltage level high	3.3 V	3.3 V
Glitch signal voltage level low	0.0 V	0.0 V
Glitch signal turn on time	20 ns	20 ns
Glitch signal rise time	5 ns	5 ns
Glitch signal fall time	5 ns	5 ns
Glitch signal on time	10 ns	10 ns

Table 4.1: Components and Values used for the Conventional Method LTSpice Simulations A and B

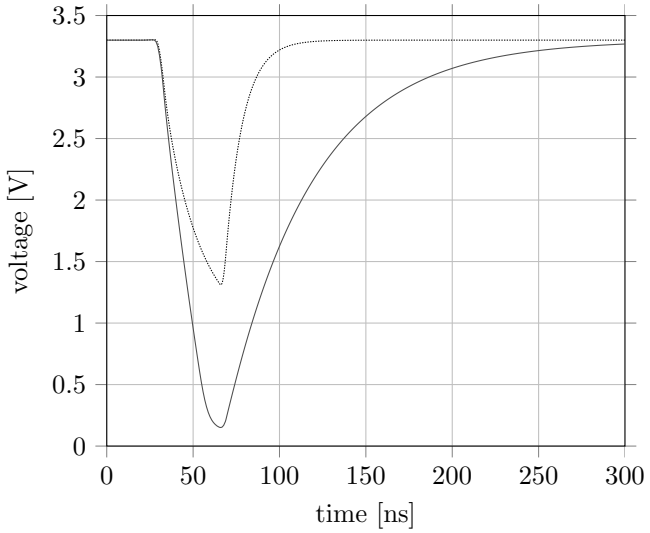

Figure 4.2: Comparison between Simulation A (solid) and Simulation B (dotted)

lower R_1 value of 0.1 Ω instead. The capacitance of the capacitor C_1 is intentionally chosen high, so that the limitations can be easier illustrated. The described tradeoff between the lowest possible glitch voltage ($V_{G_{Low}}$) and the shortest possible glitch time (G_{Width}) can be easily seen. For a successful attack, the slew rate (i.e., the glitch falling time G_{Fall} as well as the glitch rising time G_{Rise}) has to be improved in presence of the capacitance and the following limitations.

4.3 Expected Results of Negative Voltage Fault Injection Attacks

The hypothesis of this work is that negative voltage fault injection attacks provide advantages over their conventional counterparts with respect to higher slew rates and shorter glitch durations in presence of capacitive and inductive charges within microcontrollers. Utilizing negative voltage during the generation of a fault, higher slew rates are expected due to the faster discharging of the circuit implementations within microcontrollers.

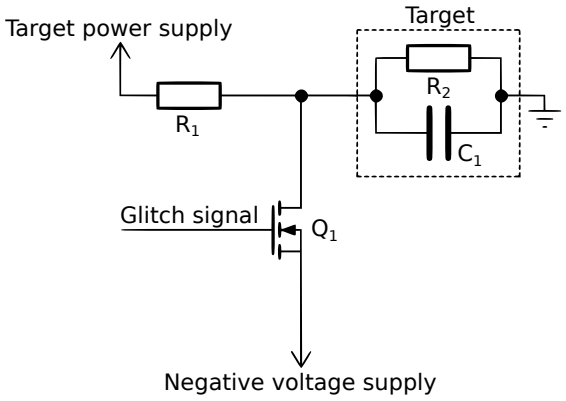

Figure 4.3: Extended Conventional Glitch Generation Method

To consider negative glitch voltages, we extend the conventional glitch generation method with a negative voltage supply as illustrated in Figure 4.3. Similar to Simulation B in the previous section, the current limitation is minimized by using a very small resistance for R_1. Although this minimization leads to an idealized very high current flow that is not feasible in practice (see Section 5.2.1 for details), the model allows to highlight the key concepts of negative voltage faults. In contrast to the conventional method, the source of MOSFET Q_1 is not connected to GND, but to a variable negative voltage supply instead.

To compare the conventional voltage fault injection method with negative voltage fault injections, we conducted two simulations with different glitch signal on times (GS_{On}). In Simulation C, we use a glitch signal on time of 10 ns while Simulation D uses a significantly shorter time of 2.1 ns. The components and values used in the simulations are illustrated in Table 4.2.

Simulation C is visible in Figure 4.4. The solid waveform shows the effect of the negative voltage glitch. Similarly to the conventional voltage fault injection, the resistor R_1 and the MOSFET build a voltage divider so that the voltage $V_{G_{Low}}$ does not reach the negative supply voltage level. However by choosing the negative supply voltage level accordingly, we can arbitrarily select $V_{G_{Low}}$. The high voltage differential between the power supply voltage level $V_{G_{High}}$ and the glitch voltage $V_{G_{Low}}$ causes a low fall time G_{Fall}. After the glitch, a low rise time (G_{Rise}) is achieved due to the low resistance of R_1. In contrast, the dotted and dot-dashed waveforms show the results of the simulation

Component	Simulation C	Simulation D
Resistor R_1	0.1 Ω	0.1 Ω
Resistor R_2	10 kΩ	10 kΩ
Capacitor C_1	560 pF	560 pF
MOSFET Q_1	IRF7821 [37]	IRF7821 [37]
Power supply voltage level	3.3 V	3.3 V
Negative supply voltage level	-2.0 V	-2.0 V
Glitch signal voltage level high	3.3 V	3.3 V
Glitch signal voltage level low	0.0 V	0.0 V
Glitch signal turn on time	20 ns	20 ns
Glitch signal rise time	5 ns	5 ns
Glitch signal fall time	5 ns	5 ns
Glitch signal on time	10 ns	2.1 ns

Table 4.2: Components and Values used for the Negative Voltage Fault Injection Method Simulations C and D

with the conventional voltage fault injection method that pulls the supply voltage to GND instead. The dotted waveform represents Simulation A with a high resistance at R_1 whereas the dot-dashed waveform represents Simulation B with a low R_1 resistance. Even though Simulations A, B and C use the same GS_{On} time of 10 ns, the limitations of the conventional method voltage fault injection are visible: Simulation A reaches sufficiently low glitch voltage $V_{G_{Low}}$ but suffers from the long rise time. Similarly, Simulation B has a fast rise time but suffers from the high glitch voltage $V_{G_{Low}}$.
Furthermore, the figure shows that the glitch duration of Simulation C (G_{Width}) lays between the glitch duration of Simulations A and B. However, the time spend on glitch voltage low ($V_{G_{Low}}$) is significantly longer compared to Simulations A and B. In a next step we show that the the glitch voltage low ($V_{G_{Low}}$) can be arbitrarily selected with the glitch signal on time GS_{On}.

Simulation D is visible in Figure 4.5. The solid waveform shows the effect of the negative voltage glitch. In contrast to Simulation C, the glitch signal on time GS_{On} is much shorter, which results in a short glitch duration (G_{Width}). Apart from that, the same values and parameters as in Simulation C are used. Due to the shortened glitch signal on time GS_{On}, the glitch voltage low $V_{G_{Low}}$ is approximately 0.2 V. The slew rate for the falling and the rising edge is the same as for Simulation C. The dotted waveform represents Simulation A and the dot-dashed waveform represents Simulation B. Both of them use the same values and parameters as described for Figure 4.4. The comparison of Simulation A, B and D shows that negative fault injection can eliminate the disadvantage of conventional voltage fault injection. Although a low resistance for R_1 is chosen to reach a low rise time (G_{Rise}), the glitch voltage low ($V_{G_{Low}}$) can be arbitrarily selected. As a consequence, for the negative voltage fault injection a short glitch (G_{Width}) and a low glitch voltage ($V_{G_{Low}}$) at the same time is not a tradeoff different than for the

conventional voltage fault injection.

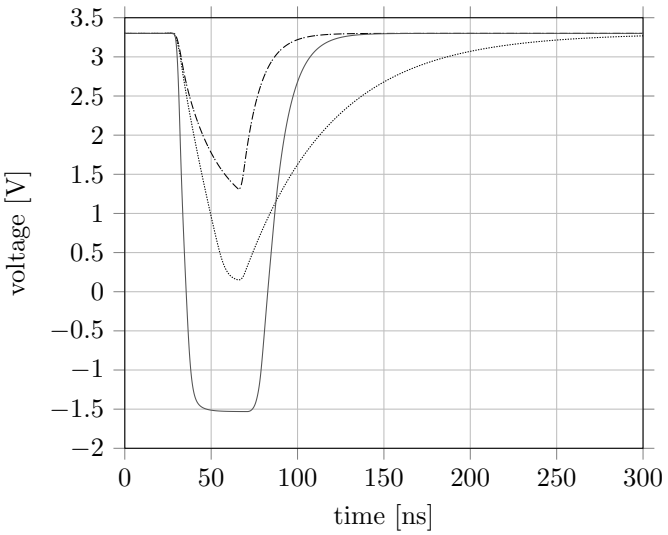

Figure 4.4: Comparison between Simulation A (dotted), Simulation B (dot-dashed) and Simulation C (solid)

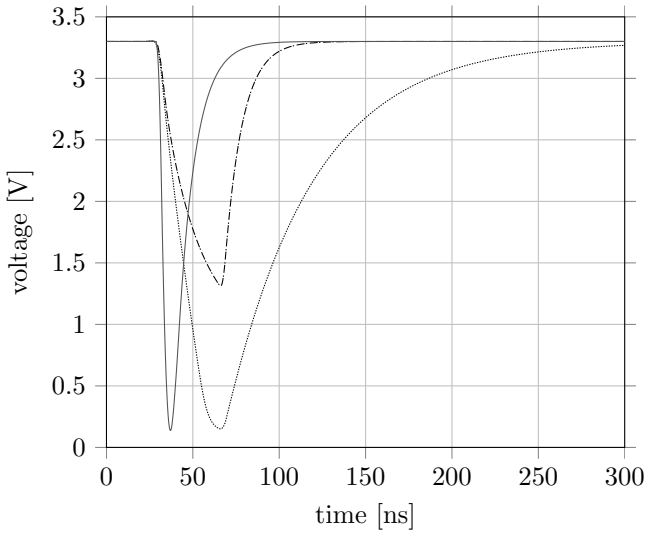

Figure 4.5: Comparison between Simulation A (dotted), Simulation B (dot-dashed) and Simulation D (solid)

CHAPTER 5

Negative Voltage Fault Injection Hardware

This chapter describes the hardware prototype implementation. In a first step, the requirements are specified. Different approaches and ideas for negative voltage glitch generation are explored and evaluated in electronic SPICE simulations. Based on the results of these simulations, the requirements for electronic components are specified and hardware prototypes for negative voltage fault injection attacks are implemented. During prototyping, printed circuit boards have been constructed, manufactured and assembled. We evaluated the different prototypes and selected the most promising approach.

5.1 Hardware Requirements

The hardware requirements are based on the simulation results from the previous chapter. The most important requirement is to increase the slew rate and thus to minimize the glitch fall time (G_{Fall}) as well as the glitch rise time (G_{Rise}). Another important requirement is that the glitch width (G_{Width}) needs to be controlled via the glitch signal width (GS_{Width}). The glitch width (G_{Width}) should be variably selectable. It should be at least 31.25 ns long so that microcontrollers up to 32 MHz can be tested. Furthermore, it should be possible to insert a series of glitches in short intervals. The glitch voltage high ($V_{G_{High}}$) should be 3.3 V, which is the default power supply voltage for modern microcontrollers [38]. The glitch voltage low ($V_{G_{Low}}$) should be variably selectable between 0.0 V and -6.0 V. The incoming logical glitch trigger signal has a low level ($V_{GS_{Low}}$) of 0.0 V and a high level ($V_{GS_{High}}$) of 3.3 V. The hardware has to be able to interpret this signal correctly. Moreover, the hardware should use a galvanic isolation to operate the target with a different ground level than the hardware that generates the incoming glitch trigger signal. This provides the advantage that during operation it is not necessary to pay attention to different ground potentials. As a result, the incoming

glitch trigger signal as well as the different voltage levels, either produced internally or with an external power supply, must be isolated.

5.2 Design Approaches

5.2.1 Design Approach 1: Extended Conventional Circuit

This design approach was introduced in Section 4.3. The schematic is illustrated in Figure 4.3. The target is powered with an external power supply. Parallel to the target, MOSFET Q_1 is placed. Between the power supply and the target, resistor R_1 is required to prevent a short circuit between the power supply and GND. If the logical glitch signal is high, the MOSFET Q_1 drives the power supply line of the target to GND.

In Section 4.3, a peak current of 48.2 A could be measured for Simulation C with a glitch signal on time (GS_{On}) of 10 ns and a resulting glitch width (G_{Width}) of 105.21 ns. In contrast, for Simulation D with a glitch signal on time (GS_{On}) of 2.1 ns and a resulting glitch width (G_{Width}) of 69.85 ns, a peak current of 43.7 A could be measured.

For both simulations, the required glitch width of 31.25 ns, as defined in Section 5.1, couldn't be fulfilled.

Furthermore, due to the necessary minimization of resistor R_1 to 0.1 Ω, the idealized high current flow is infeasible in practice. Typical MOSFETs with fast switching characteristics have a far lower continuous drain current of approximately 15 A. For instance, the Infineon IRF7821 [37] MOSFET has a continuous drain current of 13.6 A. Since we want to meet the hardware requirement that the glitch width (G_{Width}) should be at least 31.25 ns long (see Section 5.1), we need fast MOSFETs.

5.2.2 Design Approach 2: NMOS-PMOS circuit

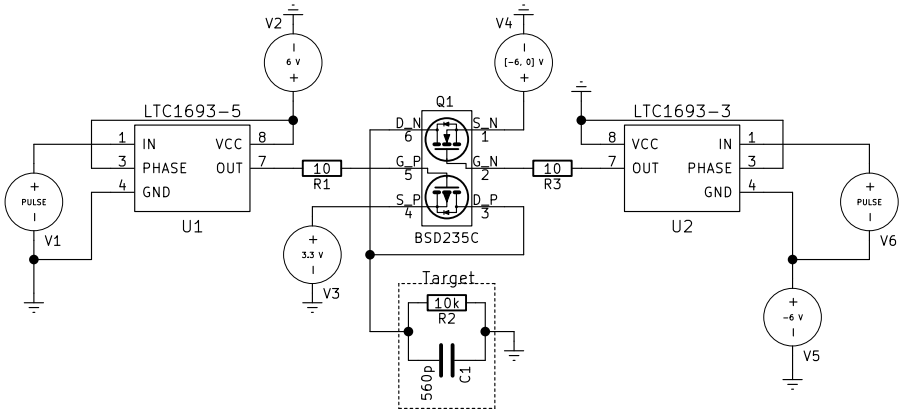

Figure 5.1: NMOS-PMOS Circuit for the Second Design Approach.

The idea of the second approach is to switch between two voltage sources. The first one provides the operating voltage required by the target. The second one can be arbitrarily adjusted between -6.0 V and 0.0 V. To insert a glitch, the power source is switched from the first one to the second one for an arbitrary amount of time. Figure 5.1 illustrates this design approach. An n-type and a p-type MOSFET, hereinafter described as NMOS and PMOS, are used to switch between the two voltage sources. As long as no glitch is injected, the NMOS isn't active and the PMOS provides the operating supply voltage. If a glitch is injected into the supply voltage, the PMOS is switched off and the NMOS is switched on to inject the glitch. MOSFET drivers are used to produce high-current drive input for the gates to ensure high slew rates [39].

For the transistors, the Infineon BSD235C [40] type is used. It provides a rise time of 5.0 ns for the PMOS and a rise time of 3.6 ns for the NMOS. The Linear Technology LTC1693-5 [41] and LTC1693-3 [42] are used as PMOS and NMOS drivers, respectively. As illustrated, both drivers have different pins. The pin IN (Pin 1) is a driver input independent from V_{CC}. The glitch signal is connected to this pin. The V_{CC} pin (Pin 8) is the power supply input. It must be between 4.5 V and 13.2 V. The output pin (Pin 7) is the driver output. When the logic signal is low, the voltage at the output is equal to the GND voltage. If the logic signal is high, the voltage at the output is equal to the V_{CC} voltage. Since the PHASE pin (Pin 3) is not used, it is connected to the V_{CC} pin as recommended in the datasheet. The current between the MOSFET drivers and the transistor gates is limited by the resistors R_1 and R_3 to protect the gates.

As explained in Section 2.2.1, in n-type MOSFETs the current between drain and source can only flow if the voltage U_{GS} between gate and source is positive and higher as the threshold U_{TH} ($U_{TH} > 0V$). In p-type MOSFETs, if the voltage U_{GS} between gate and source is negative and lower as the threshold U_{TH} ($U_{TH} < 0V$), current can flow from source to drain. According to the datasheet, the threshold U_{TH} for the NMOS is 0.95 V and for the PMOS it is -0.9 V. In the following, the boundary values for this design approach are calculated to check if the approach is technically feasible.

The terms $U_{GS_{Low}}$ and $U_{GS_{High}}$ are used below. $U_{GS_{Low}}$ is the gate source voltage of a MOSFET if no glitch is injected, and $U_{GS_{High}}$ is the gate source voltage of a MOSFET if a glitch is injected.

For the PMOS, the cases of an active and non-active glitch signal need to be considered where 3.3 V are applied at their respective source inputs. Since $U_{GS_{Low}}$ is negative ($U_{GS_{Low}} = 0 - 3.3 = -3.3$ V) and lower as the threshold U_{TH} (-3.3 V < -0.9 V), the PMOS is active and the target is supplied with the voltage from power supply V_3. In comparison, $U_{GS_{High}}$ is positive ($U_{GS_{High}} = 6 - 3.3 = 2.7$ V) and therefore the PMOS is not active if a glitch is injected. The PMOS would thus work as expected.

Since the voltage of the negative voltage source can be between -6.0 V and 0.0 V, there are two scenarios that need to be analyzed individually. For both scenarios, the case of an active and non-active glitch signal needs to be considered. First, we observe the case where -6.0 V is applied at the source of the MOSFET. Since $U_{GS_{Low}} = -6.0 - -6.0 = 0.0$

V is lower as the threshold U_{TH} (0.0 V < 0.95 V), the NMOS is not active. In comparison, $U_{GS_{High}}$ is positive ($U_{GS_{High}} = 0 - -6.0 = 6.0$ V) and higher as the threshold U_{TH} (6.0 V > 0.95 V). As a consequence if a glitch is injected, the NMOS is active and provides the voltage of the negative voltage supply to the target.
In our second case, 0.0 V is applied at the source of the MOSFET. Since $U_{GS_{Low}}$ is negative ($U_{GS_{Low}} = -6.0 - 0.0 = -6.0$ V), the NMOS is not active if no glitch is injected. However, $U_{GS_{High}}$ is positive ($U_{GS_{High}} = 0 - 0.0 = 0.0$ V) and lower as the threshold U_{TH} (0.0 V < 0.95 V). As a result, the NMOS is not active if a glitch is injected. The target would thus float since it is neither supplied with the positive nor with the negative voltage source. In order for the MOSFET to be active, in this scenario the voltage at the source would at least have to be at a lower voltage as $-U_{TH}$. However, this violates the requirement of a variable negative voltage source between -6.0 V and 0.0 V as defined in section Section 5.1. The design approach is thus not feasible.

5.2.3 Design Approach 3: NMOS circuit

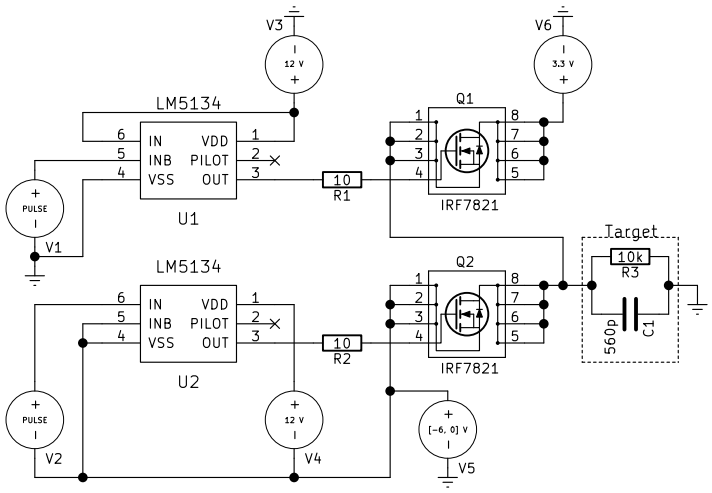

Figure 5.2: NMOS circuit for the third design approach.

Similarly to the previous approach, the general idea of this approach is to switch between two voltage sources. The first supply provides the operating voltage required by the target. The second voltage can be arbitrarily adjusted between -6.0 V and 0.0 V. To insert a glitch, the power source is switched from the first one to the second one for an arbitrary amount of time. Figure 5.2 illustrates the design approach.

Instead of a p-type and an n-type MOSFET, this design uses two identical n-type MOSFETs IRF7821 [37]. As long as no glitch is injected, NMOS Q_1 is active and NMOS Q_2 is inactive. As a consequence, the voltage source V_6 provides 3.3 V to the target. If a glitch is injected, NMOS Q_1 is inactive and NMOS Q_2 is active. The active NMOS thus connects the negative voltage source V_5 to the target. This MOSFET switching behavior is achieved by the LM5134 [43] MOSFET drivers U_1 and U_2. They are equipped with a

noninverting and inverting signal input. If the input signal is applied to the IN pin while the INB Pin is connected to V_{SS}, the OUT pin is low if no glitch is inserted and high if a glitch is inserted. In contrast, when the input signal is applied to the INB pin while the IN pin is connected to V_{DD}, the OUT pin is high if no glitch is inserted and low if a glitch is inserted. To achieve high slew rates, the output (OUT pin) high signal of the the drivers U_1 and U_2 is equal to 12.0 V relative to the GND of the drivers. Since the applied voltage at the source of NMOS Q_2 must be arbitrary selectable and the GND of MOSFET driver U_2 is connected to the source of NMOS Q_2, the GND of the driver U_2 is shifted to the voltage level of the negative voltage source V_5. The current between the MOSFET drivers and the transistor gates is limited by the resistors R_1 and R_2 to protect the gates.

As explained in Section 2.2.1, in n-type MOSFETs current between drain and source can only flow if the voltage U_{GS} between gate and source is positive and higher as the threshold U_{TH} ($U_{TH} > 0V$). According to the IRF7821 datasheet [37], the threshold U_{TH} is 1.0 V. In the following, the boundary values for this design approach are calculated to determine if the approach is feasible in practice.

First, we consider the case where no glitch is injected. In this case, the output of the driver U_1 is 12.0 V and U_2 is 0.0 V relative to V_5. For Q_1, $U_{GS_{Low}} = 12.0 - 3.3 = 8.7$ V is always positive and higher as the threshold U_{TH}. For Q_2: If V_5 is -6.0 V $U_{GS_{Low}} = -6.0 - -6.0 = 0.0$ V. In the other case, if V_5 is 0.0 V, $U_{GS_{Low}} = 0.0 - 0.0 = 0.0$ V. $U_{GS_{Low}}$ is for both cases 0.0 V, since the gate driver voltage is referenced to source instead of GND. As a result, Q_2 is not active and Q_1 is active, supplying the target with 3.3 V.
Second, we consider the the case where a glitch is injected. In this case, the output of the driver U_1 is 0.0 V and U_2 is 12.0 V relative to V_5. We assume the voltage on source Q_1 is 3.3 V. Since $U_{GS_{High}} = 0.0 - 3.3 = -3.3$ V is negative and lower as the treshhold U_{TH}, Q_1 is not active. For Q_2: If V_5 is -6.0 V, $U_{GS_{High}} = 6.0 - -6.0 = 12.0$ V. In the other case, if V_5 is 0.0 V $U_{GS_{High}} = 12.0 - 0.0 = 12.0$ V. $U_{GS_{High}}$ is for both cases 12.0 V since the gate driver voltage is referenced to source instead of GND. As a result, Q_2 is active, pulling the target to the negative voltage source V_5. From the theoretical view, the design approach would thus work as expected.

To test the behavior of the design approach and to measure the peak currents, we conducted SPICE Simulations E and F. Table 5.1 provides an overview of the used components and their values. In both simulations the same values and components are used. The only exception is that Simulation E is simulated with a negative supply voltage level (V_5) of 0.0 V, while Simulation F uses a negative supply voltage level (V_5) of -6.0 V. The results of the simulations are visible in Figure 5.1. The solid waveform represents the glitch signal for both simulations, the dotted waveform shows the result of Simulation E and the dot-dashed waveform illustrates the result of Simulation F. As in previous simulations, the target is simulated with a 10 kΩ resistor R_3 and a 560 pF capacitor C_1 in parallel.

For both simulations, the measured currents are within the maximum ratings specified in the IRF7821 datasheet [37]. In the following, the results of Simulation E are described.

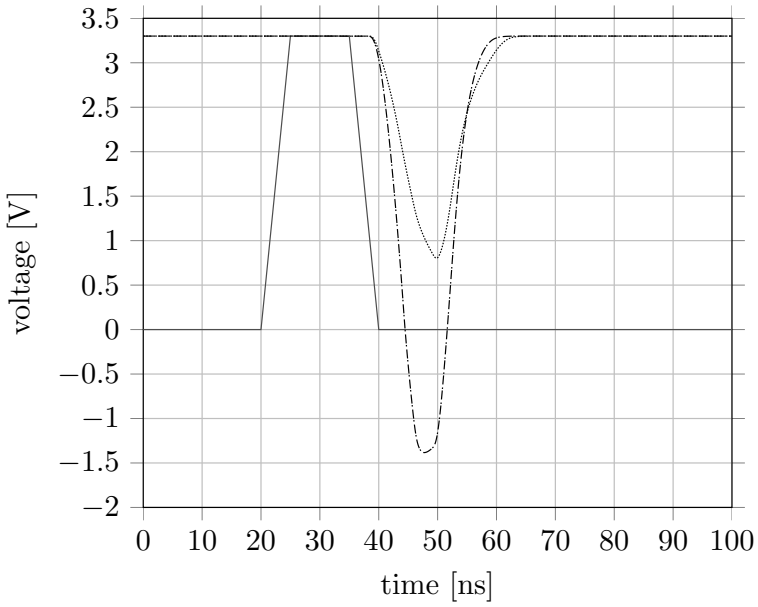

Figure 5.3: NMOS Design Approach Simulation: Glitch Signal (solid), Simulation E (dotted) and Simulation F (dot-dashed)

The glitch offset (G_{Offset}) (i.e. the time period between the moment the glitch signal rises from $V_{GS_{Low}}$ and the moment the voltage level of the power supply line voltage falls from $V_{G_{High}}$) is 18.76 ns. After 11.09 ns, a $V_{GS_{Low}}$ voltage drop to 0.79 V is achieved. The short time of 10 ns GS_{On} is not sufficient to reach the negative supply voltage level V_5. Thereafter, the supply voltage is pulled to $V_{G_{High}}$ within 13.31 ns. This results in a glitch width (G_{Width}) of 24.40 ns.
In contrast, the following values can be measured for Simulation F: As in Simulation E, the glitch offset(G_{Offset}) is 18.76 ns. For the glitch fall time (G_{Fall}), a value of 9.19 ns can be measured. The glitch rise time (G_{Rise}) is 13.62 ns. This results in a glitch width (G_{Width}) of 22.81 ns. The glitch voltage low ($V_{G_{Low}}$) is -1.37 V due to the glitch signal on time (GS_{On}) of 10 ns being too short to reach the negative power supply voltage level V_5 of -6.0 V.
In summary, the two simulations show that the third design approach works as expected and that the hardware requirements specified in Section 5.1 can be fulfilled.

Component	**Simulation E**	**Simulation F**
Resistor R_1	10 Ω	10 Ω
Resistor R_2	10 Ω	10 Ω
Resistor R_3	10 kΩ	10 kΩ
Capacitor C_1	560 pF	560 pF
MOSFET Q_1	IRF7821 [37]	IRF7821 [37]
MOSFET Q_2	IRF7821 [37]	IRF7821 [37]
MOSFET Driver U_1	LM5134 [43]	LM5134 [43]
MOSFET Driver U_2	LM5134 [43]	LM5134 [43]
Power supply voltage level V_6	3.3 V	3.3 V
Negative supply voltage level V_5	0.0 V	-6.0 V
Glitch signal voltage level high	3.3 V	3.3 V
Glitch signal voltage level low	0.0 V	0.0 V
Glitch signal turn on time	20 ns	20 ns
Glitch signal rise time	5 ns	5 ns
Glitch signal fall time	5 ns	5 ns
Glitch signal on time	10 ns	10 ns

Table 5.1: Components and Values used for the third Design Approach Simulations E and F

5.2.4 Selection of Design Approach

While the first design approach (Section 5.2.1) is suitable to understand the key concepts of negative voltage fault injection, it is only a theoretical design approach that is not feasible in practice. As indicated, the glitch width and the maximum currents are significant limitations.

The second design approach (Section 5.2.2) is also not feasible, since the negative voltage can not freely be adjusted between -6.0 V and 0.0 V.

No limitations were found for the third design approach (Section 5.2.3). The theoretical view as well as the simulations showed that the hardware requirements, specified in Section 5.1, can be fulfilled with this approach.

5.3 Implementation of Prototype

Figure 5.4 illustrates the prototype implementation. The glitch signal is inserted externally via an SMA connector. The ADuM1100 [44] digital isolators U_1 and U_2 are used to shift the logic level of the glitch signal to the level required by the MOSFET drivers. In addition, the isolators fulfill the hardware requirement that the glitch signal is galvanically isolated (Section 5.1). The isolators transfer the incoming logic signal IN with the voltage level VDD_1 to the outgoing logic signal OUT with the voltage level VDD_2. Since the incoming glitch signal is equal to 3.3 V, the voltage supply VDD_1 must also be 3.3 V.

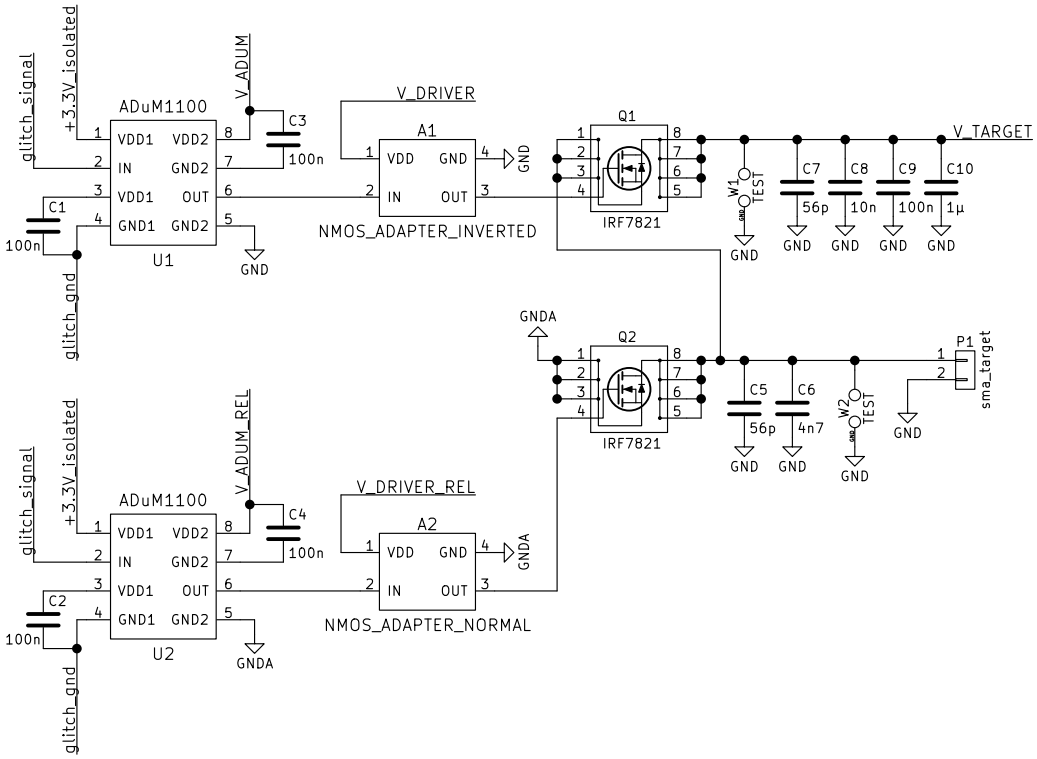

Figure 5.4: Schematic of the Prototype

This voltage is provided by the signal +3.3V_isolated which is generated by a linear voltage regulator. Actually, only the isolator U_2 would be necessary, since the glitch signal for the MOSFET driver of the MOSFET Q_1 does not have to be transferred. However, the glitch signal must arrive at both drivers as concurrently as possible. To achieve an approximately equal delay, isolator U_1 is necessary. For that reason, the voltage V_ADUM is 3.3 V. For the MOSFET driver of MOSFET Q_2, the glitch signal must be transferred to 3.3 volts relative to GNDA. This is achieved with the voltage V_ADUM_REL provided to VDD_2 of isolator U_2. GNDA can be adjusted between -6.0 V and 0.0 V by means of a potentiometer. Thus, the requirement of the arbitrary negative voltage source (see Section 5.1) is achieved. The capacitors C_1, C_2, C_3 and C_4 with a capacitance of 100 nF are used as recommended by the data sheet [44]. In order to leave open the possibility to test different MOSFET drivers, the drivers are not directly placed on the prototype PCB. Instead, they are placed on adapter boards A_1 and A_2. The adapter boards are connected to the prototype via pin headers. The input of the adapter boards is connected to the output of the isolators, and the output of the adapter boards is connected to the gates of the MOSFETs. Furthermore, the adapter boards are supplied with a supply voltage and the corresponding GND level. The adapters are explained in detail after the description of the prototype. The two NMOS Q_1 and Q_2 are alternately active and supply the target through the SMA connector sma_target

(P_1). The target is thus supplied either with 3.3 V (V_TARGET) or with a negative voltage between -6.0 V and 0.0 V (GNDA). In contrast to design approach 3 (Section 5.2.3), for the prototype the decoupling capacitors C_{5-10} are used to prevent ringing on the supply voltage of the target at the moment a glitch is inserted. The values for the decoupling capacitors were chosen according to best practice recommendations [45]. To test the prototype, test points W_1 and W_2 are provided with a special mount for the probes of an oscilloscope.

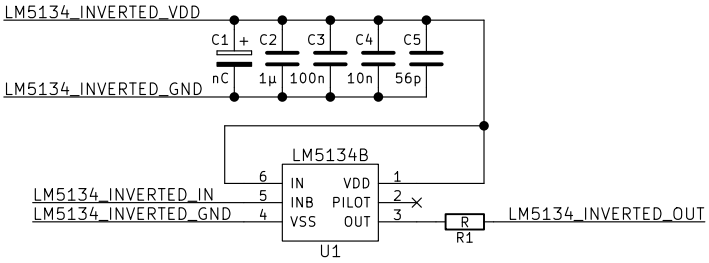

Figure 5.5: Schematic of the Inverted Adapter

In the following, the adapters mentioned above are explained. Both adapters use the MOSFET driver LM5134 [43]. At the first adapter, the input signal is connected to the inverted input (INB). For this reason, the first adapter is an inverted adapter while the second one is a normal (non inverted) adapter.

Figure 5.5 shows the schematic for the inverted adapter. The driver U_1 is powered by the supply voltage LM5134_INVERTED_VDD. The decoupling capacitors C_{1-5} ensure a stabilization of the voltage. The glitch signal, which is already shifted to the required logic level, is inserted at input INB. As a result, if the level of the glitch signal is low (e.g., no glitch is injected), the output (OUT) is equal to LM5134_INVERTED_VDD. If the level of the glitch signal is high (e.g., a glitch is injected), the output (OUT) is equal to LM5134_INVERTED_GND. The current between the driver U_1 and the MOSFET gate is limited by the resistor R_1 to protect the gate.

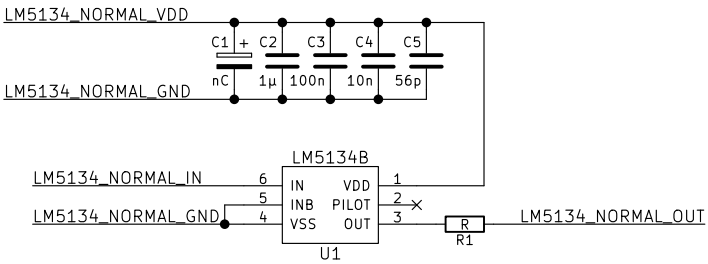

Figure 5.6: Schematic of the Normal Adapter

Figure 5.6 shows the schematic for the normal (non inverting) adapter. The driver U_1 is powered by the supply voltage LM5134_NORMAL_VDD. The decoupling capacitors C_{1-5} ensure a stabilization of the voltage. The glitch signal, which is already shifted to

the required logic level, is inserted at input IN. As a result, if the level of the glitch signal is low (e.g., no glitch is injected), the output (OUT) is equal to LM5134_NORMAL_GND. If the level of the glitch signal is high (e.g., a glitch is injected), the output (OUT) is equal to LM5134_NORMAL_VDD. Similarly to the inverted adapter, the current between the driver U_1 and the MOSFET gate is limited by the resistor R_1 to protect the gate.

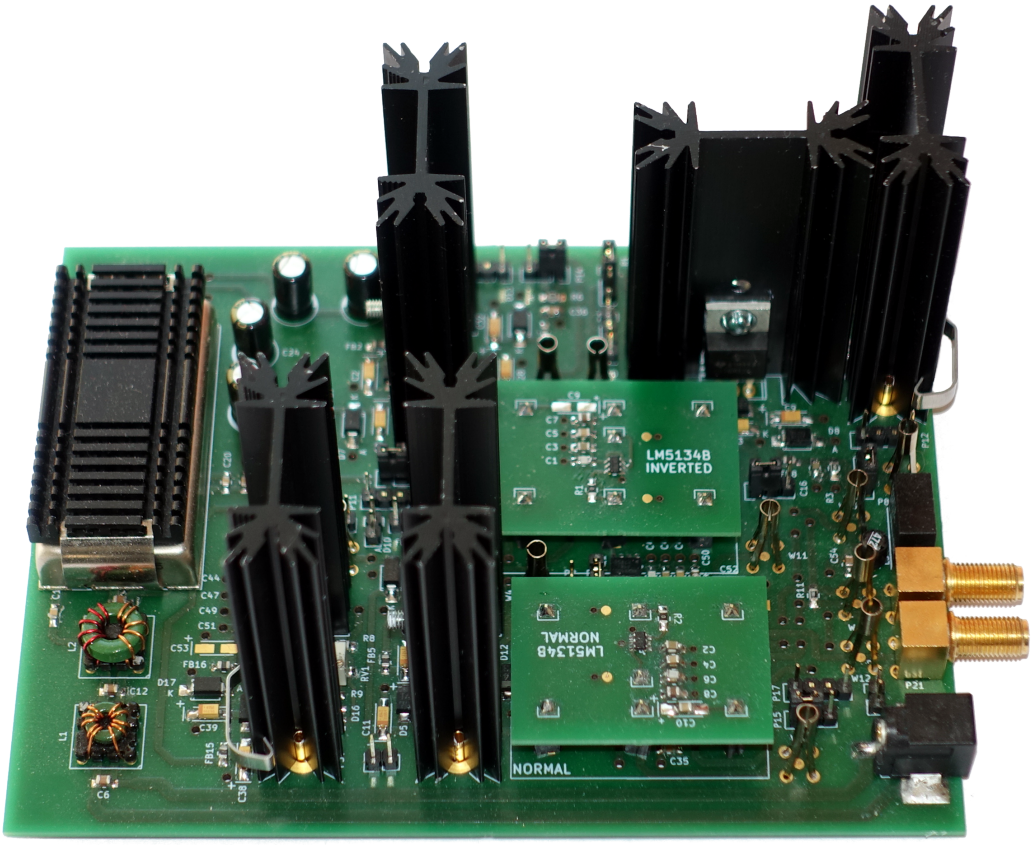

Figure 5.7: Image of the Final Prototype

We designed the printed circuit boards for the prototype and the adapters with KiCad[1], a well known electronic computer-aided design (ECAD) suite. The PCB footprints are attached in Appendix A.1. We have commissioned the production of the printed circuit boards to the PCB manufacturer PCB-POOL[2]. After receiving the printed circuit boards and all required components, the prototype was assembled. The final prototype can be seen in Figure 5.7.

[1] http://kicad-pcb.org/
[2] http://www.pcb-pool.com/

5.4 Evaluation of Prototype

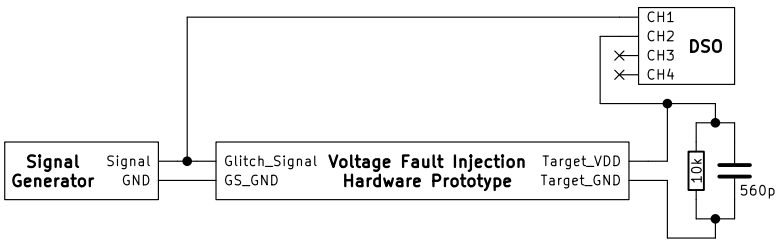

Figure 5.8: Prototype Evaluation Test Setup

To verify the functionality of the prototype, we used the test setup illustrated in Figure 5.8. A signal generator generates a pulsed signal with a GS_{Rise} time of 5 ns, a GS_{On} time of 10 ns and a GS_{Fall} time of 5 ns. This signal is used as reference glitch trigger signal and it is thus connected to the glitch signal input of the prototype. The output of the prototype is connected to a resistor and a capacitor in parallel. To check the behavior of the prototype, the glitch signal and the output of the prototype are connected to a digital storage oscilloscope.

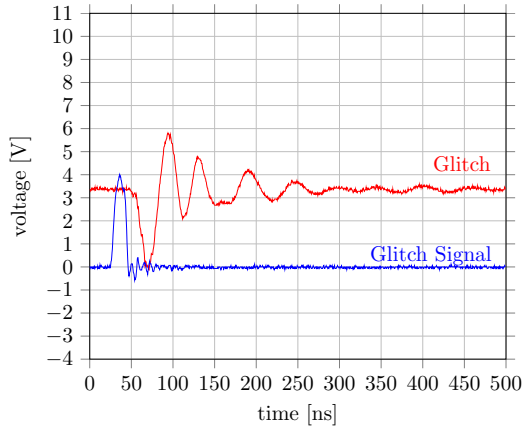

Figure 5.9: Result of the Protoype Test with a GNDA of 0.0 V

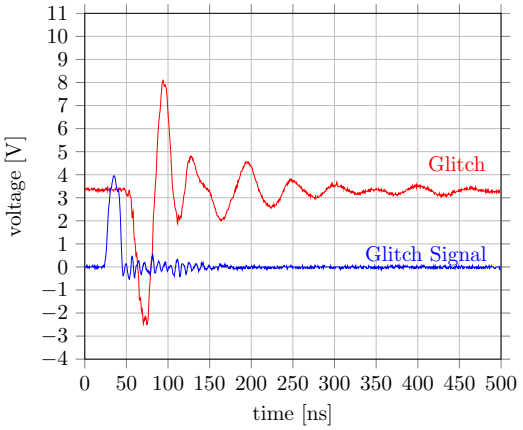

Figure 5.10: Result of the Protoype Test with a GNDA of -6.0 V

Figure 5.9 illustrates the result of the prototype test with a GNDA of 0.0 V. The blue waveform shows the glitch signal generated by the signal generator. The glitch signal width (GS_{Width}) is about 20 ns. The red waveform shows the output of the prototype with the injected glitch. The glitch width (G_{Width}) is about 35 ns. The oscillation of the glitch it due to the Gibbs phenomenon [46] and can not be avoided. The glitch reaches a $V_{G_{Low}}$ of 0.0 V.

Figure 5.10 illustrates the result of the prototype test with a GNDA of -6.0 V. The blue waveform shows the glitch signal generated by the signal generator. The glitch signal

width (GS_{Width}) is about 20 ns. The red waveform shows the output of the prototype with the injected glitch. The glitch width (G_{Width}) is about 30 ns. The glitch reaches a $V_{G_{Low}}$ of -2.5 V.

The results of the prototype test are comparable to the results of simulations E and F (see Section 5.2.3). Therefore, we can derive that the prototype fulfills the requirements specified in Section 5.1.

CHAPTER 6

Evaluation

As described in Section 4.3, the hypothesis of this work is that negative voltage fault injection attacks provide advantages over their conventional counterparts with respect to higher slew rates and shorter glitch durations in presence of capacitive and inductive charges within microcontrollers. This chapter describes the evaluation process to validate the hypothesis. The first part of the chapter explains the test setup. Thereafter, the individual test cases are described.

6.1 Test Setup

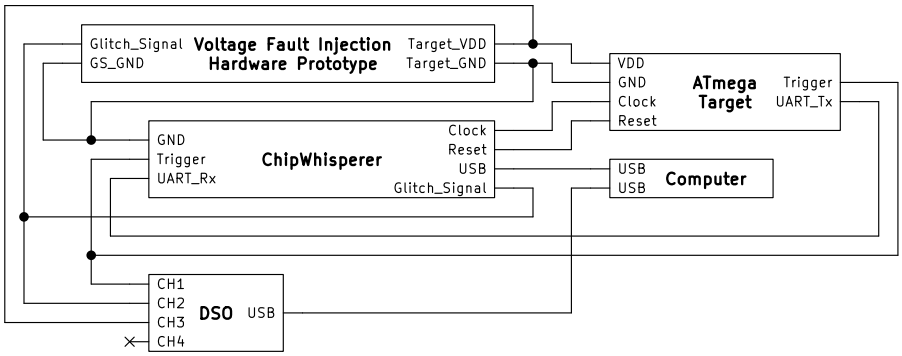

Figure 6.1: Evaluation Test Setup

Figure 6.1 shows the test setup for the evaluation. Each component of the test setup is shown as a block, with the name of the component at its center. For all components, incoming signals or voltages are located on the left side of the block whereas outgoing signals or voltages are located on the right side. In the following, each component is explained in detail.

6.1.1 Target

Hardware

For the evaluation, the selection of a common microcontroller is crucial. For this reason, we selected the ATmega328P [47]. It is used in the open-source electronic prototyping platform Arduino[1] [48]. Furthermore, it is a well known target for voltage fault attacks, it is covered in multiple scientific publications [49, 50, 51] and it thus allows the comparison of our evaluation results with previous work.

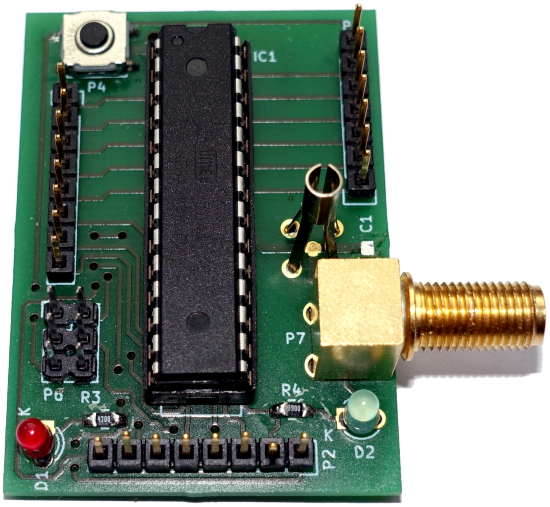

Figure 6.2: Image of the ATmega328P Target

Figure 6.2 shows an image of the ATmega328P microcontroller and its periphery assembled on a self-designed printed circuit board. Figure 6.3 illustrates the corresponding schematic. The footprint of the printed circuit board is attached in Appendix A.2.

The microcontroller is powered by the SMA socket P_7. To be able to observe the voltage drop during a glitch, the test point W_1 is provided with a special mount for the probe of an oscilloscope. Between the power supply and the GND rail, the decoupling capacitor C_1 is installed as recommended in the data sheet [47]. However during the evaluation, some tests were performed with the decoupling capacitor removed C_1 as the added capacitance generally hinders voltage fault injection attacks and, in general, it is beneficial to remove the decoupling capacitors if possible [52].

The connections to the pins of the microcontroller are available via the pin headers P_2, P_3 and P_4. In addition, the programming pins (MISO, MOSI, SCK, RESET1, +3.3V and GND) are connected to the pin header P_6. According to the datasheet [47], the microcontroller can be reset if the RESET1 pin is tied to GND. This is achieved with push button SW_1. The pull-up resistor R_2 prevents the RESET1 signal from floating.

[1] `https://www.arduino.cc/`

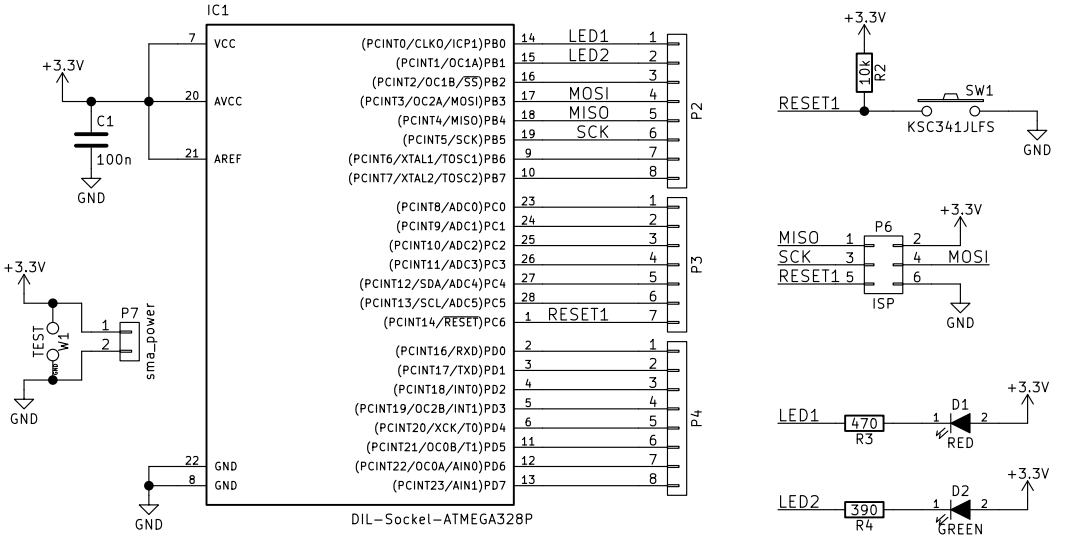

Figure 6.3: Schematic of the ATmega328P Target

Furthermore, the target board contains Light-emitting diodes(LEDs) D_1 and D_2 with the associated current limiting resistors R_3 and R_4.

In the test setup, the power supply SMA socket P_7 is connected to the voltage fault injection hardware prototype. The Clock, Reset, Trigger and UART_Tx signals are connected to the ChipWhisperer and are explained in the following sections.

Firmware

This section explains firmware parts relevant for the evaluation. The code snippets have been simplified to highlight their key concepts. The complete firmware is attached in Appendix B.6.

```
int main(void) {
    init_uart();
    trigger_setup();

    uart_puts("hello");

    while(1) {
        glitch1();
    }

    return 1;
}
```

Listing 6.1: Firmware – main Function

```
void glitch1(void) {
    volatile uint8_t a = 0;

    putch('A');

    trigger_high();
    trigger_low();

    while(a != 2){
    ;
    }

    uart_puts("1234");
}
```

Listing 6.2: Firmware – glitch1 Function

Listing 6.1 contains the simplified `main` function of the firmware. In a first step the Universal Asynchronous Receiver Transmitter (UART) is initialized. The ATmega328P target can thus send American Standard Code for Information Interchange (ASCII) characters to the ChipWhisperer via the UART_Tx connection as illustrated in Figure 6.1. Afterwards, the trigger pin is initialized. The trigger signal, which is low per default, is used to signal the Chipwhisperer that a particular location in the firmware has been reached. Subsequently, the message "hello" is sent to the ChipWhisperer via the UART connection and the function `glitch1` is executed. Since the called function enters an infinite loop, the end of the main function is never reached.

Listing 6.2 shows the simplified `glitch1` function. At the beginning of the function, the variable `a` is initiated with the value 0. The *volatile* keyword ensures that the compiler does not remove the infinite loop (`while(a != 2)`) during code optimization. Afterwards, the character "A" is transmitted to the ChipWhisperer. Hereafter, the trigger signal is raised for one clock cycle. This signals the ChipWhisperer that the infinite loop will be executed in the next step. Thereafter, the infinite loop is executed. In every loop iteration, the value of variable `a` is validated against the constant value 2. If the result of the validation is true, the code after the loop is executed. This can only happen if the value of variable `a` is changed to 2 by a successful glitch. To inform the ChipWhisperer about the successful glitch, the message "1234" is sent.

Fuse Configuration

The ATmega328P microcontroller stores its configuration in a series of so called fuse bits. They are organized in the three fuse bytes *low*, *high* and *extended*. For each fuse bit, default values are defined and documented in the data sheet [47]. Except for the changes described below, these default values are used during the evaluation. Instead of the internal 8 MHz oscillator, the use of an external clock is configured. This clock is generated by the ChipWhisperer. As a consequence, the generated glitch can be

synchronized to the target clock rate. Furthermore, the ATmega328P has a so-called brownout detection. The brownout detection resets the chip if the power supply voltage drops below a defined threshold. The brownout detection can either be deactivated or activated. If enabled, either 2.7 V or 1.8 V can be set for the threshold. During evaluation, tests with deactivated and activated brownout detections are performed.

6.1.2 Voltage Fault Injection Hardware Prototype

The voltage fault injection hardware prototype is equivalent to the prototype developed in Section 5.3. The prototype is connected to the ChipWhisperer and the ATmega328P target. If the glitch signal is logical high, a glitch is injected to the power supply of the target. In contrast, if the glitch signal is logical low, the target is supplied with 3.3 V. During evaluation, GNDA (described in Section 5.3) is either adjusted to 0 V, -2 V, -4 V or -6 V.

6.1.3 ChipWhisperer

To conduct the evaluation, we need a system to create a glitch signal with the following requirements. From the moment the trigger signal (see Section 6.1.1) is received, a freely selectable wait time is required. This period is called *offset*. Thereafter, the glitch signal should be pulled to logic high for a freely selectable period of time. This period of time is called glitch signal *width*. Afterwards, the glitch signal should be pulled back to logic low. Sometimes a series of glitches should be inserted. The number of glitch signal repetitions is called *repeat*. The reason why the glitch signal *width*, *offset* and *repeat* must be freely selectable is that the correct parameters to break out of the infinite loop (`while(a != 2)`) are not known in advance and they must be identified by using a brute force approach. Since the *offset* and the glitch signal *width* periods are within a nanosecond range, the hardware for generating the glitch signal must be sufficiently fast.

The ChipWhisperer [7] by NewAE Technology Inc.[2] is a commercial solution for conventional voltage fault injection attacks and side channel analysis. All described requirements concerning the glitch signal generation hardware are fulfilled by the ChipWhisperer. Usually, the ChipWhisperer generates a glitch signal to drive a MOSFET. This glitch generation method was described in Section 3.2 and was illustrated in Figure 3.2. To conduct the evaluation, we desoldered the MOSFET Q_1 and connected the glitch signal to the glitch SMA socket of the ChipWhisperer. As a result, we can use the glitch signal with our own prototype.

As mentioned above, the ATmega328P target uses an external clock generated by the ChipWhisperer. For the evaluation, a clock rate of 7.37 MHz is used that corresponds to a clock period of 136 ns. At the ChipWhisperer, the glitch signal *width* is specified in percent relative to the clock period. A glitch signal *width* of 10% at a clock rate of 7.37 MHz corresponds to a glitch signal *width* of 13.6 ns. The *offset* is also specified in

[2] `https://newae.com/`

percent and can be either positive or negative within a range of -49.0% to 49.0%. A positive *offset* indicates that the glitch signal is injected x ns (x = 1.36 ns * offset[%]) after the rising edge of the current clock cycle. In contrast, a negative *offset* indicates that the glitch signal is injected x ns (x = 1.36 ns * offset[%]) before the rising edge of the current clock cycle. The *repeat* value is given as positive integer.

The ChipWhisperer is connected to a PC via the USB interface. On one hand, the connection is used to pass the ASCII characters received via the UART_Tx connection to the PC. On the other hand, the connection is used to configure the parameters *width*, *offset* and *repeat*. Besides, the ChipWhisperer can reset the ATmega328P target via the reset connection.

6.1.4 Digital Storage Oscilloscope

We use the four channel digital storage oscilloscope MSOX4034A[3] to observe the behavior of the test setup. The glitch signal, the trigger signal and the supply voltage of the ATmega328P target are connected to the oscilloscope. Furthermore, the oscilloscope is connected to the PC via a USB interface so that we can access the storage buffer of the oscilloscope from the PC.

6.1.5 PC

Both the ChipWhisperer and the oscilloscope are controlled from the PC. The ChipWhisperer has its own Python API[4] and the oscilloscope can be controlled via the PyVISA[5] Python framework.

As explained above, different parameters for *width*, *offset* and *repeat* have to be tried within a brute force approach to find parameters that cause a jump out of the infinite loop (`while(a != 2)`) at the ATmega328P target. For this reason, we developed a program that automatically tests all possible parameters and writes the results of the glitches to a database. Important code snippets are explained below, with the snippets simplified for a better understanding. The complete program is attached in Appendix B.1. The structure of the database can be found in Appendix C.1.

Listing 6.3 contains the `main` function of the evaluation program. In a first step, a list of all values to be tested is created for each of the three parameters *width*, *offset* and *repeat*. For each parameter, the lower and upper bounds as well as the step width are specified. Subsequently, the lists are created with the function `create_np_array` and the function `do_glitch` is called for all possible permutations of the three lists.

Listing 6.3 contains the `do_glitch` function of the evaluation program. In a first step, the ChipWhisperer is initialized with the three glitch parameters *width*, *offset* and *repeat*. The second code line performs the current test. The ChipWhisperer resets the target and

[3] https://www.keysight.com/en/pdx-x201944-pn-MSOX4034A/
[4] https://wiki.newae.com/Making_Scripts
[5] https://pyvisa.readthedocs.io/en/stable/

```
def create_np_array(start, stop, step):
    return np.arange(start, stop + step, step)

if __name__ == "__main__":
    glitch_offset = create_np_array(Decimal(-49.0),
                                    Decimal(49.0),
                                    Decimal(0.2))
    glitch_width = create_np_array(Decimal(4.0),
                                   Decimal(6.0),
                                   Decimal(0.05))
    glitch_repeat = create_np_array(1, 5, 1)

    for offset in glitch_offset:
        for width in glitch_width:
            for repeat in glitch_repeat:
                do_glitch(offset, width, repeat)
```

Listing 6.3: Evaluation Software – main Function

```
def do_glitch(offset, width, repeat):
    CWBroker.getInstance().set_glitch_parameter(offset, width, repeat)
    self.cw.runScriptClass(CWUserScript)
    response = CWBroker.getInstance().get_response()

    if "1234" in response:
        status = "success"
    elif response.count("hello") > 1:
        status = "reset"
    else:
        status = "normal"

    glitch = Glitch(datetime.datetime.now(), status, response,
                    offset, width, repeat)
    self.db.insert(glitch)
```

Listing 6.4: Evaluation Software – do_glitch Function

waits for the trigger. When the trigger is fired, a glitch is injected into the supply voltage of the target. After the glitch, the ChipWhisperer waits for 250 ms, so that the behavior of the target can be recorded. During the test, we record all ASCII characters transmitted from the ATmega328P target to the ChipWhisperer via the UART_Tx connection. After the test, all received characters are stored as string in the variable `response`. For each test, we distinguish between three possible states. A glitch is successful (*success*) if the string "1234" is included in the response. If the string "hello" is included more than once, the glitch has resulted in a *reset*. If neither the *success* nor the *reset* status apply, we assume that the glitch had no effect on the target. This status is called *normal*. At the end of the function, the result is stored in the database.

For the evaluation, not only the test status is relevant, but also the behavior during

the test. For this reason, we developed a program that activates the single shot mode of the oscilloscope and waits for the appearance of a glitch. Thereafter, the recorded waveforms are extracted from the oscilloscope memory and the values are written to a Comma-separated values (CSV) file. The program is attached in Appendix B.5. However, since the recording and extraction to the PC takes about 10 s, the methodology was not feasible for all test cases. As a result, we had to limit the recording of waveforms to chosen glitches only. In Chapter 7, we describe which glitches are recorded with the oscilloscope and why we chose them.

6.2 Evaluation 1: ATmega Target without Decoupling Capacitor

At the Black Hat Europe conference in 2015, Brett Giller gave a talk on voltage fault injection attacks including the recommendation to remove all of the power decoupling capacitors during voltage fault injection attacks [52]. To allow comparison, we operate the ATmega328P target without decoupling capacitor C_1 (see Section 6.1.1) in our first evaluation.

For the evaluation, we limited the lower and upper bounds of the parameters *width*, *offset* and *repeat* to reasonable ranges. According to Carpi et al. [9], there exists a lower bound for the glitch width. If the glitch with is below this value, the target will not be affected by the glitch resulting in a *normal* test status. In contrast, there exists an upper bound for the glitch width as well. If the glitch width is higher than the upper bound, the target will reset resulting in a *reset* test status. Unfortunately, the parameters *offset* and *repeat* can not be limited in a similar way. For this reason, a lower bound of -49.0% and an upper bound of 49.0% are chosen for the offset. We selected a step width of 0.2%, which corresponds to a step width of 272 ps at a clock rate of 7.37 MHz. In comparison to the clock period, the step width was small enough to achieve good results within the evaluation. For the parameter repeat, we set the lower bound to 1 and the upper bound to 5. The identification of the bounds for the *width* parameter is described after the explanation of the test cases.

Table 6.1 gives an overview over the scheduled test cases. Each row except the table header characterizes one test case. The first column contains a unique identifier for each test case. The second and third columns contain the lower and upper bounds of the *offset* parameter. The fourth column states the step with for the *offset* parameter. The next three columns specify the lower bound, upper bound, and step size for the *width* parameter. Similarly, columns 8-10 specify the *repeat* parameter ranges and step width. The column “GNDA” indicates the voltage level of GNDA in the respective test case. The following column describes if the brownout protection is activated and if so, the threshold is provided. The last column indicates whether a decoupling capacitor is used and what value it has. The table illustrates that 12 test cases are carried out for the first evaluation. For each test case, 100655 tests (glitches) are performed. This results in a total of 1207860 tests (glitches) for the first evaluation.

ID	Offset			Width			Repeat			GNDA	Brownout Protection	Decoupling Capacitor
	From	To	Step	From	To	Step	From	To	Step			
1.1.1	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	0.0	No	No
1.1.2	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-2.0	No	No
1.1.3	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-4.0	No	No
1.1.4	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-6.0	No	No
1.2.1	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	0.0	2.7 V	No
1.2.2	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-2.0	2.7 V	No
1.2.3	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-4.0	2.7 V	No
1.2.4	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-6.0	2.7 V	No
1.3.1	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	0.0	1.8 V	No
1.3.2	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-2.0	1.8 V	No
1.3.3	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-4.0	1.8 V	No
1.3.4	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-6.0	1.8 V	No

Table 6.1: Test Cases for Evaluation 1

```
def create_np_array(start, stop, step):
    return np.arange(start, stop + step, step)

TEST_WIDTH = Decimal(4.0)

glitch_offset = create_np_array(Decimal(-49), Decimal(49), Decimal(0.2))
glitch_repeat = create_np_array(1, 5, 1)

for offset in glitch_offset:
    for repeat in glitch_repeat:
        call_str = "python glitch.py " + offset.to_eng_string() + " " \
                   + TEST_WIDTH.to_eng_string() + " " + str(repeat)
        subprocess.Popen(call_str, shell=True)
```

Listing 6.5: Bound Search Script for *width* Parameter

We already discussed how we identified the bounds and step sizes for the parameters *offset* and *repeat*. In the following, the lower and upper bound search strategy for the *width* parameter is described. As mentioned above, there exists a lower bound for the glitch width. If the glitch width is set to this value or lower, the target will ignore the glitch. This lower bound must be examined for each of the test cases defined in Table 6.1. Listing 6.5 contains the Python script we used to find the bounds of the *width* parameter. The variable TEST_WIDTH stores the *width* to be tested. A list with all possible values is created for both the *offset* and *repeat* parameters. The Python program `glitch.py` (Appendix B.2) is called for all possible permutations of the two lists. It executes a glitch with the given parameters and returns the resulting status. For the *width*, the value stored in variable `TEST_WIDTH` is used. This way, a *width* can be identified where every call of glitch.py barely returns the status *normal*. This process must be repeated for each test case. The final lower bound is the lowest identified value. Similarly, the same process is applied to identify the upper bound by means of the *reset* status. This way, we identified a lower bound of 4.0% and an upper bound of 6.0% for the *width* parameter.

Due to the small distance between the upper and lower bound, we have chosen a step width of 0.05%. At a clock rate of 7.37 MHz, this corresponds to a step size of 68 ps.

6.3 Evaluation 2: ATmega Target with Decoupling Capacitor

Microcontrollers with a higher integration depth often include on-chip decoupling capacitors [53, 54] that can not be removed during a voltage fault injection attack. For this reason, in contrast to the first evaluation, we operated the ATmega328P target with a 100 nF decoupling capacitor in our second evaluation. Due to this intentionally chosen high capacity, the idea of negative voltage fault injection attacks can be tested against a target with a high capacitance.

Similar to the previous evaluation, the lower and upper bounds of the parameters *width*, *offset* and *repeat* must be limited to reasonable values. Since the lower and upper bounds of the *offset* cannot be limited, the lower bound is set to -49.0% and the upper bound is set to 49.0%. Due to the high capacitance, we assumed that the distance between the upper and lower bound of the *width* parameter is much larger in comparison to the first evaluation. We thus increased the step size for the *offset* to 0.5% so that the number of tests can be performed within a reasonable time frame. For the same reason, the *repeat* parameter is limited to the value 1. The lower and upper bounds for the *width* parameter are identified in the same way as described at the first evaluation. We identified a lower bound of 8.0% and an upper bound of 49.0%, which is the maximum value possible at the ChipWhisperer.

Table 6.2 gives an overview over the scheduled test cases. The table is structured the same way as Table 6.1. The table illustrates that 12 test cases are carried out for the second evaluation. For each test case, 16351 tests (glitches) are performed. This results in a total of 196212 tests (glitches) for the second evaluation.

ID	Offset			Width			Repeat			GNDA	Brownout Protection	Decoupling Capacitor
	From	To	Step	From	To	Step	From	To	Step			
2.1.1	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	0.0	No	100 nF
2.1.2	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-2.0	No	100 nF
2.1.3	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-4.0	No	100 nF
2.1.4	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-6.0	No	100 nF
2.2.1	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	0.0	2.7 V	100 nF
2.2.2	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-2.0	2.7 V	100 nF
2.2.3	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-4.0	2.7 V	100 nF
2.2.4	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-6.0	2.7 V	100 nF
2.3.1	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	0.0	1.8 V	100 nF
2.3.2	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-2.0	1.8 V	100 nF
2.3.3	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-4.0	1.8 V	100 nF
2.3.4	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-6.0	1.8 V	100 nF

Table 6.2: Test Cases for Evaluation 2

CHAPTER 7

Results

7.1 Evaluation 1: ATmega Target without Decoupling Capacitor

In this section, we present the results of the first evaluation. The results are divided into three subsections that correspond to the chosen "Brownout Protection" setup. The first subsection contains the results with a deactivated brownout protection while the second and third subsections focus on activated brownout protection instead. The second subsection contains the results for a brownout protection threshold of 2.7 V, and the third subsection presents the results for a brownout protection threshold of 1.8 V. Due to the division, the respective test cases in the subsections differ only in the value used for parameter GNDA.

In addition to the results for the associated test cases, each subsection also contains oscilloscope recordings of two successful related glitches. For the first recording, the successful glitch with the lowest glitch signal *width* is selected. In contrast for the second recording, the successful glitch with the highest glitch signal *width* is chosen.

7.1.1 Deactivated Brownout Protection

Table 7.1 gives an overview of the evaluation results for test cases 1.1.1, 1.1.2, 1.1.3 and 1.1.4. The table is structured similarly to Tables 6.1 and 6.2, but it is extended with three columns. The column "Success" contains the number of successful glitches. In column "Reset" the number of glitches are described that led to a reset of the target. The last column documents the number of glitches that did not affect the target.

In the following, the results for each test case are discussed in detail. To illustrate which values for parameters *width*, *offset* and *repeat* resulted in a successful glitch, a 3D plot has been generated for each test case. In the plot, the parameter *offset* is shown on the

X-axis, the parameter *width* is on the Y-axis and the parameter *repeat* is on the Z-axis. The 3D plot was generated with the program attached in Appendix B.4.

ID	Offset			Width			Repeat			GNDA	Brownout Protection	Decoupling Capacitor	Result		
	From	To	Step	From	To	Step	From	To	Step				Success	Reset	Normal
1.1.1	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	0.0	No	No	500	41824	58331
1.1.2	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-2.0	No	No	27	41952	58676
1.1.3	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-4.0	No	No	179	42371	58105
1.1.4	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-6.0	No	No	88	42627	57940

Table 7.1: Test Cases with Deactivated Brownout Protection and without Decoupling Capacitor

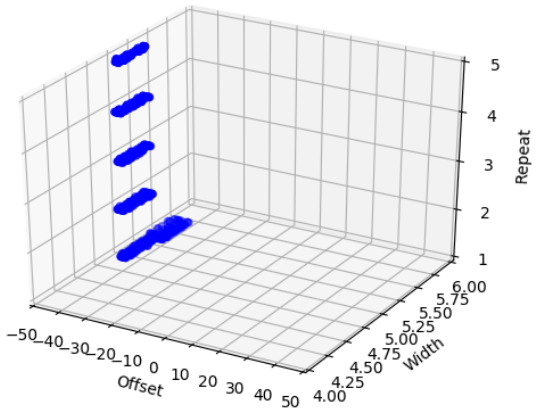

Figure 7.1: Test Case: 1.1.1, Brownout: Not Activated, GNDA: 0.0 V, Success: 500

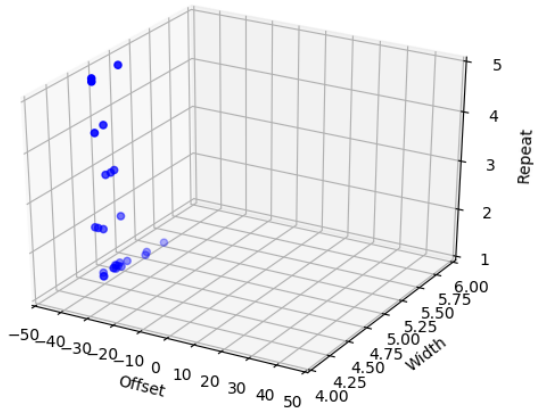

Figure 7.2: Test Case: 1.1.2, Brownout: Not Activated, GNDA: -2.0 V, Success: 27

In test case 1.1.1, 500 successful glitches were recorded. In contrast, 41824 glitches resulted in a reset of the target, and 58331 glitches did not cause an impact. Figure 7.1 shows a 3D plot of this test case. In total, 500 successful glitches are shown in the plot, where each successful glitch is represented by a dot. For a *repeat* of 1, 212 successful glitches were identified. Of those, the minimum *offset* is -47.8% and the maximum *offset* is -40.4% while the *width* is between 4.9% and 5.65%. At a *repeat* of 2 with 112 successful glitches, the *offset* is between -46.6% and -42.4% and the *width* is between 4.9% and 5.25%. In comparison, 75 successful glitches were found at a *repeat* of 3, with an *offset* between -46.2% and -42.8% and a *width* between 4.9% and 5.25%. For a *repeat* of 4 with 63 successful glitches, the minimal *offset* is -46.2% and the maximum *offset* is -43.2% while the *width* is between 4.9% and 5.25%. Finally, for a *repeat* of 5 with 38 successful glitches, the *offset* is between -46.2% and -43.2% and the *width* is between 4.9% and 5.25%.
From the results of the first test case, four observations can be made:

1. The successful glitches are close to each other.
2. The most successful glitches were found at a *repeat* of 1.

3. At a *repeat* of 1, the *width* is between 4.9% and 5.65%. At a higher *repeat*, the *with* is between 4.9% and 5.25%. The valid *with* area thus becomes smaller.

4. At a *repeat* of 1, the offset is between -47.8% and -40.4%. With increasing *repeat*, the successful *offset* area becomes smaller. At a *repeat* of 5, the successful *offset* area is between -46.2% and -43.2% only.

In test case 1.1.2, 27 successful glitches were recorded. In contrast, 41952 glitches resulted in a reset of the target, and 58676 glitches did not cause an impact. Figure 7.2 shows a 3D plot of this test case. In total, 27 successful glitches are shown in the plot, where each successful glitch is represented by a dot. For a *repeat* of 1, 15 successful glitches were identified. Of those, the minimum *offset* is -43.6% and the maximum *offset* is -39.4% while the *width* is between 4.55% and 5.25%. At a *repeat* of 2 with 4 successful glitches, the *offset* is between -44.6% and -41.4% and the *width* is between 4.55% and 4.8%. In comparison, 3 successful glitches were found at a *repeat* of 3, with an *offset* between -43.0% and -42.6% and a *width* between 4.65% and 4.75%. For a *repeat* of 4 with 2 successful glitches, the minimal *offset* is -43.2% and the maximum *offset* is -41.8% while the *width* is between 4.5% and 4.65%. Finally, for a *repeat* of 5 with 3 successful glitches, the *offset* is between -43.8% and -42.2% and the *width* is between 4.5% and 4.85%.
In this test case it can be seen that only a small number of successful glitches were found. The minimum width of 4.5% is significantly smaller than the minimum width of 4.9% in test case 1.1.1. Likewise to test case 1.1.1, successful glitches are close to each other.

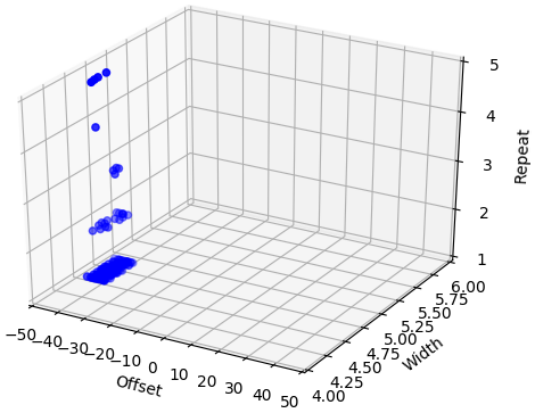

Figure 7.3: Test Case: 1.1.3, Brownout: Not Activated, GNDA: -4.0 V, Success: 179

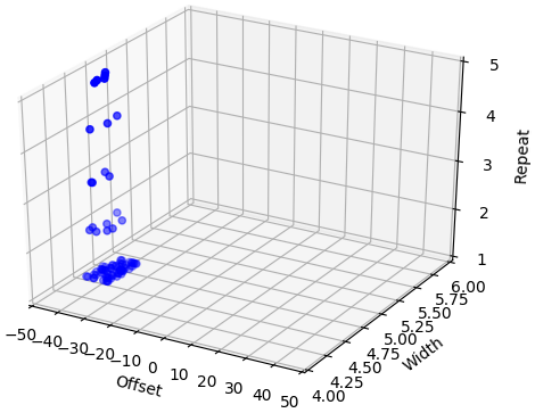

Figure 7.4: Test Case: 1.1.4, Brownout: Not Activated, GNDA: -6.0 V, Success: 88

In test case 1.1.3, 179 successful glitches were recorded. In contrast, 42371 glitches resulted in a reset of the target, and 58105 glitches did not cause an impact. Figure 7.3 shows a 3D plot of this test case. In total, 179 successful glitches are shown in the plot, where each successful glitch is represented by a dot. For a *repeat* of 1, 153 successful glitches were identified. Of those, the minimum *offset* is -45.8% and the maximum *offset* is -38.2% while the *width* is between 4.5% and 4.85%. At a *repeat* of 2 with 17 successful

glitches, the *offset* is between -44.4% and -40.0% and the *width* is between 4.5% and 4.85%. In comparison, 4 successful glitches were found at a *repeat* of 3, with an *offset* between -42.4% and -39.8% and a *width* between 4.7% and 4.8%. For a *repeat* of 4 only 1 successful glitch with an *offset* of -43.4% and a *width* of 4.6% was identified. Finally, for a *repeat* of 5 with 4 successful glitches, the *offset* is between -41.8% and -41.2% and the *width* is between 4.5% and 4.7%.
While the majority of the successful glitches has a *repeat* of 1, successful glitches were identified for each *repeat*. In general, the *width* area is significantly shorter in comparison to the first two test cases. Similar to the previous test cases, successful glitches are close to each other.

In the last test case 1.1.4, 88 successful glitches were recorded. In contrast, 42627 glitches resulted in a reset of the target, and 57940 glitches did not cause an impact. Figure 7.4 shows a 3D plot of this test case. In total, 88 successful glitches are shown in the plot, where each successful glitch is represented by a dot. For a *repeat* of 1, 68 successful glitches were identified. Of those, the minimum *offset* is -46.0% and the maximum *offset* is -37.0% while the *width* is between 4.5% and 4.85%. At a *repeat* of 2 with 8 successful glitches, the *offset* is between -44.4% and -38.2% and the *width* is between 4.5% and 4.85%. In comparison, 4 successful glitches were found at a *repeat* of 3, with an *offset* between -43.6% and -40.4% and a *width* between 4.5% and 4.7%. For a *repeat* of 4 with 3 successful glitches, the minimal *offset* is -44.0% and the maximum *offset* is -42.0% while the *width* is between 4.55% and 4.85%. Finally, for a *repeat* of 5 with 5 successful glitches, the *offset* is between -42.2% and -39.4% and the *width* is between 4.5% and 4.7%.
The last test case also fits into the pattern of the other test cases. The most successful glitches are at a *repeat* of 1. Again, the successful glitches are close together. The values for *offset* and *width* that result in a successful glitch are very similar to the values of test case 1.1.3.

For each brownout detection configuration, we conducted oscilloscope measurements. Figure 7.5 shows the successful glitch with the lowest *width*. The green waveform ("Trigger") shows the trigger signal, the blue waveform ("Glitch Signal") shows the glitch signal and the red waveform ("Glitch") shows the power supply voltage of the target. The time is shown on the X-Axis while the voltage is shown on the Y-axis. The glitch was identified in test case 1.1.3. The *offset* is -41.2%, the *width* is 4.5% and the *repeat* is 5. For the glitch a $V_{G_{Low}}$ of 1.2 V, a glitch signal width (GS_{Width}) of 7 ns and a glitch width (G_{Width}) of 38 ns can be measured.

In comparison, Figure 7.6 shows the successful glitch with the highest *width*. The glitch was identified in test case 1.1.1. The *offset* is -40.8%, the *width* is 5.65% and the *repeat* is 1. For the glitch a $V_{G_{Low}}$ of 1.2 V, a glitch signal width (GS_{Width}) of 8 ns and a glitch width (G_{Width}) of 40 ns can be measured.

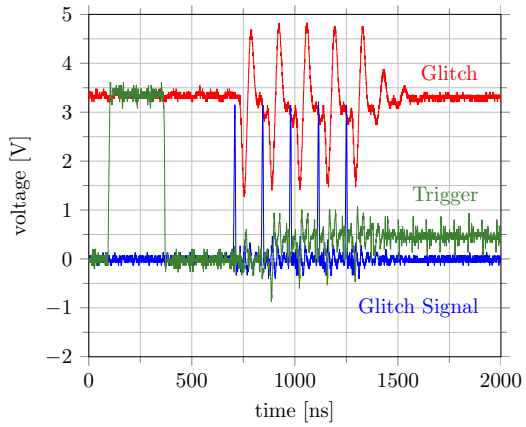

Figure 7.5: Brownout: Not Activated, GNDA: -4.0 V, Offset: -41.2, Width: 4.5, Repeat: 5, $V_{G_{Low}}$: 1.2 V, GS_{Width}: 7 ns, G_{Width}: 38 ns

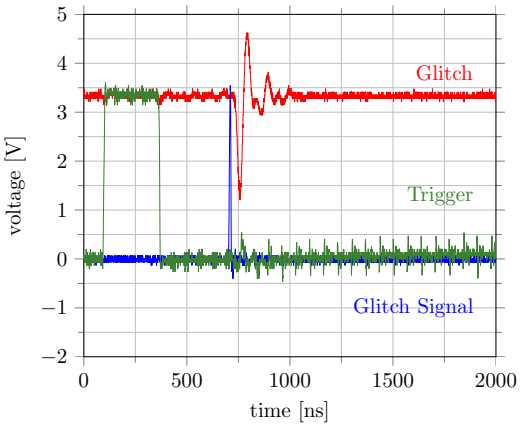

Figure 7.6: Brownout: Not Activated, GNDA: 0.0 V, Offset: -40.8, Width: 5.65, Repeat: 1, $V_{G_{Low}}$: 1.2 V, GS_{Width}: 8 ns, G_{Width}: 40 ns

7.1.2 Brownout: 2.7 Volt

Table 7.2 gives an overview of the evaluation results for test cases 1.2.1, 1.2.2, 1.2.3 and 1.2.4. The table is structured in the same way as Table 7.1. In the following, the results for each test case are discussed in detail. For this purpose, we conducted a 3D plot for each test case as well.

ID	Offset			Width			Repeat			GNDA	Brownout Protection	Decoupling Capacitor	Result		
	From	To	Step	From	To	Step	From	To	Step				Success	Reset	Normal
1.2.1	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	0.0	2.7 V	No	1409	40763	58483
1.2.2	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-2.0	2.7 V	No	941	41134	58580
1.2.3	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-4.0	2.7 V	No	468	41486	58701
1.2.4	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-6.0	2.7 V	No	103	41897	58655

Table 7.2: Test Cases with 2.7 V Brownout Protection and without Decoupling Capacitor

In test case 1.2.1, 1409 successful glitches were recorded. In contrast, 40763 glitches resulted in a reset of the target, and 58483 glitches did not cause an impact. Figure 7.7 shows a 3D plot of this test case. In total, 1409 successful glitches are shown in the plot, where each successful glitch is represented by a dot. For a *repeat* of 1, 449 successful glitches were identified. Of those, the minimum *offset* is -47.6% and the maximum *offset* is -38.6% while the *width* is between 4.5% and 5.65%. At a *repeat* of 2 with 318 successful glitches, the *offset* is between -47.4% and -40.2% and the *width* is between 4.5% and 5.25%. In comparison, 279 successful glitches were found at a *repeat* of 3, with an *offset* between -47.0% and -40.4% and a *width* between 4.5% and 5.25%. For a *repeat* of 4 with 208 successful glitches, the minimal *offset* is -46.8% and the maximum *offset* is -40.2% while the *width* is between 4.5% and 5.25%. Finally, for a *repeat* of 5 with 155 successful glitches, the *offset* is between -46.6% and -40.6% and the *width* is between 4.5% and 5.25%.

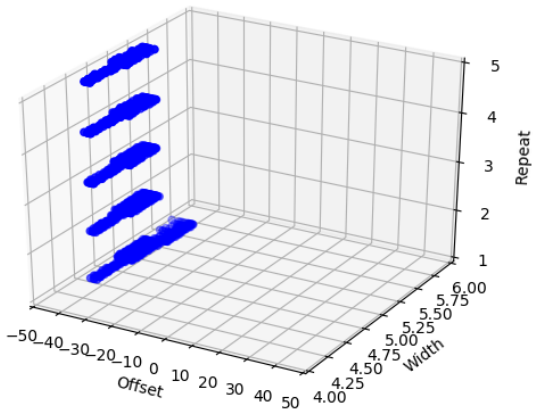

Figure 7.7: Test Case: 1.2.1, Brownout: 2.7 V, GNDA: 0.0 V, Success: 1409

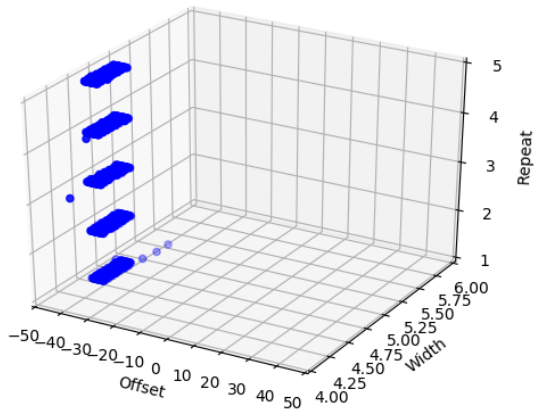

Figure 7.8: Test Case: 1.2.2, Brownout: 2.7 V, GNDA: -2.0 V, Success: 941

Despite the activated brownout protection, in this test case significantly more successful glitches were found in comparison to test case 1.1.1. Again, the successful glitches are close to each other and at a *repeat* of 1, the most successful glitches can be found. For a *repeat* of 1, the *width* is between 4.5% and 5.65%. At a higher *repeat*, the *width* is between 4.5% and 5.25%. The upper bound of the *width* is the same as for test case 1.1.1. In contrast, the lower bound is 4.5% in this test case and 4.9% in test case 1.1.1. In can thus be seen that a smaller *width* is encouraged by the activated brownout protection with a threshold of 2.7 V.

In test case 1.2.2, 941 successful glitches were recorded. In contrast, 41134 glitches resulted in a reset of the target, and 58580 glitches did not cause an impact. Figure 7.8 shows a 3D plot of this test case. In total, 941 successful glitches are shown in the plot, where each successful glitch is represented by a dot. For a *repeat* of 1, 194 successful glitches were identified. Of those, the minimum *offset* is -46.4% and the maximum *offset* is -38.0% while the *width* is between 4.5% and 5.25%. At a *repeat* of 2 with 199 successful glitches, the *offset* is between -45.4% and -39.8% and the *width* is between 4.5% and 4.85%. In comparison, 184 successful glitches were found at a *repeat* of 3, with an *offset* between -45.6% and -39.8% and a *width* between 4.2% and 4.85%. For a *repeat* of 4 with 187 successful glitches, the minimal *offset* is -45.4% and the maximum *offset* is -39.6% while the *width* is between 4.4% and 4.85%. Finally, for a *repeat* of 5 with 177 successful glitches, the *offset* is between -45.4% and -39.4% and the *width* is between 4.5% and 4.85%. In this test case, it is the first time that more successful glitches were found at a *repeat* of 2 than for a *repeat* of 1. However, the number of successful glitches over the *repeats* is evenly distributed. Apart from the four successful glitches that stand out, the values for the *offset* and the *width* are uniform. The successful glitches are close to each other once again.

In test case 1.2.3, 468 successful glitches were recorded. In contrast, 41486 glitches resulted in a reset of the target, and 58701 glitches did not cause an impact. Figure 7.9

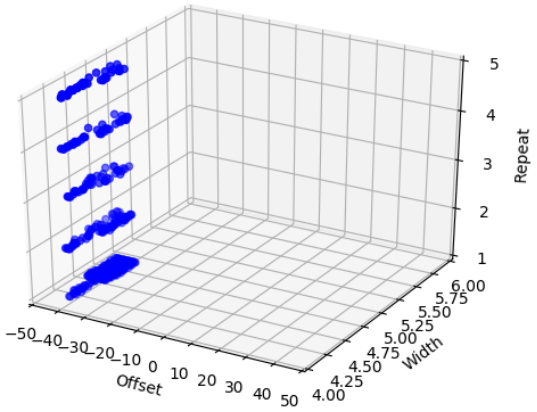

Figure 7.9: Test Case: 1.2.3, Brownout: 2.7 V, GNDA: -4.0 V, Success: 468

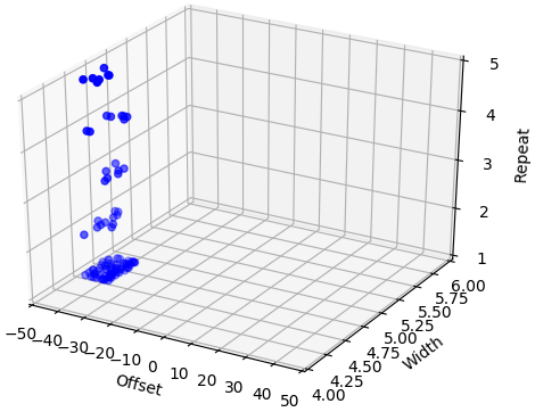

Figure 7.10: Test Case: 1.2.4, Brownout: 2.7 V, GNDA: -6.0 V, Success: 103

shows a 3D plot of this test case. In total, 468 successful glitches are shown in the plot, where each successful glitch is represented by a dot. For a *repeat* of 1, 275 successful glitches were identified. Of those, the minimum *offset* is -46.0% and the maximum *offset* is -37.2% while the *width* is between 4.15% and 4.85%. At a *repeat* of 2 with 82 successful glitches, the *offset* is between -44.6% and -37.8% and the *width* is between 4.15% and x%. In comparison, 43 successful glitches were found at a *repeat* of 3, with an *offset* between -45.0% and -37.8% and a *width* between 4.2% and 4.85%. For a *repeat* of 4 with 34 successful glitches, the minimal *offset* is -44.6% and the maximum *offset* is -38.0% while the *width* is between 4.15% and 4.85%. Finally, for a *repeat* of 5 with 34 successful glitches, the *offset* is between -44.0% and -38.2% and the *width* is between 4.15% and 4.85%.
Again, the successful glitches are close to each other and the most successful glitches are at a *repeat* of 1. It is noticeable, that the *width* of 4.15% is very low. In general, the *repeat* has no effect on the *width*, since the upper and lower bounds of the *width* are nearly the same.

In test case 1.2.4, there were 103 successful glitches. In contrast, 41897 glitches resulted in a reset of the target and 58655 glitches did not cause an impact. Figure 7.10 shows a 3D plot of this test case. In total, 103 successful glitches are shown in the plot, where each successful glitch is represented by a dot. For a *repeat* of 1, 70 successful glitches were identified. Of those, the minimum *offset* is -47.4% and the maximum *offset* is -37.2% while the *width* is between 4.5% and 4.85%. At a *repeat* of 2 with 11 successful glitches, the *offset* is between -45.4% and -38.6% and the *width* is between 4.4% and 4.85%. In comparison, 7 successful glitches were found at a *repeat* of 3, with an *offset* between -44.2% and -38.6% and a *width* between 4.55% and 4.85%. For a *repeat* of 4 with 8 successful glitches, the minimal *offset* is -45.4% and the maximum *offset* is -38.8% while the *width* is between 4.5% and 4.85%. Finally, for a *repeat* of 5 with 7 successful glitches, the *offset* is between -44.2% and -39.0% and the *width* is between 4.5% and 4.75%.

Again, the most successful glitches are at a *repeat* of 1. For higher *repeat* values only a few successful glitches were found.

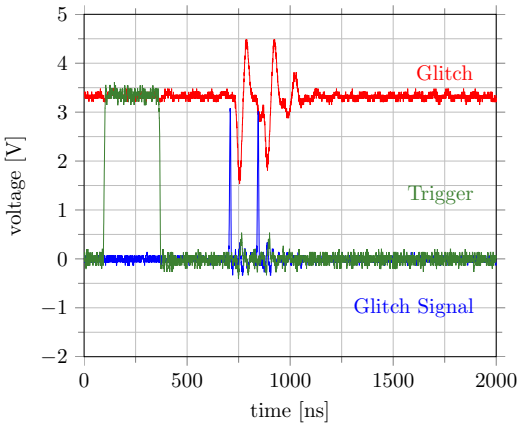

Figure 7.11: Brownout: 2.7 V, GNDA: -4.0 V, Offset: -40.6, Width: 4.15, Repeat: 2, $V_{G_{Low}}$: 1.75 V, GS_{Width}: 6 ns, G_{Width}: 38 ns

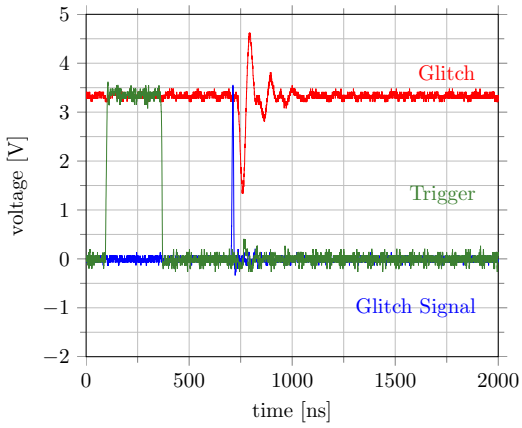

Figure 7.12: Brownout: 2.7 V, GNDA: 0.0 V, Offset: -39.0, Width: 5.65, Repeat: 1, $V_{G_{Low}}$: 1.28 V, GS_{Width}: 8 ns, G_{Width}: 44 ns

The oscilloscope measurement in Figure 7.11 shows the successful glitch with the lowest *width* for an activated brownout protection with a brownout protection threshold of 2.7 V. The glitch was identified in test case 1.2.3. The *offset* is -40.6%, the *width* is 4.15% and the *repeat* is 2. For the glitch, a $V_{G_{Low}}$ of 1.75 V, a glitch signal width (GS_{Width}) of 6 ns and a glitch width (G_{Width}) of 38 ns can be measured.

Figure 7.12 shows the successful glitch with the highest *width* for an activated brownout protection with a brownout protection threshold of 2.7 V. The glitch was identified in test case 1.2.1. The *offset* is -39.0%, the *width* is 5.65% and the *repeat* is 1. For the glitch a $V_{G_{Low}}$ of 1.28 V, a glitch signal width (GS_{Width}) of 8 ns and a glitch width (G_{Width}) of 44 ns can be measured.

7.1.3 Brownout: 1.8 Volt

Table 7.3 gives an overview of the evaluation results for test cases 1.3.1, 1.3.2, 1.3.3 and 1.3.4. The table is structured in the same way as Table 7.1. As can be seen from the table, no successful glitches were identified for the test cases with an activated brownout protection and a brownout protection threshold of 1.8 V. Therefore, we cannot present more results for this test case.

ID	Offset			Width			Repeat			GNDA	Brownout Protection	Decoupling Capacitor	Result		
	From	To	Step	From	To	Step	From	To	Step				Success	Reset	Normal
1.3.1	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	0.0	1.8 V	No	0	41963	58692
1.3.2	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-2.0	1.8 V	No	0	42187	58468
1.3.3	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-4.0	1.8 V	No	0	42471	58184
1.3.4	-49.0	49.0	0.2	4.0	6.0	0.05	1	5	1	-6.0	1.8 V	No	0	43322	57333

Table 7.3: Test Cases with 1.8 V Brownout Protection and without Decoupling Capacitor

7.2 Evaluation 2: ATmega Target with Decoupling Capacitor

In this section, we present the results of the second evaluation. The results are divided into three subsections that correspond to the chosen "Brownout Protection" setup. The first subsection contains the results with a deactivated brownout protection while the second and third subsections focus on activated brownout protection instead. The second subsection contains the results for a brownout protection threshold of 2.7 V, and the third subsection presents the results for a brownout protection threshold of 1.8 V. Due to the division, the respective test cases in the subsections differ only in the value used for parameter GNDA.

In addition to the results for the associated test cases, each subsection also contains oscilloscope recordings of two successful related glitches. For the first recording the successful glitch with the lowest glitch signal *width* is selected. In contrast for the second recording, the successful glitch with the highest glitch signal *width* is chosen.

7.2.1 Deactivated Brownout Protection

Table 7.4 gives an overview about the evaluation results for test cases 2.1.1, 2.1.2, 2.1.3 and 2.1.4. The table is structured in the same way as Table 7.1. In the following, the results for each test case are discussed in detail. Similar to the first evaluation, we conducted plots to present the results. Since the *repeat* parameter is fixed to value 1 for the second evaluation, this time we use a 2D plot instead. In the plot, the parameter *offset* is shown on the X-axis and the parameter *width* is shown on the Y-axis. In contrast to the first evaluation, the plot not only contains successful glitches, but is also contains glitches that led to a reset of the target, and glitches that did not affect the target at all. Each glitch is illustrated as dot: A successful glitch is shown as black dot, a glitch that led to a reset is shown as gray dot, and a glitch that did not affect the target is shown as white dot. We created the 2D plot with the program attached in Appendix B.3.

ID	Offset			Width			Repeat			GNDA	Brownout Protection	Decoupling Capacitor	Result		
	From	To	Step	From	To	Step	From	To	Step				Success	Reset	Normal
2.1.1	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	0.0	No	100 nF	0	2	16349
2.1.2	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-2.0	No	100 nF	496	2088	13767
2.1.3	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-4.0	No	100 nF	406	3319	12629
2.1.4	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-6.0	No	100 nF	271	4004	12076

Table 7.4: Test Cases with Deactivated Brownout Protection and with 100 nF Decoupling Capacitor

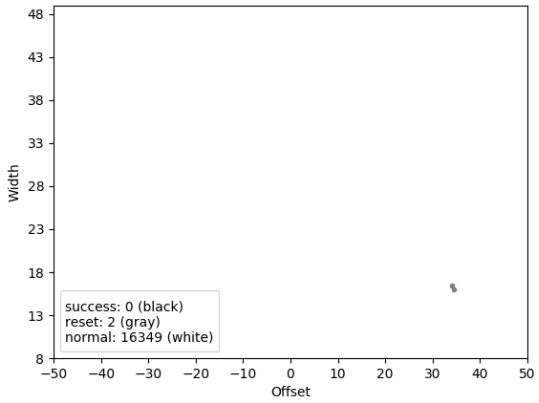

Figure 7.13: Test Case: 2.1.1, Brownout: Not Activated, GNDA: 0.0 V

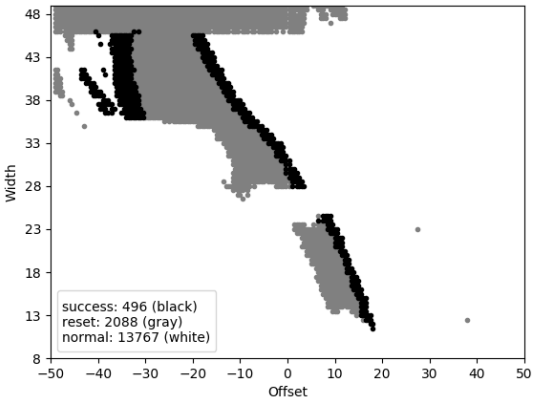

Figure 7.14: Test Case: 2.1.2, Brownout: Not Activated, GNDA: -2.0 V

For test case 2.1.1, 0 successful glitches, 2 glitches resulting in a reset of the target, and 16349 glitches without an impact were identified. Figure 7.13 shows a 2D plot of this test case. Due to the high capacitance of the decoupling capacitor, the power supply of the target could not be pulled down far enough with a GNDA of 0.0 V. For that reason, no successful glitch and only two resets were identified.

For test case 2.1.1, 496 successful glitches, 2088 glitches resulting in a reset of the target, and 13767 glitches without an impact were identified. Figure 7.14 shows a 2D plot for this test case. The minimal *offset* of a successful glitch is -43.5% and the maximum *offset* is 18.0%. In contrast, the minimal *width* of a successful glitch is 11.5% and the maximum *width* is 46.0%. As can be seen from the plot, the successful glitches are more distributed in comparison to the first evaluation. However, they are still distributed in a clear pattern. A significant number of successful glitches can be found in the upper left corner including glitches with with a negative *offset* and a large *width*. The second part of successful glitches is visible in the form of a line from bottom right to top left. The absence of glitches in the middle of the plot is striking and the reasons are unclear at this point.

For test case 2.1.3, 406 successful glitches, 3319 glitches resulting in a reset of the target, and 12629 glitches without an impact were identified. Figure 7.15 shows a 2D plot for this test case. The minimal *offset* for a successful glitch is -49.0% and the maximum *offset* is 22.0%. In contrast, the minimal *width* for a successful glitch is 8.5% and the maximum *width* is 42.0%. The plot is similar to the plot from test case 2.1.2. However, it is visible that the required *width* for successful glitches is lower. This is the reason why the plot looks as if the previous plot had just been moved downwards. Another anomaly is the horizontal cut on the left side of the plot at a *width* of 35.0%.

For test case 2.1.4, 271 successful glitches, 4004 glitches resulting in a reset of the target, and 12076 glitches without an impact were identified. Figure 7.16 shows a 2D plot for this test case. The minimal *offset* for a successful glitch is -49.0% and the maximum

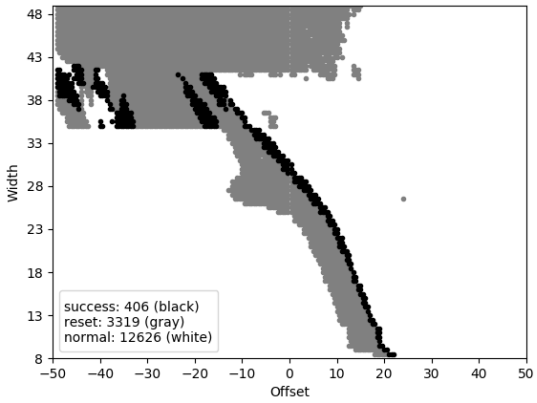

Figure 7.15: Test Case: 2.1.3, Brownout: Not Activated, GNDA: -4.0 V

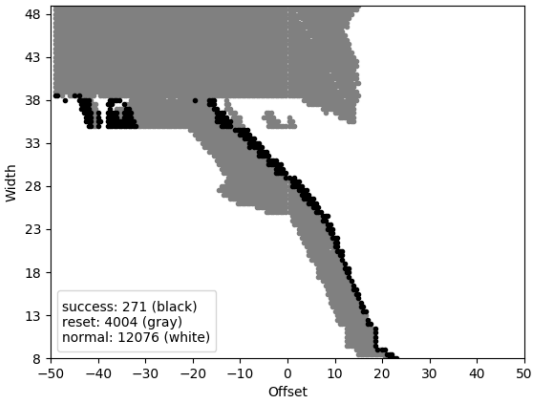

Figure 7.16: Test Case: 2.1.4, Brownout: Not Activated, GNDA: -6.0 V

offset is 23.0%. In contrast, the minimal *width* for a successful glitch is 8.0% and the maximum *width* is 38.5%. The plot is similar to the plot from test case 2.1.3. Once again, it is visible that the required *width* for successful glitches is lower. The plot looks like the previous plot had been moved downwards even more. Also, there is a horizontal cut at a *width* of 35.0%.

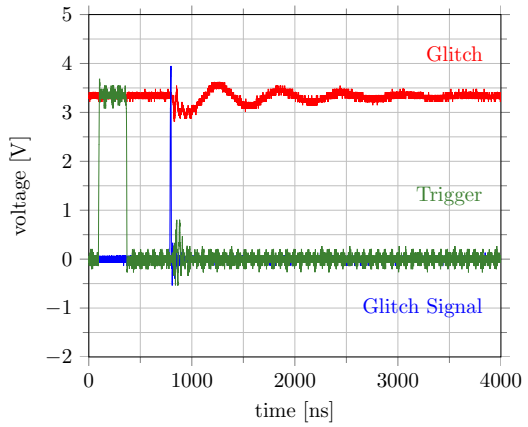

Figure 7.17: Brownout: Not Activated, GNDA: -6.0 V, Offset: 21.5, Width: 8.0, Repeat: 1, $V_{G_{Low}}$: 2.75 V, GS_{Width}: 11 ns, G_{Width}: 305 ns

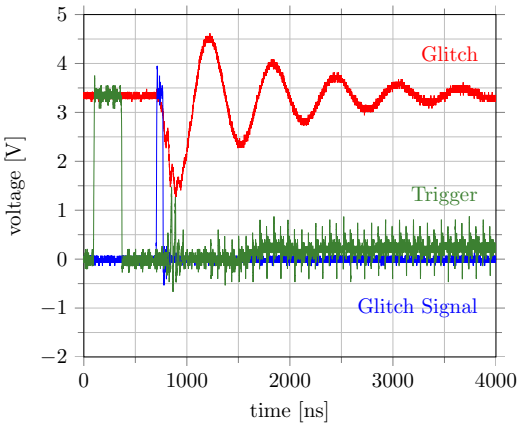

Figure 7.18: Brownout: Not Activated, GNDA: -2.0 V, Offset: 40.5, Width: 46.0, Repeat: 1, $V_{G_{Low}}$: 1.35 V, GS_{Width}: 62 ns, G_{Width}: 350 ns

The oscilloscope measurement in Figure 7.17 shows the successful glitch with the lowest *width* for a deactivated brownout protection. The glitch was identified in test case 2.1.4. The *offset* is 21.5% and the *width* is 8.0%. For the glitch, a $V_{G_{Low}}$ of 2.75 V, a glitch signal width (GS_{Width}) of 11 ns and a glitch width (G_{Width}) of 305 ns can be measured.

Figure 7.18 shows the successful glitch with the highest *width* for a deactivated brownout protection. The glitch was identified in test case 2.1.2. The *offset* is 40.5% and the *width* is 46.0%. For the glitch a $V_{G_{Low}}$ of 1.35 V, a glitch signal width (GS_{Width}) of 62 ns and a glitch width (G_{Width}) of 350 ns can be measured.

7.2.2 Brownout: 2.7 Volt

ID	Offset			Width			Repeat			GNDA	Brownout Protection	Decoupling Capacitor	Result		
	From	To	Step	From	To	Step	From	To	Step				Success	Reset	Normal
2.2.1	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	0.0	2.7 V	100 nF	0	3	16348
2.2.2	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-2.0	2.7 V	100 nF	349	1175	14827
2.2.3	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-4.0	2.7 V	100 nF	542	2957	12852
2.2.4	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-6.0	2.7 V	100 nF	257	3956	12138

Table 7.5: Test Cases with 2.7 V Brownout Protection and with 100 nF Decoupling Capacitor

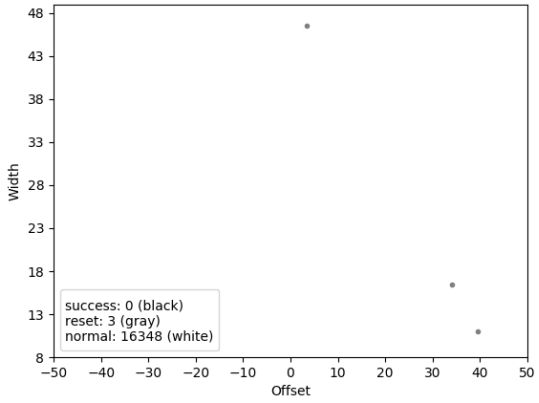

Figure 7.19: Test Case: 2.2.1, Brownout: 2.7 V, GNDA: 0.0 V

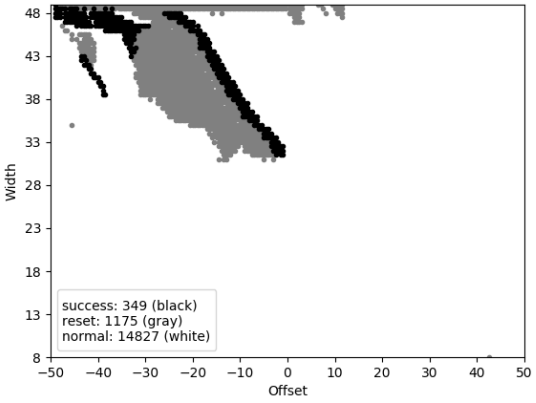

Figure 7.20: Test Case: 2.2.2, Brownout: 2.7 V, GNDA: -2.0 V

For test case 2.2.1, 0 successful glitches, 3 glitches resulting in a reset of the target, and 16348 glitches without an impact were identified. Figure 7.19 shows a 2D plot for this test case. Due to the high capacitance of the decoupling capacitor, the power supply of the target cannot be pulled down far enough with a GNDA of 0.0 V. For that reason, no successful glitch and only three resets are identified.

For test case 2.2.2, 349 successful glitches, 1175 glitches resulting in a reset of the target, and 14827 glitches without an impact were identified. Figure 7.20 shows a 2D plot for this test case. The minimal *offset* for a successful glitch is -49.0% and the maximum *offset* is -1.0%. In contrast, the minimal *width* for a successful glitch is 31.5% and the maximum *width* is 49.0%. Again, there are several groups of successful glitches in this test case. The right group has the shape of a line and is similar to test case 2.1.2. However, the line only goes down to a *width* of 30.0%. The successful glitches in the upper left corner are distributed over an offset of -49.0% to -28.5%. This results in a significantly different plot in comparison to test case 2.1.2.

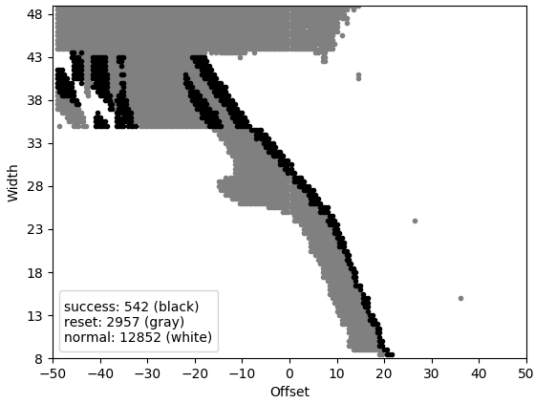

Figure 7.21: Test Case: 2.2.3, Brownout: 2.7 V, GNDA: -4.0 V

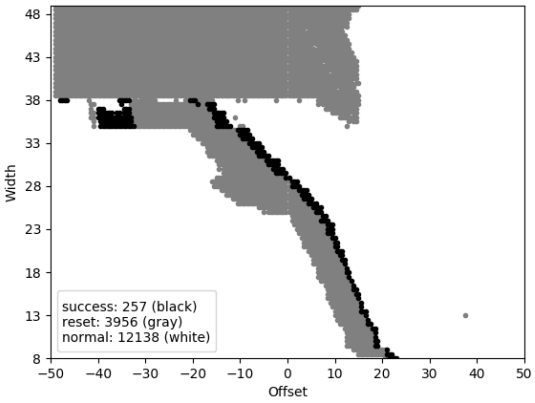

Figure 7.22: Test Case: 2.2.4, Brownout: 2.7 V, GNDA: -6.0 V

For test case 2.2.3, 542 successful glitches, 2957 glitches resulting in a reset of the target, and 12852 glitches without an impact were identified. Figure 7.21 shows a 2D plot for this test case. The minimal *offset* for a successful glitch is -49.0% and the maximum *offset* is 21.5%. In contrast, the minimal *width* for a successful glitch is 8.5% and the maximum *width* is 43.5%. For this plot, large differences compared to the plot for test case 2.2.2 can be seen. The maximum *width* is lower than in the previous plot. The successful glitches on the right have the shape of a line and are continuous to the bottom of the plot. As visible in the results for the deactivated brownout protection, there is a horizontal cut on the left side of the plot at a *width* of 35.0%.

For test case 2.2.4, 257 successful glitches, 3956 glitches resulting in a reset of the target, and 12138 glitches without an impact were identified. Figure 7.22 shows a 2D plot for this test case. The minimal *offset* for a successful glitch is -48.0% and the maximum *offset* is 23.0%. In contrast, the minimal *width* for a successful glitch is 8.0% and the maximum *width* is 38.0%. Compared to the plot of test case 2.2.3, the required *width* for successful glitches is lower. Once again, there is a cut at a *width* of 35.0%.

The oscilloscope measurement in Figure 7.23 shows the successful glitch with the lowest *width* for an activated brownout protection with a brownout protection threshold of 2.7 V. The glitch was identified in test case 2.2.4. The *offset* is 23.0% and the *width* is 8.0%. For the glitch a $V_{G_{Low}}$ of 2.75 V, a glitch signal width (GS_{Width}) of 11 ns and a glitch width (G_{Width}) of 305 ns can be measured.

Figure 7.24 shows the successful glitch with the highest *width* for an activated brownout protection with a brownout protection threshold of 2.7 V. The glitch was identified in test case 2.2.2. The *offset* is -49.0% and the *width* is 49.0%. For the glitch, a $V_{G_{Low}}$ of 0.95 V, a glitch signal width (GS_{Width}) of 102 ns and a glitch width (G_{Width}) of 370 ns can be measured.

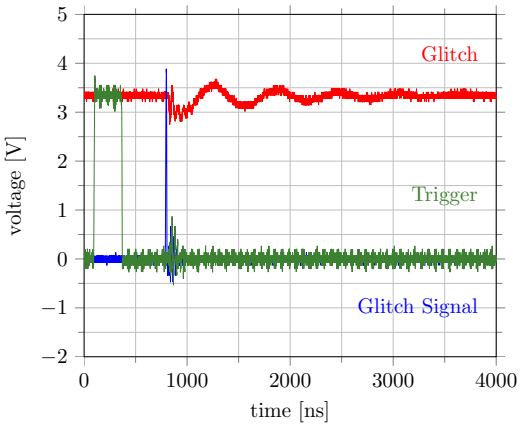

Figure 7.23: Brownout: 2.7 V, GNDA: -6.0 V, Offset: 23.0, Width: 8.0, Repeat: 1, $V_{G_{Low}}$: 2.75 V, GS_{Width}: 11 ns, G_{Width}: 305 ns

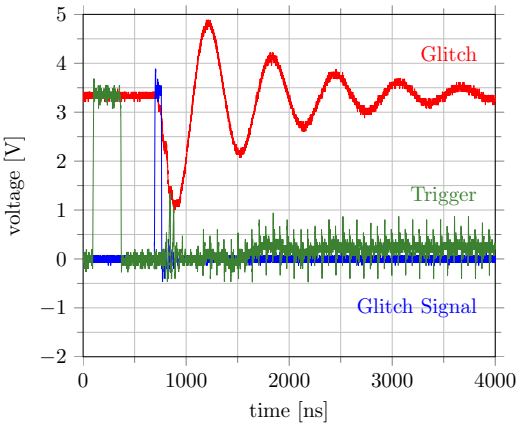

Figure 7.24: Brownout: 2.7 V, GNDA: -2.0 V, Offset: -49.0, Width: 49.0, Repeat: 1, $V_{G_{Low}}$: 0.95 V, GS_{Width}: 102 ns, G_{Width}: 370 ns

7.2.3 Brownout: 1.8 Volt

ID	Offset			Width			Repeat			GNDA	Brownout Protection	Decoupling Capacitor	Result		
	From	To	Step	From	To	Step	From	To	Step				Success	Reset	Normal
2.3.1	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	0.0	1.8 V	100 nF	0	0	16351
2.3.2	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-2.0	1.8 V	100 nF	415	1564	14372
2.3.3	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-4.0	1.8 V	100 nF	558	2775	13018
2.3.4	-49.0	49.0	0.5	8.0	49.0	0.5	1	1	0	-6.0	1.8 V	100 nF	302	3846	12203

Table 7.6: Test Cases with 1.8 V Brownout Protection and with 100 nF Decoupling Capacitor

For test case 2.3.1, 0 successful glitches, 0 glitches resulting in a reset of the target, and 16351 glitches without an impact were identified. Figure 7.25 shows a 2D plot of this test case. Due to the high capacitance of the decoupling capacitor, the power supply of the target cannot pulled down far enough with a GNDA of 0.0 V. For that reason, no successful glitch and no resets are identified.

For test case 2.3.2, 415 successful glitches, 1564 glitches resulting in a reset of the target, and 14372 glitches without an impact were identified. Figure 7.26 shows a 2D plot of this test case. The minimal *offset* for a successful glitch is -49.0% and the maximum *offset* is 0.5%. In contrast, the minimal *width* for a successful glitch is 31.0% and the maximum *width* is 46.5%. Again, there are several groups of successful glitches in this test case. The right group in the shape of a line is similar to the line of test case 2.2.2. The second group is again in the upper left area.

For test case 2.3.3, 558 successful glitches, 2775 glitches resulting in a reset of the target, and 13018 glitches without an impact were identified. Figure 7.27 shows a 2D plot of this test case. The minimal *offset* for a successful glitch is -49.0% and the maximum *offset*

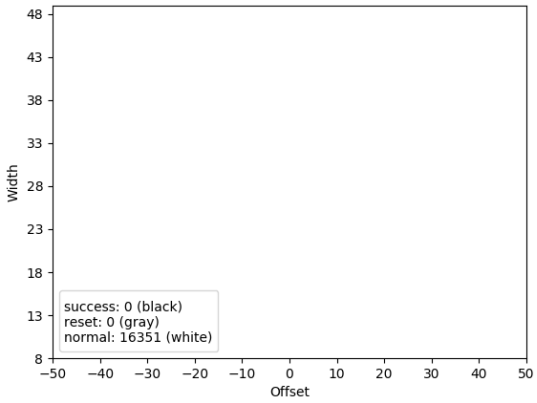

Figure 7.25: Test Case: 2.3.1, Brownout: 1.8 V, GNDA: 0.0 V

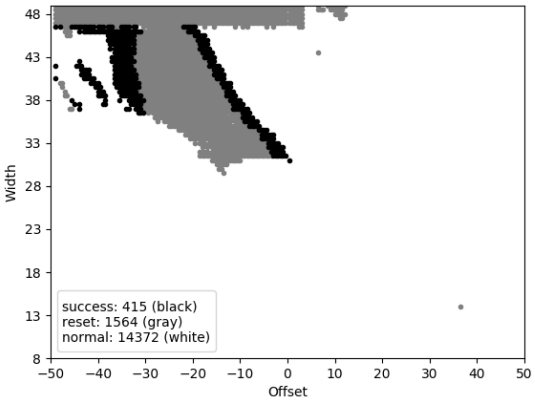

Figure 7.26: Test Case: 2.3.2, Brownout: 1.8 V, GNDA: -2.0 V

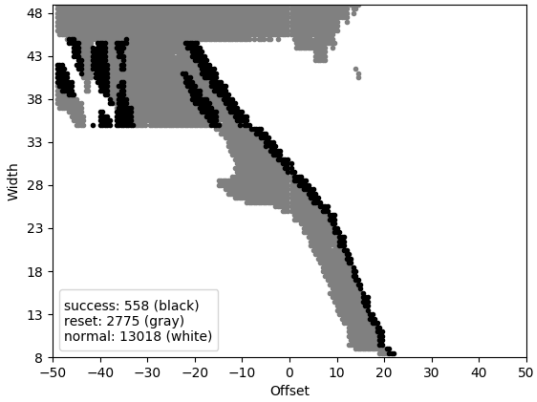

Figure 7.27: Test Case: 2.3.3, Brownout: 1.8 V, GNDA: -4.0 V

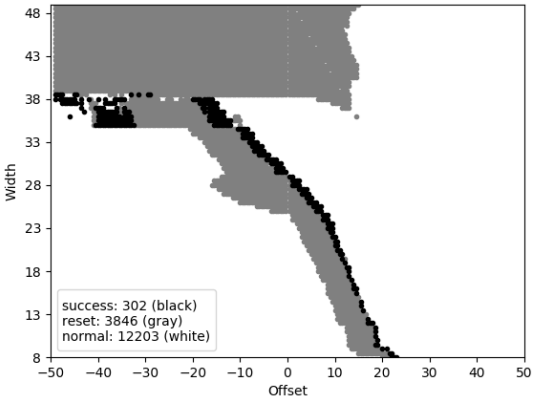

Figure 7.28: Test Case: 2.3.4, Brownout: 1.8 V, GNDA: -6.0 V

for a successful glitch is 22.0%. In contrast, the minimal *width* for a successful glitch is 8.5% and the maximum *width* is 45.0%. The maximum *width* is lower in comparison to the last plot. Furthermore, the plot is similar to the plot of test case 2.2.3. Once again, there is a horizontal cut at a *width* of 35.0%.

For test case 2.3.4, 302 successful glitches, 3846 glitches resulting in a reset of the target, and 12203 glitches without an impact were identified. Figure 7.28 shows a 2D plot of this test case. The minimal *offset* for a successful glitch is -49.0% and the maximum *offset* is 23.0%. In contrast, the minimal *width* for a successful glitch is 8.0% and the maximum *width* is 38.5%. Comparable to the previous plot, there is the horizontal cut at a *width* of 35.0%. In addition, the maximum *width* is lower than in the last plot.

The oscilloscope measurement in Figure 7.29 shows the successful glitch with the lowest *width* for an activated brownout protection with a brownout protection threshold of 1.8 V.

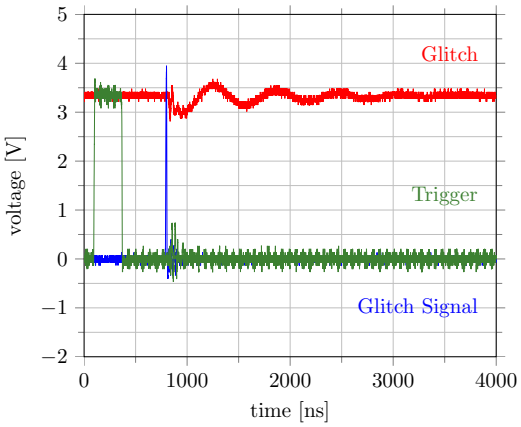

Figure 7.29: Brownout: 1.8 V, GNDA: -6.0 V, Offset: 23.0, Width: 8.0, Repeat: 1, $V_{G_{Low}}$: 2.75 V, GS_{Width}: 11 ns, G_{Width}: 305 ns

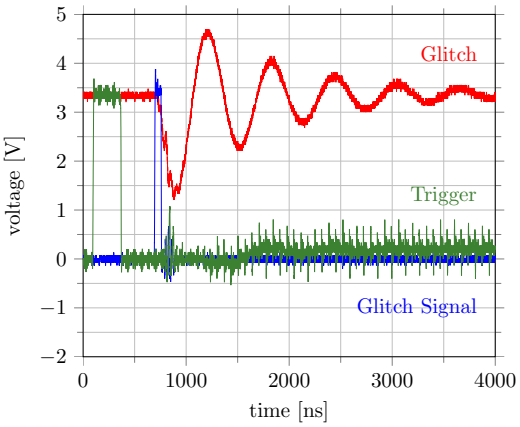

Figure 7.30: Brownout: 1.8 V, GNDA: -2.0 V, Offset: -49.0, Width: 46.5, Repeat: 1, $V_{G_{Low}}$: 1.15 V, GS_{Width}: 64 ns, G_{Width}: 355 ns

The glitch was identified in test case 2.3.4. The *offset* is 23.0% and the *width* is 8.0%. For the glitch a $V_{G_{Low}}$ of 2.75 V, a glitch signal width (GS_{Width}) of 11 ns and a glitch width (G_{Width}) of 305 ns can be measured.

Figure 7.30 shows the successful glitch with the highest *width* for an activated brownout protection with a brownout protection threshold of 1.8 V. The glitch was identified in test case 2.3.2. The *offset* is -49.0% and the *width* is 46.5%. For the glitch a $V_{G_{Low}}$ of 1.15 V, a glitch signal width (GS_{Width}) of 64 ns and a glitch width (G_{Width}) of 355 ns can be measured.

CHAPTER 8

Summary and Conclusion

After a short introduction in Chapter 1 and a description of the fundamentals of microcontrollers in Chapter 2, we discussed the basics of conventional voltage fault injection attacks in Chapter 3. In Chapter 4, we introduced the idea of negative voltage fault injection attacks and defined the basic terms for a better understanding. Furthermore, we showed the limitations of conventional voltage volt fault injection attacks through theoretical considerations and SPICE simulations. In Chapter 5, we defined the requirements for a prototype and discussed several design approaches. For the most promising design approach, we implemented and tested the final prototype. To evaluate our hypothesis on the basis of the implemented prototype, we described the evaluation methodology and the test cases in Chapter 6. Finally in Chapter 7, we discussed the results of the test cases in detail.

The results of the first evaluation, where we removed the decoupling capacitor from the target, showed that the *width* parameter can be minimized by negative voltage fault injection attacks. For all tested configurations, the minimum *width* to find a successful glitch was significantly lower with a negative GNDA in comparison to conventional voltage fault injection attacks that merely pull to GNDA of 0.0 V. Even a low negative GNDA voltage of -2.0 V has significantly shortened the minimum *width* required for a successful glitch. However, it should be noted that most successful glitches were still found at a GNDA of 0.0 V. We believe that a negative GNDA causes the target to reset faster. We were surprised that the successful glitches in all test cases are very close together. Prior to the evaluation, we assumed a greater dispersion. Furthermore, we were surprised that for an activated brownout protection with a brownout protection threshold of 2.7 V, the number of successful glitches was significantly higher than with deactivated brownout protection. In contrast, with an activated brownout protection with a lower brownout protection threshold of 1.8 V, it was not possible to find any successful glitches at all.

The results of the second evaluation with a 100 nF decoupling capacitor have confirmed the results of the first evaluation. If GNDA is chosen lower, this results in a lower required *width* for a successful glitch. However, the maximum possible *width* for a successful glitch has also decreased with the decrease of GNDA. Unlike to the first evaluation where the successful glitches were distributed very locally, in the second evaluation the successful glitches had a distributed shape. Nevertheless, the glitches were still distributed according to a clear pattern. Due to the high capacitance of the decoupling capacitor, it was not possible to pull down the power supply rail of the target far enough with a GNDA of 0.0 V. For that reason, no successful glitches were identified for test cases 2.1.1, 2.2.1 and 2.3.1. Since we were not able to identify successful glitches for an activated brownout protection with a brownout threshold of 1.8 V in the first evaluation, we were surprised by the fact that we found successful glitches for this configuration in the second evaluation.

The hypothesis of this work was that negative voltage fault injection attacks provide advantages over their conventional counterparts with respect to higher slew rates and shorter glitch durations in presence of capacitive and inductive charges within microcontrollers. Our evaluation showed that negative voltage fault injection attacks indeed provide advantages over their conventional counterparts. Especially in our second evaluation, where we intentionally chose a high decoupling capacity to simulate targets with on-chip decoupling technology, the advantages are obvious. With a GNDA of 0.0 V that corresponds to conventional voltage fault injection attacks, it was infeasible to identify any successful glitches at all. In contrast, with a -2.0 V, -4.0 V or -6.0 V negative voltage for GNDA, a significant number of successful glitches could be identified. Both evaluations showed that the *width* parameter can be minimized by negative voltage fault injection attacks. We believe this result is especially significant for faster targets where conventional voltage fault injection attacks are less applicable since the minimum width of the voltage glitch is increasingly becoming longer than the period of the system clock.

CHAPTER 9

Further Work

This thesis showed that negative voltage fault injection attacks provide advantages over their conventional counterparts. However, the attacks were performed on a microcontroller that does not include countermeasures against fault injection attacks. In further work, it should be tested if the countermeasures in those devices can be bypassed with negative voltage fault injection attacks. In addition, we are looking forward to test microcontrollers from different manufacturers as well.

The two evaluations also raised new questions that originally were not part of this work. In the second evaluation, for all test cases with a GNDA of -4.0 V and -6.0 V, a horizontal cut at a *width* of 35.0% was identified. At a higher *width*, glitches with significantly lower offsets could be observed while at a lower *width* those glitches abruptly disappear. This raises the questions why the glitches appear so abruptly and whether this could be used for further attacks. In the first evaluation another question concerns the brownout protection. With activated brownout detection at a detection threshold of 2.7 V, significantly more successful glitches could be generated in comparison to a deactivated brownout protection. However, with a lower brownout detection threshold of 1.8 V it was not possible to find a successful glitch at all. It is thus an open question why the success rate benefits from one threshold and the other threshold prevents glitches instead.

There are several ideas to improve the negative voltage fault injection attack hardware within further work. Currently, the GNDA voltage is configured mechanically via a potentiometer. The possibility to set up the voltage digitally would simplify the configuration of GNDA. Besides, the current hardware implementation requires additional hardware that generates the glitch signal. In that regard, the negative voltage fault injection hardware could be extended by an FPGA to include the glitch signal generation as well so that additional hardware is no longer necessary. Furthermore, additional functionality such as clock synchronization and power analysis could be implemented

within the FPGA so that glitches can be synchronized more easily to real world target devices.

APPENDIX A

Printed Circuit Boards

A.1 Voltage Fault Injection Hardware Prototype

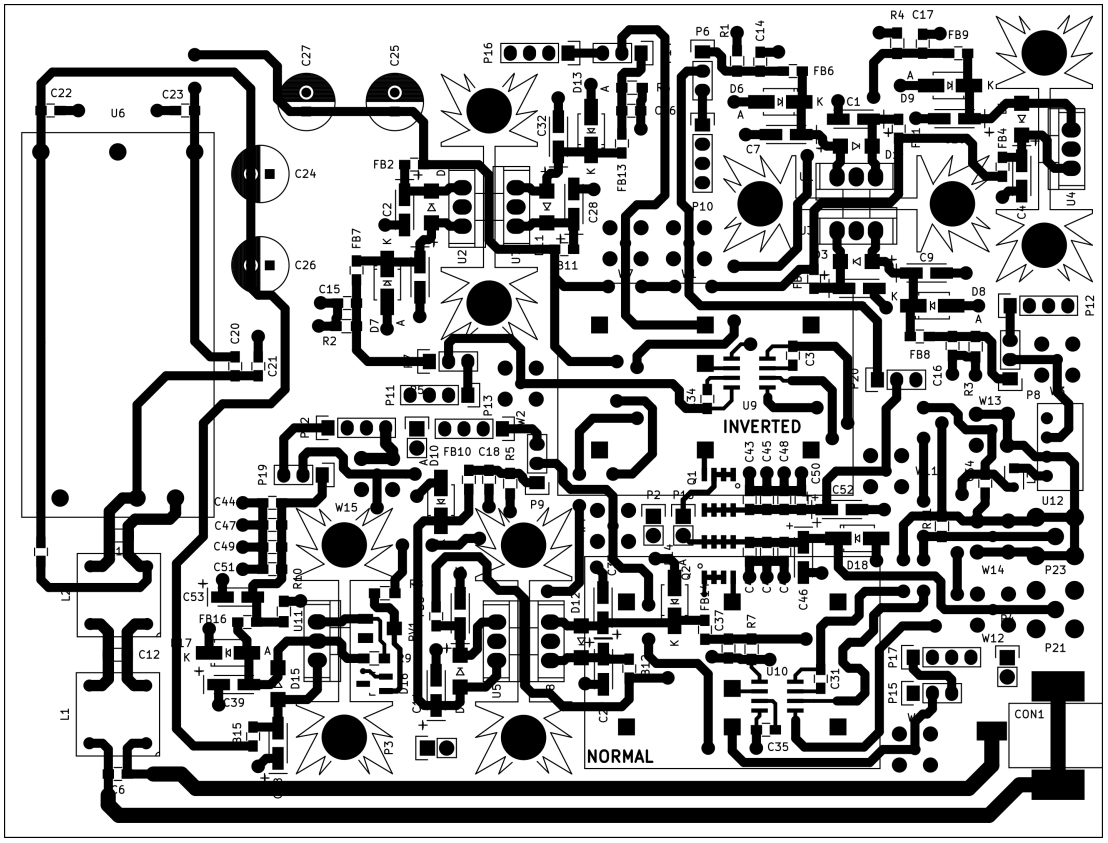

Figure A.1: Front Side of the Voltage Fault Injection Hardware Prototype

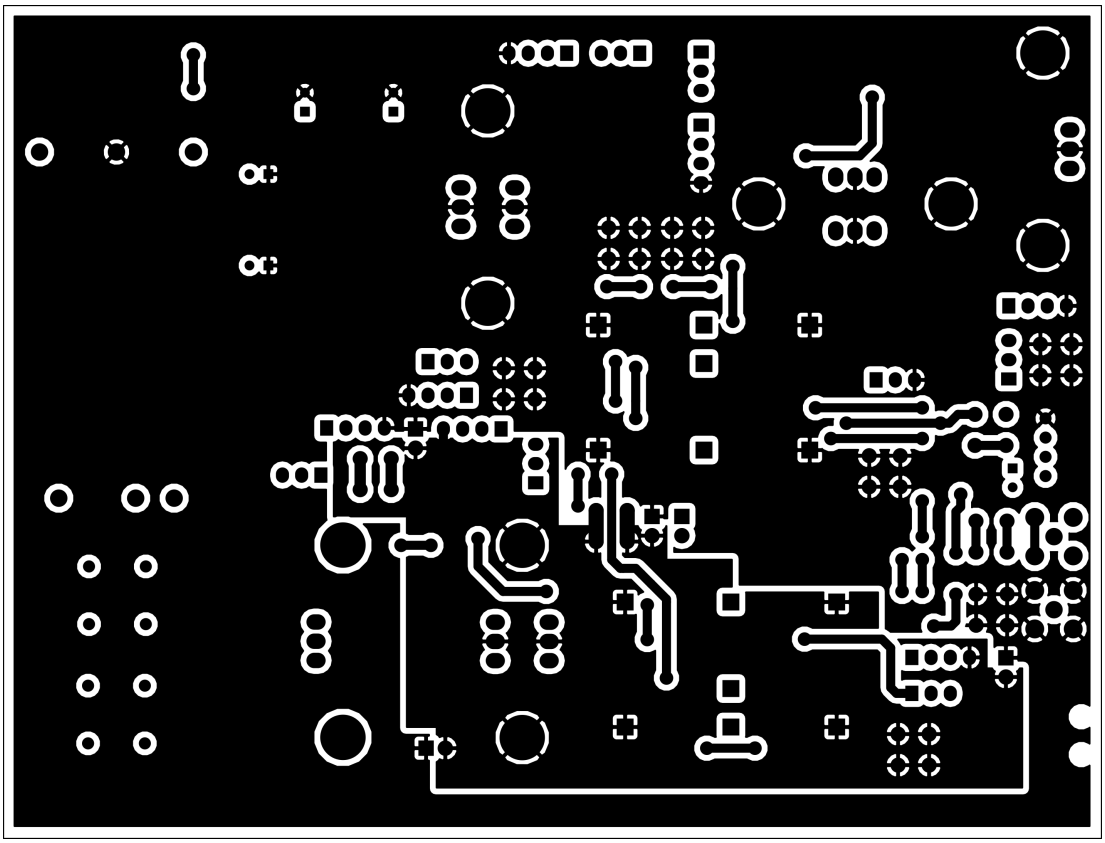

Figure A.2: Back Side of the Voltage Fault Injection Hardware Prototype

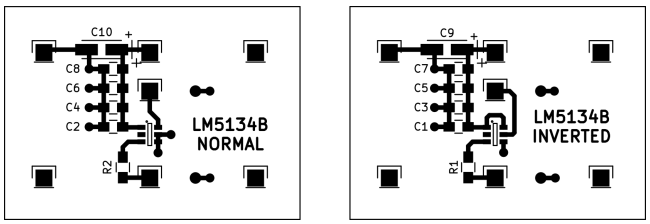

Figure A.3: Front Side of the Voltage Fault Injection Hardware MOSFET Driver Adapters

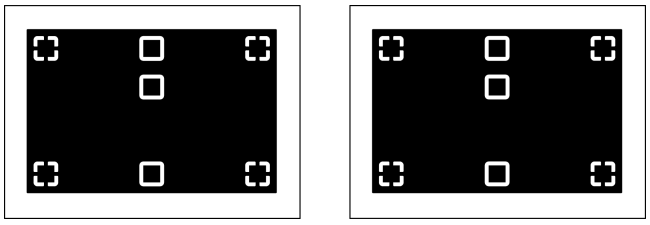

Figure A.4: Back Side of the Voltage Fault Injection Hardware MOSFET Driver Adapters

A.2 ATmega Target

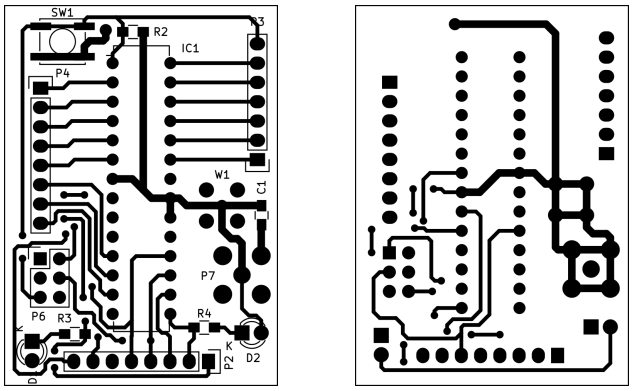

Figure A.5: Left: Front Side of ATmega Target, Right: Back Side of ATmega Target

APPENDIX B

Source Code

B.1 Evaluation Software

B.1.1 capture.py

```
from database import Database
from models import Glitch, Waveform
#from oscilloscope import Oscilloscope
from cwuserscript import CWBroker, CWUserScript
from chipwhisperer.common.api.CWCoreAPI import CWCoreAPI
import configparser
import numpy as np
import logging
import datetime
import io
import signal
from decimal import Decimal, getcontext


STOPPED = False


def signal_handler(signal, frame):
    global STOPPED
    STOPPED = True


class GlitchNotRecordedException(Exception):
    pass


class Capture():
    def __init__(self, project_name, iteration_count, glitch_offset,
                 glitch_width, glitch_repeat, glitch_v_gs_low,
                 atmega_flags_low, atmega_flags_high, atmega_flags_extended,
                 enable_oscilloscope, fast_continue):
        self.log = logging.getLogger(self.__class__.__name__)
        self.log.setLevel(logging.INFO)

        username, password, database = self.get_mysql_settings()
        self.log.debug("init database")
        self.db = Database(username, password, database)

        self.project = self.db.get_or_create_project(project_name)
```

```
        offset_step, width_step, repeat_step = self.calculate_step_values(glitch_offset,
                                                                           glitch_width,
                                                                           glitch_repeat)
        self.setup = self.db.get_or_create_setup(iteration_count,
                                                 glitch_offset[:1][0],
                                                 glitch_offset[-1:][0],
                                                 offset_step,
                                                 glitch_width[:1][0],
                                                 glitch_width[-1:][0],
                                                 width_step,
                                                 glitch_repeat[:1][0],
                                                 glitch_repeat[-1:][0],
                                                 repeat_step,
                                                 glitch_v_gs_low,
                                                 atmega_flags_low,
                                                 atmega_flags_high,
                                                 atmega_flags_extended)

        self.iteration_count = iteration_count
        self.glitch_offset = glitch_offset
        self.glitch_width = glitch_width
        self.glitch_repeat = glitch_repeat
        self.counter = 1

        self.glitch_count = self.calculate_glitch_count()

        self.enable_oscilloscope = enable_oscilloscope
        if self.enable_oscilloscope:
            self.init_oscilloscope()

        self.fast_continue = fast_continue

        self.init_CW()

    def calculate_step_values(self, glitch_offset, glitch_width, glitch_repeat):
        if len(glitch_offset) > 1:
            offset_step = glitch_offset[1] - glitch_offset[0]
        else:
            offset_step = 0

        if len(glitch_width) > 1:
            width_step = glitch_width[1] - glitch_width[0]
        else:
            width_step = 0

        if len(glitch_repeat) > 1:
            repeat_step = glitch_repeat[1] - glitch_repeat[0]
        else:
            repeat_step = 0

        return offset_step, width_step, repeat_step

    def init_CW(self):
        self.cw = CWCoreAPI()
        self.cw.runScriptClass(CWUserScript, funcName="init")

    def init_oscilloscope(self):
        self.osci = Oscilloscope()

        # Default Setup
        self.osci.query(":system:preset")

        # Set horizontal scale and offset
        self.osci.write(":timebase:scale 0.0000002")
        self.osci.write(":timebase:position 0.0000008")

        # Configure channel 1
        self.osci.write(":channel1:display ON")
        self.osci.write(":channel1:scale 1.500")
        self.osci.write(":channel1:offset 3.3000")

        # Configure channel 2
```

```
        self.osci.write(":channel2:display ON")
        self.osci.write(":channel2:scale 1.500")
        self.osci.write(":channel2:offset 0.000")

        # Configure channel 3
        self.osci.write(":channel3:display ON")
        self.osci.write(":channel3:scale 1.500")
        self.osci.write(":channel3:offset 0.000")

        # Configure channel 4
        self.osci.write(":channel4:display ON")
        self.osci.write(":channel4:scale 1.500")
        self.osci.write(":channel4:offset 0.000")

        # Configure trigger
        self.osci.write(":trigger:edge:source channel3")
        self.osci.write(":trigger:edge:level 2.000")

    def calculate_glitch_count(self):
        return self.iteration_count * len(self.glitch_offset) \
            * len(self.glitch_width) * len(self.glitch_repeat)

    def capture(self):
        global STOPPED

        if fast_continue:
            max_iteration = self.db.get_max(Glitch.iteration, [(Glitch.project_id, self.project.id),
                                                               (Glitch.setup_id, self.setup.id)])
            max_offset = self.db.get_max(Glitch.offset, [(Glitch.project_id, self.project.id),
                                                         (Glitch.setup_id, self.setup.id),
                                                         (Glitch.iteration, max_iteration)])

            if max_iteration is not None and max_offset is not None:
                new_offset = create_np_array(max_offset, self.setup.offset_to, self.setup.offset_step)

                count_todo = (self.iteration_count - max_iteration + 1) \
                    * len(new_offset) * len(self.glitch_width) * len(self.glitch_repeat)

                self.glitch_offset = new_offset
                self.counter = self.glitch_count - count_todo
            else:
                max_iteration = 1

        for iteration in xrange(max_iteration, self.iteration_count + 1):
            for offset in self.glitch_offset:
                for width in self.glitch_width:
                    for repeat in self.glitch_repeat:
                        while True:
                            if self.do_glitch(iteration, offset, width, repeat):
                                break
                        self.counter += 1
                        if STOPPED:
                            return

    def do_glitch(self, iteration, offset, width, repeat):
        log_str = "glitch " + str(self.counter) + "/" + str(self.glitch_count) + ": "

        if self.db.glitch_exists(iteration, offset, width, repeat,
                                 self.project.id, self.setup.id):
            self.log.info(log_str + "already in database, continuing")
            return True

        self.log.info(log_str + "iteration: %d, offset: %f, width: %f, repeat: %d, v_gs_low: %f" %
                      (iteration, offset, width, repeat, self.setup.v_gs_low))

        CWBroker.getInstance().set_glitch_parameter(offset, width, repeat)

        if self.enable_oscilloscope:
            self.osci.query(":single")

        self.cw.runScriptClass(CWUserScript)

        response = CWBroker.getInstance().get_response()
```

```
            if response is "":
                self.log.info(log_str + "error: empty response")
                self.cw.runScriptClass(CWUserScript, funcName="init")

                return False

            if "1234" in response:
                status = "success"
            elif response.count("hello") > 1:
                status = "reset"
            else:
                status = "normal"
            self.log.info(log_str + "status: " + status + "\n\n")

            if self.enable_oscilloscope:
                if int(self.osci.query(":operegister:condition?")) & 0x08 == 8:
                    raise GlitchNotRecordedException("the glitch was not recorded")

            glitch = Glitch(iteration, datetime.datetime.now(), status, response,
                            offset, width, repeat, self.project, self.setup)
            self.db.insert(glitch)

            if self.enable_oscilloscope:
                channel_glitch = self.osci.get_waveform("channel1")
                waveform_glitch_compressed = io.BytesIO()
                np.savez_compressed(waveform_glitch_compressed, np.array(channel_glitch))
                waveform_glitch = Waveform("Glitch", waveform_glitch_compressed.getvalue(), glitch)
                self.db.insert(waveform_glitch)

                channel_enable = self.osci.get_waveform("channel2")
                waveform_enable_compressed = io.BytesIO()
                np.savez_compressed(waveform_enable_compressed, np.array(channel_enable))
                waveform_enable = Waveform("Enable", waveform_enable_compressed.getvalue(), glitch)
                self.db.insert(waveform_enable)

                channel_trigger = self.osci.get_waveform("channel3")
                waveform_trigger_compressed = io.BytesIO()
                np.savez_compressed(waveform_trigger_compressed, np.array(channel_trigger))
                waveform_trigger = Waveform("Trigger", waveform_trigger_compressed.getvalue(), glitch)
                self.db.insert(waveform_trigger)

                channel_clock = self.osci.get_waveform("channel4")
                waveform_clock_compressed = io.BytesIO()
                np.savez_compressed(waveform_clock_compressed, np.array(channel_clock))
                waveform_clock = Waveform("Clock", waveform_clock_compressed.getvalue(), glitch)
                self.db.insert(waveform_clock)

            return True

        def get_mysql_settings(self):
            config = configparser.ConfigParser()
            config.read("settings.ini")

            username = config["mysql"]["username"]
            password = config["mysql"]["password"]
            database = config["mysql"]["database"]

            return username, password, database


    def create_np_array(start, stop, step):
        return np.arange(start, stop + step, step)


    if __name__ == "__main__":
        signal.signal(signal.SIGINT, signal_handler)

        logging.basicConfig(level=logging.CRITICAL,
                            format="%(asctime)s:%(name)s:%(levelname)s:%(message)s",
                            datefmt="%Y-%m-%d %H:%M:%S")

        getcontext().prec = 3
```

```
    project_name = "atmega_capacitor_evaluation"
    iteration_count = 1
    glitch_offset = create_np_array(Decimal(-49.0), Decimal(49.0), Decimal(0.2))
    glitch_width = create_np_array(Decimal(4.0), Decimal(6.0), Decimal(0.05))
    glitch_repeat = create_np_array(1, 5, 1)
    glitch_v_gs_low = 0.0
    atmega_flags_low = 0xe0
    atmega_flags_high = 0xd9
    atmega_flags_extended = 0xfd

    enable_oscilloscope = False
    fast_continue = True

    print len(glitch_offset) * len(glitch_width) * len(glitch_repeat)

    capture = Capture(project_name, iteration_count, glitch_offset,
                      glitch_width, glitch_repeat, glitch_v_gs_low,
                      atmega_flags_low, atmega_flags_high, atmega_flags_extended,
                      enable_oscilloscope, fast_continue)
    capture.capture()
```

B.1.2 database.py

```
from sqlalchemy import *
from sqlalchemy.orm import sessionmaker
from models import Project, Glitch, Setup
import logging


Session = sessionmaker()


class Database:
    def __init__(self, username, password, database):
        self.log = logging.getLogger(self.__class__.__name__)
        self.log.setLevel(logging.INFO)

        self.log.debug("connecting to database")
        uri = "mysql://" + username + ":" + password + "@localhost/" + database
        engine = create_engine(uri)

        Session.configure(bind=engine)

        self.session = Session()

    def get_max(self, entity, filter_entities):
        filter_list = []
        for filter_entity in filter_entities:
            filter_list.append(filter_entity[0] == filter_entity[1])

        return self.session.query(func.max(entity)).filter(and_(*filter_list)).first()[0]

    def insert(self, object):
        self.session.add(object)
        self.session.commit()

    def get_success_glitches(self, project, setup):
        and_statement = and_(Glitch.status == "success",
                             Glitch.project_id == project.id,
                             Glitch.setup_id == setup.id)
        return self.session.query(Glitch).filter(and_statement).all()

    def get_normal_glitches(self, project, setup):
        and_statement = and_(Glitch.status == "normal",
                             Glitch.project_id == project.id,
                             Glitch.setup_id == setup.id)
        return self.session.query(Glitch).filter(and_statement).all()

    def get_reset_glitches(self, project, setup):
```

```
        and_statement = and_(Glitch.status == "reset",
                             Glitch.project_id == project.id,
                             Glitch.setup_id == setup.id)
        return self.session.query(Glitch).filter(and_statement).all()

    def get_or_create_project(self, name):
        project = self.get_project(name)

        if project is None:
            self.log.debug("can't find project \"" + name + "\", creating it")
            project = Project(name)
            self.insert(project)

        return project

    def get_projects(self):
        return self.session.query(Project).all()

    def get_project(self, name):
        return self.session.query(Project).filter_by(name=name).first()

    def get_or_create_setup(self, iteration_count, offset_from, offset_to, offset_step,
                            width_from, width_to, width_step, repeat_from, repeat_to,
                            repeat_step, v_gs_low, flags_low, flags_high, flags_extended):
        setup = self.get_setup(iteration_count, offset_from, offset_to, offset_step,
                               width_from, width_to, width_step, repeat_from, repeat_to,
                               repeat_step, v_gs_low, flags_low, flags_high, flags_extended)

        if setup is None:
            self.log.debug("can't find current setup, creating it")
            setup = Setup(iteration_count, offset_from, offset_to, offset_step,
                          width_from, width_to, width_step, repeat_from, repeat_to,
                          repeat_step, v_gs_low, flags_low, flags_high, flags_extended)
            self.insert(setup)

        return setup

    def get_setup(self, iteration_count, offset_from, offset_to, offset_step,
                  width_from, width_to, width_step, repeat_from, repeat_to,
                  repeat_step, v_gs_low, flags_low, flags_high, flags_extended):
        and_statement = and_(Setup.iteration_count == iteration_count,
                             Setup.offset_from == offset_from,
                             Setup.offset_to == offset_to,
                             Setup.offset_step == offset_step,
                             Setup.width_from == width_from,
                             Setup.width_to == width_to,
                             Setup.width_step == width_step,
                             Setup.repeat_from == repeat_from,
                             Setup.repeat_to == repeat_to,
                             Setup.repeat_step == repeat_step,
                             Setup.v_gs_low == v_gs_low,
                             Setup.flags_low == flags_low,
                             Setup.flags_high == flags_high,
                             Setup.flags_extended == flags_extended)

        return self.session.query(Setup).filter(and_statement).first()

    def glitch_exists(self, iteration, offset, width, repeat,
                      project_id, setup_id):
        and_statement = and_(Glitch.iteration == iteration,
                             Glitch.offset == offset,
                             Glitch.width == width,
                             Glitch.repeat == repeat,
                             Glitch.project_id == project_id,
                             Glitch.setup_id == setup_id)

        glitch = self.session.query(Glitch).filter(and_statement).first()

        if glitch is None:
            return False
        return True

    def get_setups_by_project(self, project):
```

```
        return self.session.query(Setup).join(Glitch, Setup.id == Glitch.setup_id) \
            .join(Project, Glitch.project_id == Project.id).filter(Project.name == project.name) \
            .distinct().all()

    def get_setup_by_id(self, project_id):
        return self.session.query(Setup).filter_by(id=project_id).first()
```

B.1.3 models.py

```
from sqlalchemy import Column, DateTime, Enum, ForeignKey, Integer, Numeric, String
from sqlalchemy.dialects.mysql.types import MEDIUMBLOB, TINYBLOB
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base


Base = declarative_base()
metadata = Base.metadata


class Glitch(Base):
    __tablename__ = 'glitch'

    id = Column(Integer, primary_key=True)
    iteration = Column(Integer, nullable=False)
    time = Column(DateTime, nullable=False)
    status = Column(Enum(u'normal', u'success', u'reset'), nullable=False)
    answer = Column(TINYBLOB, nullable=False)
    offset = Column(Numeric(6, 3), nullable=False)
    width = Column(Numeric(6, 3), nullable=False)
    repeat = Column(Integer, nullable=False)
    project_id = Column(ForeignKey(u'project.id'), nullable=False, index=True)
    setup_id = Column(ForeignKey(u'setup.id'), nullable=False, index=True)

    project = relationship(u'Project')
    setup = relationship(u'Setup')

    def __init__(self, iteration, time, status, answer, offset, width, repeat,
                 project, setup):
        self.iteration = iteration
        self.time = time
        self.status = status
        self.answer = answer
        self.offset = offset
        self.width = width
        self.repeat = repeat
        self.project = project
        self.setup = setup


class Project(Base):
    __tablename__ = 'project'

    id = Column(Integer, primary_key=True)
    name = Column(String(100), nullable=False, unique=True)

    def __init__(self, name):
        self.name = name


class Setup(Base):
    __tablename__ = 'setup'

    id = Column(Integer, primary_key=True)
    iteration_count = Column(Integer, nullable=False)
    offset_from = Column(Numeric(6, 3), nullable=False)
    offset_to = Column(Numeric(6, 3), nullable=False)
    offset_step = Column(Numeric(6, 3), nullable=False)
    width_from = Column(Numeric(6, 3), nullable=False)
    width_to = Column(Numeric(6, 3), nullable=False)
    width_step = Column(Numeric(6, 3), nullable=False)
```

```
    repeat_from = Column(Integer, nullable=False)
    repeat_to = Column(Integer, nullable=False)
    repeat_step = Column(Integer, nullable=False)
    v_gs_low = Column(Numeric(6, 3), nullable=False)
    flags_low = Column(Integer, nullable=False)
    flags_high = Column(Integer, nullable=False)
    flags_extended = Column(Integer, nullable=False)

    def __init__(self, iteration_count, offset_from, offset_to, offset_step,
                 width_from, width_to, width_step, repeat_from, repeat_to,
                 repeat_step, v_gs_low, flags_low, flags_high, flags_extended):
        self.iteration_count = iteration_count
        self.offset_from = offset_from
        self.offset_to = offset_to
        self.offset_step = offset_step
        self.width_from = width_from
        self.width_to = width_to
        self.width_step = width_step
        self.repeat_from = repeat_from
        self.repeat_to = repeat_to
        self.repeat_step = repeat_step
        self.v_gs_low = v_gs_low
        self.flags_low = flags_low
        self.flags_high = flags_high
        self.flags_extended = flags_extended


class Waveform(Base):
    __tablename__ = 'waveform'

    id = Column(Integer, primary_key=True)
    label = Column(String(100), nullable=False)
    data = Column(MEDIUMBLOB, nullable=False)
    glitch_id = Column(ForeignKey(u'glitch.id'), nullable=False, index=True)

    glitch = relationship(u'Glitch')

    def __init__(self, label, data, glitch):
        self.label = label
        self.data = data
        self.glitch = glitch
```

B.1.4 cwuserscript.py

```
from chipwhisperer.common.scripts.base import UserScriptBase


class GlitchParametersNotNoneException(Exception):
    pass


class GlitchParametersNoneException(Exception):
    pass


class ResponseNotNoneException(Exception):
    pass


class ResponseNoneException(Exception):
    pass


class CWBroker:
    __instance = None

    @staticmethod
    def getInstance():
        if CWBroker.__instance is None:
            CWBroker()
```

```
        return CWBroker.__instance

    def __init__(self):
        self.__glitch_parameters = None
        self.__response = None

        if CWBroker.__instance is not None:
            raise Exception("This class is a singleton!")
        else:
            CWBroker.__instance = self

    def set_glitch_parameter(self, offset, width, repeat):
        if self.__glitch_parameters is not None:
            raise GlitchParametersNotNoneException("The glitch parameters are not none!")

        self.__glitch_parameters = (offset, width, repeat)

    def get_glitch_parameter(self):
        if self.__glitch_parameters is None:
            raise GlitchParametersNoneException("The glitch parameters are none!")

        temp_glitch_parameters = self.__glitch_parameters
        self.__glitch_parameters = None
        return temp_glitch_parameters

    def set_response(self, response):
        if self.__response is not None:
            raise ResponseNotNoneException("The response is not none!")

        self.__response = response

    def get_response(self):
        if self.__response is None:
            raise ResponseNoneException("The response is none!")

        temp_response = self.__response
        self.__response = None
        return temp_response


class CWUserScript(UserScriptBase):
    def __init__(self, api):
        super(CWUserScript, self).__init__(api)

    def init(self):
        self.api.setParameter(["Generic Settings", "Scope Module", "ChipWhisperer/OpenADC"])
        self.api.setParameter(["Generic Settings", "Target Module", "Simple Serial"])
        self.api.setParameter(["Generic Settings", "Trace Format", "None"])
        self.api.setParameter(["Generic Settings", "Auxiliary Module", "GPIO Toggle"])
        self.api.setParameter(["Simple Serial", "Connection", "NewAE USB (CWLite/CW1200)"])
        self.api.setParameter(["ChipWhisperer/OpenADC", "Connection", "NewAE USB (CWLite/CW1200)"])
        self.api.setParameter(["Aux Settings", "GPIO Toggle", "GPIO Pin", "nRST"])
        self.api.setParameter(["Aux Settings", "GPIO Toggle", "Standby State", "High"])
        self.api.connect()

        self.api.setParameter(["OpenADC", "Clock Setup", "CLKGEN Settings", "Desired Frequency", 7372800.0])
        self.api.setParameter(["OpenADC", "Clock Setup", "ADC Clock", "Reset ADC DCM", None])
        self.api.setParameter(["CW Extra Settings", "Target HS IO-Out", "CLKGEN"])

        self.api.setParameter(["Glitch Module", "Clock Source", "CLKGEN"])
        self.api.setParameter(["Glitch Module", "Glitch Trigger", "Ext Trigger:Single-Shot"])
        self.api.setParameter(["Glitch Module", "Output Mode", "Glitch Only"])
        self.api.setParameter(["CW Extra Settings", "HS-Glitch Out Enable (High Power)", 1])
        self.api.setParameter(["Glitch Module", "Ext Trigger Offset", 0])

    def run(self):
        offset, width, repeat = CWBroker.getInstance().get_glitch_parameter()

        self.serial = self.api.getTarget().ser
        self.serial.terminal_flush()

        self.api.setParameter(["Glitch Module", "Glitch Width (as % of period)", float(width)])
        self.api.setParameter(["Glitch Module", "Glitch Offset (as % of period)", float(offset)])
```

```
        self.api.setParameter(["Glitch Module", "Repeat", int(repeat)])
        self.api.capture1()

        bavail = self.serial.terminal_inWaiting()
        data = []
        while bavail > 0:
            data += self.serial.terminal_read(bavail)
            bavail = self.serial.terminal_inWaiting()

        response = ""
        for symbol in data:
            if symbol[0] == "in":
                response += symbol[1]
        CWBroker.getInstance().set_response(response)
```

B.1.5 oscilloscope.py

```
import visa
import struct


class OscilloscopeException(Exception):
    pass


class Oscilloscope:
    def __init__(self, idVendor=0x0957, idProduct=0x17a0):
        rm = visa.ResourceManager("@py")

        osci_str = None
        for resource in rm.list_resources():
            if str(idVendor) in resource and str(idProduct) in resource:
                osci_str = resource

        if osci_str is None:
            raise OscilloscopeException("Error: Can't find Oscilloscope!")

        self.osci = rm.open_resource(osci_str)

    def query(self, str):
        while True:
            try:
                return self.osci.query(str)
                break
            except ValueError:
                continue

    def write(self, str):
        while True:
            try:
                self.osci.write(str)
                break
            except ValueError:
                continue

    def get_waveform(self, source):
        self.write(":waveform:points:mode normal")
        self.write(":waveform:source %s" % source)
        self.write(":waveform:format byte")

        # Read preamble
        pre = self.query(":waveform:preamble?")
        pre = pre.split(',')

        format = int(pre[0])
        type = int(pre[1])
        xincrement = float(pre[4])
        xorigin = float(pre[5])
        xreference = int(float(pre[6]))
        yincrement = float(pre[7])
```

```
        yorigin = float(pre[8])
        yreference = int(float(pre[9]))

        if type == 1:
            raise Exception()

        if format != 0:
            raise Exception()

        # Read waveform data
        sData = self.osci.query_binary_values(":waveform:data?",
                                              datatype="s")[0]
        values = struct.unpack("%dB" % len(sData), sData)

        data = []
        for i in xrange(0, len(values) - 1):
            time_val = xorigin + (i - xreference) * xincrement
            voltage = ((values[i] - yreference) * yincrement) + yorigin
            data.append((time_val, voltage))
        return data
```

B.2 Glitch Software

B.2.1 glitch.py

```
from cwuserscript import CWBroker, CWUserScript
from chipwhisperer.common.api.CWCoreAPI import CWCoreAPI
from decimal import Decimal, getcontext
import argparse


def glitch(offset, width, repeat, print_response, iterate):
    cw = CWCoreAPI()
    cw.runScriptClass(CWUserScript, funcName="init")

    for i in xrange(1, iterate + 1):
        CWBroker.getInstance().set_glitch_parameter(offset, width, repeat)
        cw.runScriptClass(CWUserScript)
        response = CWBroker.getInstance().get_response()

        if response is "":
            print "Error: empty response!"

        if "1234" in response:
            status = "success"
        elif response.count("hello") > 1:
            status = "reset"
        else:
            status = "normal"

        if print_response:
            print "Glitch status: " + status + "\tresponse: " + response.replace("\n", "")
        else:
            print "Glitch status: " + status


if __name__ == "__main__":
    getcontext().prec = 3

    parser = argparse.ArgumentParser()
    parser.add_argument("offset", type=float)
    parser.add_argument("width", type=float)
    parser.add_argument("repeat", type=int)
    parser.add_argument("--iterate", type=int)
    parser.add_argument("--response", action="store_true")
    parser.set_defaults(response=False)
    parser.set_defaults(iterate=1)
    args = parser.parse_args()
```

```
    glitch(Decimal(args.offset), Decimal(args.width), args.repeat, args.response, args.iterate)
```

B.3 Plot Software 2D

B.3.1 plot2d.py

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.legend as legend
import numpy as np
from decimal import *
from database import Database
import configparser
import sys
import argparse


def get_mysql_settings():
    config = configparser.ConfigParser()
    config.read("settings.ini")

    username = config["mysql"]["username"]
    password = config["mysql"]["password"]
    database = config["mysql"]["database"]

    return username, password, database


def create_plot(success_offsets, success_widths, reset_offsets, reset_widths,
                normal_offsets, normal_widths):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection="rectilinear")

    ax.scatter(normal_offsets, normal_widths, c="white", marker=".")
    ax.scatter(reset_offsets, reset_widths, c="gray", marker=".")
    ax.scatter(success_offsets, success_widths, c="black", marker=".")

    ax.set_xlim(left=-49.0, right=49.0)
    ax.set_xticks(np.arange(-50.0, 60.0, 10.0))

    ax.set_ylim(bottom=7.0, top=49.0)
    ax.set_yticks(np.arange(7.0, 49.0, 5.0))

    ax.set_xlabel("Offset")
    ax.set_ylabel("Width")

    title_str = "success: " + str(len(success_offsets)) + " (black)\n" \
                "reset: " + str(len(reset_offsets)) + " (gray)\n" \
                "normal: " + str(len(normal_offsets)) + " (white)"
    leg = plt.legend([], [], loc="lower left", title=title_str)
    leg.get_title().set_position((0, -5))
    plt.show()


def run(project_name, setup_id, db):
    project = db.get_project(project_name)

    if project is None:
        print "Error: can't find project!"
        sys.exit(1)

    setup = db.get_setup_by_id(setup_id)

    if setup is None:
        print "Error: can't find setup!"
        sys.exit(1)
```

```
    success = db.get_success_glitches(project, setup)
    reset = db.get_reset_glitches(project, setup)
    normal = db.get_normal_glitches(project, setup)

    brownout = get_brownout_str(setup.flags_extended)
    print "setup id: " + str(setup.id) + "\t\tv_gs_low: " + str(setup.v_gs_low) \
        + "\t\tbrownout: " + brownout + "\t\tsuccess: " + str(len(success)) \
        + "\t\treset: " + str(len(reset)) + "\t\tnormal: " + str(len(normal))

    success_offsets = []
    success_widths = []

    for glitch in success:
        success_offsets.append(glitch.offset)
        success_widths.append(glitch.width)

    reset_offsets = []
    reset_widths = []

    for glitch in reset:
        reset_offsets.append(glitch.offset)
        reset_widths.append(glitch.width)

    normal_offsets = []
    normal_widths = []

    for glitch in normal:
        normal_offsets.append(glitch.offset)
        normal_widths.append(glitch.width)

    create_plot(success_offsets, success_widths, reset_offsets, reset_widths,
                normal_offsets, normal_widths)


def get_brownout_str(flags_extended):
    if flags_extended == 0xfd:
        return "2.7 V"
    elif flags_extended == 0xfe:
        return "1.8 V"
    else:
        return "not activated"


def list_projects(db):
    projects = db.get_projects()

    for project in projects:
        print project.name


def list_setups(project_name, db):
    project = db.get_project(project_name)

    if project is None:
        print "Error: can't find project!"
        sys.exit(1)

    for setup in db.get_setups_by_project(project):

        brownout = get_brownout_str(setup.flags_extended)

        v_gs_low_str = str(setup.v_gs_low)

        if len(v_gs_low_str) == 5:
            v_gs_low_str = " " + v_gs_low_str

        print "setup id: " + str(setup.id) + "\t\tv_gs_low: " + v_gs_low_str + "\t\tbrownout: " + brownout


if __name__ == "__main__":
    getcontext().prec = 3
```

```
    if len(sys.argv) <= 1:
        sys.argv.append("-h")

    parser = argparse.ArgumentParser()
    parser.add_argument("--plot", nargs=2, metavar=("project_name", "setup_id"))
    parser.add_argument("--list-projects", action="store_true")
    parser.add_argument("--list-setups", type=str, metavar="project_name")
    args = parser.parse_args()

    if args.plot is not None:
        try:
            setup_id = int(args.plot[1])
        except ValueError:
            parser.error("setup_id must be an integer!")

    username, password, database = get_mysql_settings()
    db = Database(username, password, database)

    if args.list_projects:
        list_projects(db)

    if args.list_setups is not None:
        list_setups(args.list_setups, db)

    if args.plot is not None:
        run(args.plot[0], setup_id, db)
```

B.4 Plot Software 3D

B.4.1 plot3d.py

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from decimal import *
from database import Database
import configparser
import sys
import argparse


def get_mysql_settings():
    config = configparser.ConfigParser()
    config.read("settings.ini")

    username = config["mysql"]["username"]
    password = config["mysql"]["password"]
    database = config["mysql"]["database"]

    return username, password, database


def create_plot(offsets, widths, repeats):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")

    ax.scatter(offsets, widths, repeats, c="b", marker="o")

    ax.set_xlim(left=-49.0, right=49.0)
    ax.set_xticks(np.arange(-50.0, 60.0, 10.0))

    ax.set_ylim(bottom=4.0, top=6.0)
    ax.set_yticks(np.arange(4.0, 6.25, 0.25))

    ax.set_zlim(bottom=1, top=5)
    ax.set_zticks([1, 2, 3, 4, 5])

    ax.set_xlabel("Offset")
```

```
    ax.set_ylabel("Width")
    ax.set_zlabel("Repeat")

    plt.show()


def run(project_name, setup_id, db):
    project = db.get_project(project_name)

    if project is None:
        print "Error: can't find project!"
        sys.exit(1)

    setup = db.get_setup_by_id(setup_id)

    if setup is None:
        print "Error: can't find setup!"
        sys.exit(1)

    glitches = db.get_success_glitches(project, setup)

    brownout = get_brownout_str(setup.flags_extended)
    print "setup id: " + str(setup.id) + "\t\tv_gs_low: " + str(setup.v_gs_low) \
        + "\t\tbrownout: " + brownout + "\t\tsuccess: " + str(len(glitches))

    offsets = []
    widths = []
    repeats = []

    for glitch in glitches:
        offsets.append(glitch.offset)
        widths.append(glitch.width)
        repeats.append(glitch.repeat)

    create_plot(offsets, widths, repeats)


def get_brownout_str(flags_extended):
    if flags_extended == 0xfd:
        return "2.7 V"
    elif flags_extended == 0xfe:
        return "1.8 V"
    else:
        return "not activated"


def list_projects(db):
    projects = db.get_projects()

    for project in projects:
        print project.name


def list_setups(project_name, db):
    project = db.get_project(project_name)

    if project is None:
        print "Error: can't find project!"
        sys.exit(1)

    for setup in db.get_setups_by_project(project):

        brownout = get_brownout_str(setup.flags_extended)

        v_gs_low_str = str(setup.v_gs_low)

        if len(v_gs_low_str) == 5:
            v_gs_low_str = " " + v_gs_low_str

        print "setup id: " + str(setup.id) + "\t\tv_gs_low: " + v_gs_low_str + "\t\tbrownout: " + brownout


if __name__ == "__main__":
```

```
getcontext().prec = 3

if len(sys.argv) <= 1:
    sys.argv.append("-h")

parser = argparse.ArgumentParser()
parser.add_argument("--plot", nargs=2, metavar=("project_name", "setup_id"))
parser.add_argument("--list-projects", action="store_true")
parser.add_argument("--list-setups", type=str, metavar="project_name")
args = parser.parse_args()

if args.plot is not None:
    try:
        setup_id = int(args.plot[1])
    except ValueError:
        parser.error("setup_id must be an integer!")

username, password, database = get_mysql_settings()
db = Database(username, password, database)

if args.list_projects:
    list_projects(db)

if args.list_setups is not None:
    list_setups(args.list_setups, db)

if args.plot is not None:
    run(args.plot[0], setup_id, db)
```

B.5 Oscilloscope to CSV

B.5.1 osci2csv.py

```
from oscilloscope import Oscilloscope
import sys
import argparse
from decimal import Decimal


def run(channel1_name, channel2_name, channel3_name, channel4_name, filename, delimiter=","):
    osci = Oscilloscope()
    osci.query(":single")

    while int(osci.query(":operegister:condition?")) & 0x08 == 8:
        pass

    master = None
    csv = "time"

    if channel1_name is not None:
        channel1 = osci.get_waveform("channel1")
        master = channel1
        csv += delimiter + channel1_name

    if channel2_name is not None:
        channel2 = osci.get_waveform("channel2")
        master = channel2
        csv += delimiter + channel2_name

    if channel3_name is not None:
        channel3 = osci.get_waveform("channel3")
        master = channel3
        csv += delimiter + channel3_name

    if channel4_name is not None:
        channel4 = osci.get_waveform("channel4")
        master = channel4
        csv += delimiter + channel4_name
```

```
    csv += "\n"

    starting_value = Decimal(master[0][0])
    for i in xrange(1, len(master)):
        time = Decimal(master[i][0]) - starting_value
        csv += time.to_eng_string()

        if channel1_name is not None:
            csv += delimiter + str(channel1[i][1])

        if channel2_name is not None:
            csv += delimiter + str(channel2[i][1])

        if channel3_name is not None:
            csv += delimiter + str(channel3[i][1])

        if channel4_name is not None:
            csv += delimiter + str(channel4[i][1])

        csv += "\n"

    if filename is not None:
        with open(filename, "w") as f:
            f.write(csv)
    else:
        print csv


if __name__ == "__main__":
    if len(sys.argv) <= 1:
        sys.argv.append("-h")

    parser = argparse.ArgumentParser()
    parser.add_argument("--channel1", type=str, dest="channel1", metavar="name")
    parser.add_argument("--channel2", type=str, dest="channel2", metavar="name")
    parser.add_argument("--channel3", type=str, dest="channel3", metavar="name")
    parser.add_argument("--channel4", type=str, dest="channel4", metavar="name")
    parser.add_argument("-o", "--output", type=str, dest="filename", metavar="filename")
    args = parser.parse_args()

    if args.channel1 is None and args.channel2 is None and args.channel3 is None and args.channel4 is None:
        parser.error("At least one channel is required!")

    run(args.channel1, args.channel2, args.channel3, args.channel4, args.filename)
```

B.6 ATmega328P Target Firmware

B.6.1 main.c

```
#include <avr/io.h>
#include <stdio.h>

#define F_CPU 7372800UL  // 7.37 MHz
#include <util/delay.h>

#define trigger_setup() DDRC |= 0x01
#define trigger_high()  PORTC |= 0x01
#define trigger_low()   PORTC &= ~(0x01)

#define BAUD_RATE 38400
#define BAUD_RATE_REG (unsigned int)(F_CPU / (16 * BAUD_RATE) ) - 1

void init_uart() {
    //turn on TX and RX
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);

    //set up baud rate
```

```
    UBRR0H = (unsigned char)(BAUD_RATE_REG >> 8);
    UBRR0L = (unsigned char)BAUD_RATE_REG;
}

void putch(char data) {
    while (!(UCSR0A & (1<<UDRE0))) {
        ;
    }

    UDR0 = data;
    return;
}

void uart_puts(char * s) {
    while(*s) {
        putch(*(s++));
    }
}

void led_setup() {
    DDRB |= 1 << PB0; // Set PB0 as output (red LED)
    DDRB |= 1 << PB1; // Set PB1 as output (green LED)
}

void led_ok(uint8_t value) {
    if(value == 1) {
        PORTB &= ~(1 << PB1); // Turn on green LED
    } else {
        PORTB |= 1 << PB1; // Turn off green LED
    }
}

void led_error(uint8_t value) {
    if(value == 1) {
        PORTB &= ~(1 << PB0); // Turn on red LED
    } else {
        PORTB |= 1 << PB0; // Turn off red LED
    }
}

void glitch1(void) {
    led_ok(1);
    led_error(0);

    //Some fake variable
    volatile uint8_t a = 0;

    putch('A');

    //External trigger logic
    trigger_high();
    trigger_low();

    //Should be an infinite loop
    while(a != 2){
    ;
    }

    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);

    uart_puts("1234");

    led_error(1);
    led_error(1);
```

```
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);

    //Several loops in order to try and prevent restarting
    while(1) {
    ;
    }
    while(1) {
    ;
    }
    while(1) {
    ;
    }
    while(1) {
    ;
    }
    while(1) {
    ;
    }
}

int main(void) {
    led_setup();
    init_uart();
    trigger_setup();

    putch('h');
    putch('e');
    putch('l');
    putch('l');
    putch('o');
    putch('\n');

    while(1) {
        glitch1();
    }

    return 1;
}
```

APPENDIX C

MySQL Database

C.1 Database glitcher

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `glitcher`
--

-- --------------------------------------------------------

--
-- Table structure for table `glitch`
--

CREATE TABLE `glitch` (
  `id` int(10) UNSIGNED NOT NULL,
  `iteration` tinyint(3) UNSIGNED NOT NULL,
  `time` datetime NOT NULL,
  `status` enum('normal','success','reset') NOT NULL,
  `answer` tinyblob NOT NULL,
  `offset` decimal(6,3) NOT NULL,
  `width` decimal(6,3) NOT NULL,
  `repeat` tinyint(3) UNSIGNED NOT NULL,
  `project_id` int(10) UNSIGNED NOT NULL,
  `setup_id` int(10) UNSIGNED NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Table structure for table `project`
--

CREATE TABLE `project` (
  `id` int(11) UNSIGNED NOT NULL,
  `name` varchar(100) NOT NULL
```

```
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Table structure for table `setup`
--

CREATE TABLE `setup` (
  `id` int(10) UNSIGNED NOT NULL,
  `iteration_count` tinyint(3) UNSIGNED NOT NULL,
  `offset_from` decimal(6,3) NOT NULL,
  `offset_to` decimal(6,3) NOT NULL,
  `offset_step` decimal(6,3) NOT NULL,
  `width_from` decimal(6,3) NOT NULL,
  `width_to` decimal(6,3) NOT NULL,
  `width_step` decimal(6,3) NOT NULL,
  `repeat_from` tinyint(3) UNSIGNED NOT NULL,
  `repeat_to` tinyint(3) UNSIGNED NOT NULL,
  `repeat_step` tinyint(3) UNSIGNED NOT NULL,
  `v_gs_low` decimal(6,3) NOT NULL,
  `flags_low` tinyint(3) UNSIGNED NOT NULL,
  `flags_high` tinyint(3) UNSIGNED NOT NULL,
  `flags_extended` tinyint(3) UNSIGNED NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Table structure for table `waveform`
--

CREATE TABLE `waveform` (
  `id` int(10) UNSIGNED NOT NULL,
  `label` varchar(100) NOT NULL,
  `data` mediumblob NOT NULL,
  `glitch_id` int(10) UNSIGNED NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Indexes for dumped tables
--

--
-- Indexes for table `glitch`
--
ALTER TABLE `glitch`
  ADD PRIMARY KEY (`id`),
  ADD KEY `project_id` (`project_id`),
  ADD KEY `setup_id` (`setup_id`);

--
-- Indexes for table `project`
--
ALTER TABLE `project`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `name` (`name`);

--
-- Indexes for table `setup`
--
ALTER TABLE `setup`
  ADD PRIMARY KEY (`id`);

--
-- Indexes for table `waveform`
--
ALTER TABLE `waveform`
  ADD PRIMARY KEY (`id`),
  ADD KEY `glitch_id` (`glitch_id`);

--
-- AUTO_INCREMENT for dumped tables
```

```
--

--
-- AUTO_INCREMENT for table `glitch`
--
ALTER TABLE `glitch`
  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=759308;

--
-- AUTO_INCREMENT for table `project`
--
ALTER TABLE `project`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;

--
-- AUTO_INCREMENT for table `setup`
--
ALTER TABLE `setup`
  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=76;

--
-- AUTO_INCREMENT for table `waveform`
--
ALTER TABLE `waveform`
  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT;

--
-- Constraints for dumped tables
--

--
-- Constraints for table `glitch`
--
ALTER TABLE `glitch`
  ADD CONSTRAINT `glitch_ibfk_1` FOREIGN KEY (`project_id`) REFERENCES `project` (`id`),
  ADD CONSTRAINT `glitch_ibfk_2` FOREIGN KEY (`setup_id`) REFERENCES `setup` (`id`);

--
-- Constraints for table `waveform`
--
ALTER TABLE `waveform`
  ADD CONSTRAINT `waveform_ibfk_1` FOREIGN KEY (`glitch_id`) REFERENCES `glitch` (`id`);
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

List of Figures

List of Tables

Glossary

hardware security Hardware security deals with physical attacks on hardware and their countermeasures. 1

security-by-obscurity Security-by-obscurity is the belief that a system of any sort can be secure so long as nobody outside of its implementation group is allowed to find out anything about its internal mechanisms. 2

Acronyms

ALU Arithmetic Logic Unit. 9–11, 13–15, 109, 111

ASCII American Standard Code for Information Interchange. 54, 56, 57

CISC Complex Instruction Set Computer. 21, 22

CMOS complementary metal-oxide-semiconductor. 6–9, 28

CSV Comma-separated values. 58

DRAM dynamic random-access memory. 17, 19, 20

EEPROM electrical erasable programmable ROM. 20

EPROM erasable programmable ROM. 19, 20

FET field-effect transistor. 6

FGMOS floating-gate MOSFET. 6, 7

FPGA Field Programmable Gate Array. 2

ISA instruction set architecture. 23

ISP in-system programming. 24

JTAG Joint Test Action Group. 24

LED Light-emitting diode. 53

MAR Memory Address Register. 12, 13, 15

MBR Memory Buffer Register. 12, 13, 15

MIC Micro Instruction Counter. 13, 15

MIR Micro Instruction Register. 13, 15

MOSFET metal–oxide–semiconductor field-effect transistor. 6–8, 18–20, 40–43, 45–48, 55, 82, 109, 111

MROM mask ROM. 19

NVRAM non-volatile random-access memory. 17, 20

PoR power-on reset. 20

PROM programmable ROM. 19

RISC Reduced Instruction Set Computer. 21

ROM read-only memory. 17, 19, 20

SPICE Simulation Program with Integrated Circuit Emphasis. 3, 33, 39, 43, 77

SRAM static random-access memory. 17, 18, 20

SWD Serial Wire Debug. 24

UART Universal Asynchronous Receiver Transmitter. 54

Bibliography

[1] Rob van der Meulen. Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015, November 2015. accessed 26 September 2017, http://www.gartner.com/newsroom/id/3165317.

[2] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[3] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. *On the Importance of Checking Cryptographic Protocols for Faults*, pages 37–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[4] E. DeBusschere and M. McCambridge. Modern game console exploitation. Technical report, Department of Computer Science, University of Arizona, 2012.

[5] Riscure. VC Glitcher Datasheet. accessed 26 September 2017, https://www.riscure.com/uploads/2017/07/datasheet_vcglitcher.pdf.

[6] Riscure. Glitch Amplifier Datasheet. accessed 26 September 2017, https://www.riscure.com/uploads/2017/07/datasheet_glitchamplifier.pdf.

[7] Colin O'Flynn and Zhizhang (David) Chen. *ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research*, pages 243–260. Springer International Publishing, Cham, 2014.

[8] Sergei P. Skorobogatov. Copy Protection in Modern Microcontrollers. Technical report, Security Group, Computer Laboratory, University of Cambridge, 2000. accessed 26 September 2017, http://www.cl.cam.ac.uk/~sps32/mcu_lock.html.

[9] Rafael Boix Carpi, Stjepan Picek, Lejla Batina, Federico Menarini, Domagoj Jakobovic, and Marin Golub. *Glitch It If You Can: Parameter Search Strategies for Successful Fault Injection*, pages 236–252. Springer International Publishing, Cham, 2014.

[10] L. Zussa, J. M. Dutertre, J. Clediere, and B. Robisson. Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 130–135, May 2014.

[11] Stan Augarten. *The Most Widely Used Computer on a Chip: The TMS 1000*. Houghton Mifflin, 1983.

[12] Atmel. ATtiny4 / ATtiny5 / ATtiny9 / ATtiny10 Datasheet. accessed 10 October 2017, `https://www.microchip.com/wwwproducts/en/ATtiny4`.

[13] Leonhard Stiny. *Aktive elektronische Bauelemente: Aufbau, Struktur, Wirkungsweise, Eigenschaften und praktischer Einsatz diskreter und integrierter Halbleiter-Bauteile*. Springer Fachmedien Wiesbaden, 2016.

[14] Klaus Wüst. *Mikroprozessortechnik: Grundlagen, Architekturen, Schaltungstechnik und Betrieb von Mikroprozessoren und Mikrocontrollern*. Vieweg+Teubner Verlag, 2011.

[15] D. Kahng and S. M. Sze. A Floating Gate and Its Application to Memory Devices. *Bell System Technical Journal*, 46(6):1288–1295, 1967.

[16] Gerhard Helge Schildt, Daniela Kahn, Christopher Kruegel, and Christian Moerz. *Einführung in die Technische Informatik*. Springer Vienna, 2005.

[17] Brian Matas and Christian DeSubercausau. *Memory, 1997: Complete Coverage of DRAM, Sram, EPROM, and Flash Memory IC's*. Integrated Circuit Engineering Corp., Scottsdale, AZ, USA, 1997.

[18] Liu Jiang, Wang Xueqiang, ge Qin, Wu Dong, Zhang Zhigang, Pan Liyang, and Liu Ming. A low-voltage sense amplifier for high-performance embedded flash memory. 31, 11 2010.

[19] Inductiveload. SRAM Cell (6 Transistors), January 2009. accessed 09 February 2018, `https://commons.wikimedia.org/wiki/File:SRAM_Cell_(6_Transistors).svg`.

[20] J.M. Daga, C Papaix, M Merandat, Stephane Ricard, G Medulla, J Guichaoua, and D Auvergne. Design Techniques for EEPROMs Embedded in Portable Systems on Chips. 20:68 – 75, 02 2003.

[21] IEEE. IEEE Standard Test Access Port and Boundary Scan Architecture. *IEEE Std 1149.1-2001*, pages 1–212, July 2001.

[22] ARM. ARM Debug Interface Architecture Specification ADIv5.0 to ADIv5.2. accessed 12 January 2018, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0031c/index.html.

[23] Atmel. AVR042: AVR Hardware Design Considerations. accessed 12 January 2018, http://ww1.microchip.com/downloads/en/appnotes/atmel-2521-avr-hardware-design-considerations_applicationnote_avr042.pdf.

[24] Timothy C. May and Murray H. Woods. A New Physical Mechanism for Soft Errors in Dynamic Memories. In *16th International Reliability Physics Symposium*, pages 33–40, April 1978.

[25] Oliver Kömmerling and Markus G. Kuhn. Design Principles for Tamper-resistant Smartcard Processors. In *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, WOST'99, pages 2–2, Berkeley, CA, USA, 1999. USENIX Association.

[26] Ross Anderson and Markus Kuhn. Tamper Resistance: A Cautionary Note. In *Proceedings of the 2Nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, WOEC'96, pages 1–1, Berkeley, CA, USA, 1996. USENIX Association.

[27] Ross J. Anderson and Markus G. Kuhn. Low Cost Attacks on Tamper Resistant Devices. In *Proceedings of the 5th International Workshop on Security Protocols*, pages 125–136, London, UK, UK, 1998. Springer-Verlag.

[28] J. Balasch, B. Gierlichs, and I. Verbauwhede. An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 105–114, Sept 2011.

[29] Hamid Choukri and Michael Tunstall. *Round Reduction Using Faults*, pages 13–24. Fault Diagnosis and Tolerance in Cryptography – FDTC 2005, 2005.

[30] P. Tummeltshammer and A. Steininger. On the role of the power supply as an entry for common cause faults. In *12th International Symposium on Design and Diagnostics of Electronic Circuits Systems*, pages 152–157, April 2009.

[31] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache. Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, Nov 2012.

[32] A. Djellid-Ouar, G. Cathebras, and F. Bancel. Supply voltage glitches effects on CMOS circuits. In *International Conference on Design and Test of Integrated Systems in Nanoscale Technology, 2006. DTIS 2006.*, pages 257–261, Sept 2006.

[33] L. Zussa, J. M. Dutertre, J. Clédière, and A. Tria. Power supply glitch induced faults on FPGA: An in-depth analysis of the injection mechanism. In *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*, pages 110–115, July 2013.

[34] NewAE Technology Inc. Tutorial A3 VCC Glitch Attacks. accessed 15 December 2017, https://wiki.newae.com/Tutorial_A3_VCC_Glitch_Attacks.

[35] T. Korak and M. Hoefler. On the Effects of Clock and Power Supply Tampering on Two Microcontroller Platforms. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 8–17, Sept 2014.

[36] K. Gomina, J. B. Rigaud, P. Gendrier, P. Candelier, and A. Tria. Power supply glitch attacks: Design and evaluation of detection circuits. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 136–141, May 2014.

[37] Infineon. IRF7821 N-Channel HEXFET Power MOSFET Datasheet. accessed 15 December 2017, http://www.infineon.com/dgdl/irf7821pbf.pdf?fileId=5546d462533600a401535608d7f31d06.

[38] JEDEC. Interface Standard for Nominal 3 V/3.3 V Supply Digital Integrated Circuits. *JESD8C.01*, 2007.

[39] Linear Technology. Micropower High Side MOSFET Drivers. *Application Note 53*, 1993. accessed 16 January 2018, http://cds.linear.com/docs/en/application-note/an53.pdf.

[40] Infineon. BSD235C OptiMOS2 + OptiMOS-P 2 Small Signal Transistor. accessed 22 December 2017, http://www.infineon.com/dgdl/Infineon-BSD235C-DS-v02_04-EN.pdf?fileId=db3a30433580b371013585a2d0d53326.

[41] Linear Technology. LTC1693-5 High Speed SingleP-Channel MOSFET Driver Datasheet. accessed 22 December 2017, http://cds.linear.com/docs/en/datasheet/16935f.pdf.

[42] Linear Technology. LTC1693-3 High Speed Single/DualN-Channel MOSFET Driver Datasheet. accessed 22 December 2017, http://cds.linear.com/docs/en/datasheet/1693fa.pdf.

[43] Texas Instruments. LM5134 Single7.6-A Peak Current Low-SideGate Driver With a PILOT Output. accessed 22 December 2016, http://www.ti.com/lit/ds/symlink/lm5134.pdf.

[44] Analog Devices. ADuM 1100 Datasheet. accessed 31 January 2018, http://www.analog.com/media/en/technical-documentation/data-sheets/ADUM1100.pdf.

[45] P. Horowitz and W. Hill. *The Art of Electronics*. Cambridge University Press, 2015.

[46] Edwin Hewitt and Robert E. Hewitt. The Gibbs-Wilbraham phenomenon: An episode in fourier analysis. *Archive for History of Exact Sciences*, 21(2):129–160, Jun 1979.

[47] Atmel. ATmega328P Datasheet, October 2016. accessed 11 February 2018, http://www.microchip.com/wwwproducts/en/ATmega328p.

[48] Arduino. Arduino Uno Rev3, 2018. accessed 11 February 2018, https://store.arduino.cc/arduino-uno-rev3.

[49] Flawed. Glitching 101: modifying code execution paths using only voltage, January 2017. accessed 11 February 2018, https://flawed.net.nz/2017/01/29/.

[50] NewAE Technology Inc. CW308T-AVR Target, October 2017. accessed 11 February 2018, https://wiki.newae.com/CW308T-AVR.

[51] NewAE Technology Inc. CW301 Multi-Target, March 2017. accessed 11 February 2018, https://wiki.newae.com/CW301_Multi-Target.

[52] Brett Giller. Implementing Practical Electrical Glitching Attacks. Black Hat Europe 2015, November 2015. accessed 11 February 2018, https://www.blackhat.com/docs/eu-15/materials/eu-15-Giller-Implementing-Electrical-Glitching-Attacks.pdf.

[53] M. Popovich, E. G. Friedman, R. M. Secareanu, and O. L. Hartin. Efficient placement of distributed on-chip decoupling capacitors in nanoscale ICs. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 811–816, Nov 2007.

[54] T. Charania, A. Opal, and M. Sachdev. Analysis and Design of On-Chip Decoupling Capacitors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(4):648–658, April 2013.