Dipl.-Ing. Johannes Grinschgl

Acceleration of System-on-Chip Fault Attack Emulation

Dissertation

vorgelegt an der
Technischen Universität Graz

zur Erlangung des akademischen Grades
Doktor der Technischen Wissenschaften
(Dr. techn.)

durchgeführt am Institut für Technische Informatik
Technische Universität Graz
Vorstand: Em. O. Univ.-Prof. Dipl.-Ing. Dr. techn. Reinhold Weiß

Graz, im Januar 2013

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ....

(Unterschrift)

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....

date (signature)

Kurzfassung

In den letzten Jahren hat sich die Komplexität von System-on-Chips (SoC), ähnlich dem Gesetz nach Moore, exponentiell erhöht. Dies führt dazu, dass der Bedarf an neuen Systemen zur Verifikation bzw. Evaluation der Sicherheitsmaßnahmen im SoC deutlich angestiegen ist und es aktuell zahlreiche Bestrebungen zur Entwicklung neuer Fehlerinjektionsverfahren gibt. Eine dieser Methoden stellt die sogenannte „Emulation von Fehlerattacken“ dar, welche im Vergleich zu den anderen Verfahrensweisen eine Überprüfung von vielen unterschiedlichen Fehlerangriffen binnen kürzester Zeit ermöglicht. In diesem Zusammenhang spielt neben einer verbesserten Geschwindigkeit der Fehleremulation auch eine Reduktion der Anzahl der Fehlertests auf ein Minimum eine besondere Rolle.

In der vorliegenden Dissertation wird eine neu entwickelte Methode zur Emulation von Fehlerattacken präsentiert, welche auf einer Verbesserung der Fehlerinjektionsmethodik, Fehlerinjektionsstrategien sowie Auswertung der Fehlereffekte aufbaut und zur Beschleunigung der emulations-basierten Sicherheitsverifikation von komplexen SoCs dient. In dieser Arbeit wird ein spezieller modularer Fehlerinjektor zur gezielten Einbringung von Fehlern verwendet, der über die Ansteuerung von sogenannten „Saboteuren“ zu einer gezielten Manipulation der Signale im Design führt. Bevor diese Saboteure verwendet werden können, müssen sie in die Hardwarebeschreibung des SoC über eine entsprechende automatisierte Modifizierung integriert werden. Außerdem wurde eine neue Strategie zur Beschleunigung des Fehlerinjektionsverfahrens entwickelt, welche mit Hilfe einer automatisierten Analyse eines sogenannten „Golden Model Run“ eine gezielte Einbringung von Fehlern ausschließlich in die aktiven Regionen am SoC zur Folge hat, wodurch die Resistenz gegen Fehlerattacken eines ganzen SoC im Vergleich zu herkömmlichen Fehlerinjektionsverfahren innerhalb einer viel kürzeren Zeitspanne überprüft werden kann. Nach Ablauf der erwähnten Fehlerinjektionstests erfolgt anhand einer Überprüfung von Ein- und Ausgängen, Kontrollfluss sowie nichtflüchtigem Speicher des Systems die Evaluation der erzielten Effekte auf das System direkt auf der Emulationsplattform mit Hervorhebung der sicherheitsrelevanten Fehleremulationsdurchläufe.

Abstract

Over the years the complexity of system-on-chips (SoCs) has increased in a speed similar to the defined law by Moore. Especially for complex security SoCs, as the effort to verify their security against fault attacks is a difficult process. Therefore, many different fault testing methods have been developed over the years to cope with this problem. One of these methods is fault emulation. This method is a very fast method compared to other fault injection methods. The disadvantage of this method is that the hardware description of the device under test has to be modified, and it is harder to analyze what effect an attack produces in the system. It is not only the speed of the fault testing approach, but also the test strategies that are important. The fastest test method without proper test strategies cannot handle complex SoCs because of the high number of possible fault locations.

In this thesis a novel method is presented on how to accelerate emulation based security testing of complex SoCs, especially for operating system development, by an improved fault emulation method together with an improved fault injection test scenario. The concept of this fault injection method is based on the emulation of fault effects, because for operating system testing it is not important how a fault is produced, it only matters if the operating system can cope with the fault effect. To emulate such effects a flexible and modular fault injector is used which controls saboteurs. These saboteurs manipulate the system's behavior to produce fault effects which can be used to test the security features of the operating system. The placement of the modular fault injection controller and the saboteurs is a time consuming and error prone process which is replaced by an automatic placement method. Because of the complexity of the SoCs it is not possible to test every fault. Therefore, several improvements to the fault injection campaigns have to be developed. These improvements can be split into two main groups. The reduction of the required fault injection campaigns and the automatic analysis of what effect an attack has on the system. The reduction of the fault injection campaigns is based on an analysis of the system during a golden model run to detect security critical regions in the software and the hardware. The post injection analysis is a step which is especially required to accelerate the evaluation by the test engineer to highlight critical attacks.

Acknowledgements

First of all I would thank my parents for their great support over the last years. Without their help it would not of been possible to finish this thesis.

Foremost, I would like to thank Professor Reinhold Weiss for the possibility to write my thesis on the Institute for Technical Informatics. Furthermore, I appreciate his advice and thoughtful comments to compose and finish this thesis.

This whole thesis would not of been possible if the Power-Modes Project was not founded by the Austrian Federal Ministry of Transport, Innovation and Technology. Funding alone is not enough to get a project running. Therefore, a special thanks goes to Christian Steger, Josef Haid and Holger Bock for their great project management and helpful support over the last few years. I also want to thank the project partners of the Power-Modes Project, Infineon Technologies Austria AG and Austria Card.

A special thanks goes to my project partner Armin Krieg for the interesting discussions and our good cooperation during the whole Power-Modes project. Also my working colleagues Manuel Menghin, Norbert Druml, Christian Bachmann and Andreas Genser must be honored for their constructive comments about my thesis. For the great support over the last years I would also like to thank Silvia Reiter and Engelbert Meissl.

Finally I would like to thank my girlfriend Sandra for her love and support during the whole Ph.D.

Graz/Austria
January, 2013

Johannes Grinschgl

Extended Abstract

The complexity of system-on-chips (SoCs) has been increased in a speed similar to Moore's law. In some publications it is already spoken of as an increase more-than-moore. This increase is reached by an increasing level of integration and decreasing size of circuit elements. This small size makes the system more vulnerable against external influences such as: cosmic rays, light and radiation. Because of recent chip technologies (e.g. 20nm technology) the chance of multi-bit faults caused by natural influences (e.g. cosmic rays) is increasing. On the other hand the equipment of attackers has improved over the years. The attackers often use multi-bit attacks and multi-attack scenarios to make the system perform unintended operations. To increase the security level of a system the SoC has to be verified against such external influences. To verify a system against multi-bit fault attacks, requires much more verification effort than a simple single event upset (SEU) verification strategy. The additional verification effort depends on the used verification model which defines which signals are attacked in parallel. For dependability verification it can be assumed that bits with a near location on the silicon are disturbed simultaneously. In security applications this assumption is often not true, because an attacker can use multiple attacks to manipulate the system at different locations. The verification can be done using three different methods. The simulation method can be performed very early on during the design of a SoC, and is also the most flexible and easiest observable method. The main disadvantage is that this method is also the slowest method. The second method uses physical tests which only can be done very late in the SoC development process because a physical chip must exist. If a fault was detected during this stage a redesign of the chip is required. The last method is emulation were the functionality of the SoC is mapped on an FPGA. This method is a really fast method for fault testing because the system

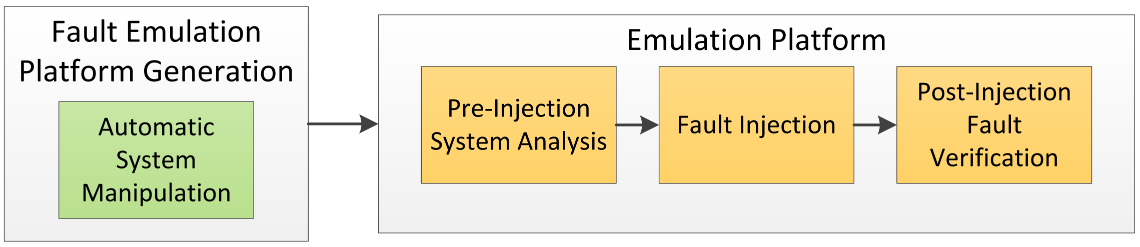

Fault Emulation Platform Generation
Automatic System Manipulation
Emulation Platform
Pre-Injection System Analysis
Fault Injection
Post-Injection Fault Verification

Figure 1: Concept of the novel automated fault injection method. This method is based on two parts: the generation of the injection platform, and the fault injection process. The generation of the emulation platform is based on automatic system manipulation. The injection process can be split into three categories: Pre-Injection System Analysis, Fault Injection and Post- Injection Fault Verification.

has a similar performance to the produced chip. The main disadvantages are that for the fault emulation process the system has to be modified. None of these methods can verify a whole system in a feasible time without proper verification strategies. The increase of SoC complexity increases the verification gap between the possible attack targets and the number of testable fault attacks, because the increase of the SoC complexity is growing faster than the verification methods and verification strategies. Therefore, methods have to be developed to accelerate fault verification and to decrease the amount of required fault verification to an absolute minimum without leaving out important tests.

In this thesis a novel method for accelerating fault attack verification is presented. The acceleration of fault attack verification is done using several different steps. These steps can be split into three main groups, automatic code adaption, pre-injection system analysis and post-injection fault effect analysis. This thesis is part of the Power-Modes[1] Project which focuses on fast fault emulation and power emulation for smart card developments. The method is based on a modular fault injector[2] (MFI) which is based on an in-system fault injection strategy. This strategy abstracts the internal location of the fault by using an automatic test pattern injection. This abstraction allows for the emulation of different fault effects without exact knowledge of the internal structure. This is especially important for software developers who do not know exactly how the hardware is build. The controller has a general interface that allows the use of MFI on most emulation platforms. To disturb the normal behavior of the system the MFI also controls saboteurs, which are small blocks which are placed between the source and the sink of a signal. With the saboteurs it is possible to reproduce different fault models e.g., stuck-at faults, delays, bit-flips and so on.[3] This MFI and the saboteurs have to be added into the VHDL description of the SoC. Doing this manually is an error prone and time consuming process. To reduce the effort of adding the required code for the fault injection into the system, an automatic placement tool was developed. This tool analyses the structure of the SoC with the VMAGIC tool and modifies the system depending on the attack targets[4].

After the SoC is modified the fault verification method can be used for security verification. This method emulates fault effects to verify if the software security features fulfill the requirements. The reason only fault effects are emulated is because for software verification it is not important how a fault is produced, it only matters what effect a fault has on the software. For example the manipulation of a memory cell can be done in many different ways (write access attack, direct memory cell attack), but the software only notices that the memory is attacked. With this limitation the number of required fault attacks can be reduced[5]. The best fault injection strategy needed for verifying a

[1]POWer EmulatoR and MOdel based DEpendability and Security evaluation platform. Project Partners are Infineon Technologies Austria AG, Austria Card Plastikkarten und Ausweissysteme GmbH and TU Graz. The project is funded by the Austrian Federal Ministry for Technology under the FIT-IT contract FFG 825749

[2]*Modular Fault Injection for Multiple Fault Dependability and Security Evaluation*, 14th Euromicro Conference on Digital System Design, Oulu, Finland, 2011

[3]*Case Study on Multiple Fault Dependability and Security Evaluation*, Elsevier Microprocessors and Microsystems, in press

[4]*Automatic Saboteur Placement for Emulation-Based Multi-Bit Fault Injection*, 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip, Montpellier, France, 2011

[5]*POWER-MODES- POWer EmulatoR and MOdel based Dependability and Security evaluation*, ACM Transactions on Reconfigurable Technology and Systems, Volume 5, Issue 4, Article 19

large SoC in a reasonable time are proper test scenarios. Therefore, a new pre-injection system analysis method is needed to be developed. This analysis should show security critical regions in the system in such a way that fault injection campaigns can be reduced to an absolute minimum. Therefore, the activity of memory accesses and activity of different modules are analyzed. This information is generated during a golden-model-run on the emulation platform. To reduce the time overhead of sending this information back to the environment this information is analyzed directly on the FPGA[6]. Another step which reduces the number of fault injection scenarios is to find reasonable attack patterns on how the memory should be attacked. Therefore, the locality of memory elements on the silicon is analyzed. If a memory is attacked by an attacker it is very likely that the memory cell nearby is also disturbed. Therefore, a design analysis tool was implemented which highlights nearby memory cells. With this method it is possible to generate proper test patterns for memory attacks or register attack scenarios[7]. The pre-injection system analysis reduces the required fault injection campaigns but there are still millions of fault injection campaigns required to verify the security of the SoC against fault attacks. For a verification engineer it is nearly impossible to check all results by hand. Therefore, a post-fault injection effect analysis was developed. This method already checks on the FPGA what effect fault attacks have on a specific system. This effect can be: manipulation of output, program flow manipulation, memory changes and so on[8]. Fault attacks can also have an effect on the power-profile of a system on chip. To give the verification engineer a hint what effect an attack has on the power profile an emulation based power analysis tool was integrated into the post fault injection effect analysis tool[9]. In future work an automatic method for side-channel analysis should be improved. This is required because novel high-order fault attacks try to manipulate the system in such a way that it leaks additional side-channel information, like timing and power. Therefore, it would be necessary to analyze the timing and the power leakage of the SoC with the whole fault emulation process.

[6] *Efficient Fault Emulation using Automatic Pre-Injection Memory Access Analysis*, 25th IEEE System-on-Chip Conference, Niagara Falls, USA, 2012

[7] *Acceleration of Fault Attack Emulation by Consideration of Fault Propagation*, International Conference on Field-Programmable Technology, Seoul, Korea, 2012

[8] *Efficient Fault Emulation based on Post-Injection Fault Effect Analysis (PIFEA)*, 55th International Midwest Symposium on Circuits and Systems, Boise, USA, 2012

[9] *Characterization and Handling of Low-Cost Micro-Architectural Signatures in MPSoCs*, IEEE 25th European Test Symposium, Annecy, France, 2012

Contents

List of Figures

List of Tables

Glossary

Fault
A *fault* is a broken or manipulated part of a system which can lead to an error. A example of a fault is that one signal in the chip is stuck at zero and cannot switch back to one. For example, such faults can be caused by production problem, external influences or aging.

Error
An *error* is a unintended behavior of a system. An example of an error is when the system has an output different to the expected result. An error can be caused by a fault.

Emulation
An *emulation* is defined by [13] "to copy something achieved by someone else and try to do it as well as they have". In the case of this thesis it means to map the functionality of a chip on a FPGA platform in such a way that the digital parts act like a real chip.

Fault Attack
A *fault attack* is a intentional manipulation of the SoC, with the aim to produce an error in the system which allows an attacker to extract security critical information.

Fault Injection Campaign
A *fault injection campaign* uses multiple fault injections, which are required to test the security of a specific part of a system-under-verification. For example a fault injection campaign verifying if a AES core is vulnerable against fault attacks, consists of multiple fault injection runs where the AES core is attacked.

List of Abbreviations

ALU	Arithmetic Logic Unit
ASM	Assembler
BRAM	Block random-access memory
CPU	Central Processing Unit
DUT	Device-under-Test
EAL	Evaluation Assurance Level
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
LUT	Look-up-Tables
MEU	Multiple Event Upset
MFI	Modular Fault Injector
NVM	Non Volatile Memory
PC	Personal Computer
PIFEA	Post Injection Fault Effect Analysis
Power-Modes	POWer EmulatoR and MOdel based DEpendability and Security evaluation platform
RFID	Radio-frequency identification
SEU	Single Event Upset
SoC	System-on-Chip
SRAM	Static random-access memory
SuV	System-under-Verification
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Chapter 1

Introduction

Worldwide the number of system-on-chips (SoC) is rapidly growing. Not only are the number of SoCs growing, but also the complexity is rising. As seen in Figure 1.1 the complexity of portable devices is increasing with each year in an exponential way. Not only are the logic size and memory size expected to increase but also the number of processing engines as well. In some publications there is already talk about an increase of speed more than moore [14, 15]. This increase in complexity must be handleable by development processes and test concepts. The test concepts have to make sure the functionality of the device matches the predefined requirements. Especially in safety and security domains as the requirements have to be examined with particular attention. If these requirements are not fulfilled this can lead to enormous costs for the company, loss of trust in the products of this company and in and in the safety environment if the product poses a danger when used.

1.1 Smart Cards

Smart cards are chip cards which have to fulfill higher security requirements than RFID cards [2]. As seen in Figure 1.2 the difference between RFID cards and smart cards is the complexity and their security features. This additional security level and complexity leads to higher costs in development, because of this financial effort Smart Cards are used only if the application requires the additional effort. Some typical areas where they an used are [16] Enterprise ID, Financial, Government, Healthcare, Identity, Telecommunications, Pay-TV and Transportation.

The number of smart card chips has grown over the last few years. From 2010 to 2012 an increase from 5520 million to 6925 million was forecast [3] (see Figure 1.3). This is an increase of 25,5 percent over 2 years. The trend shows that this increase will continue over the next few years. This increase of usage will also increase the number of attackers who will try to manipulate the smart cards to get an advantage out of it. Depending on the application it is worthwhile for the attacker to invest different amounts of money to attack the smart card. Because the cost of the equipment for such attacks is going down and the knowledge of the attackers is increasing, the smart card systems have to be improved to cope with the attackers. Therefore, new countermeasures against attacks have to be implemented and tested.

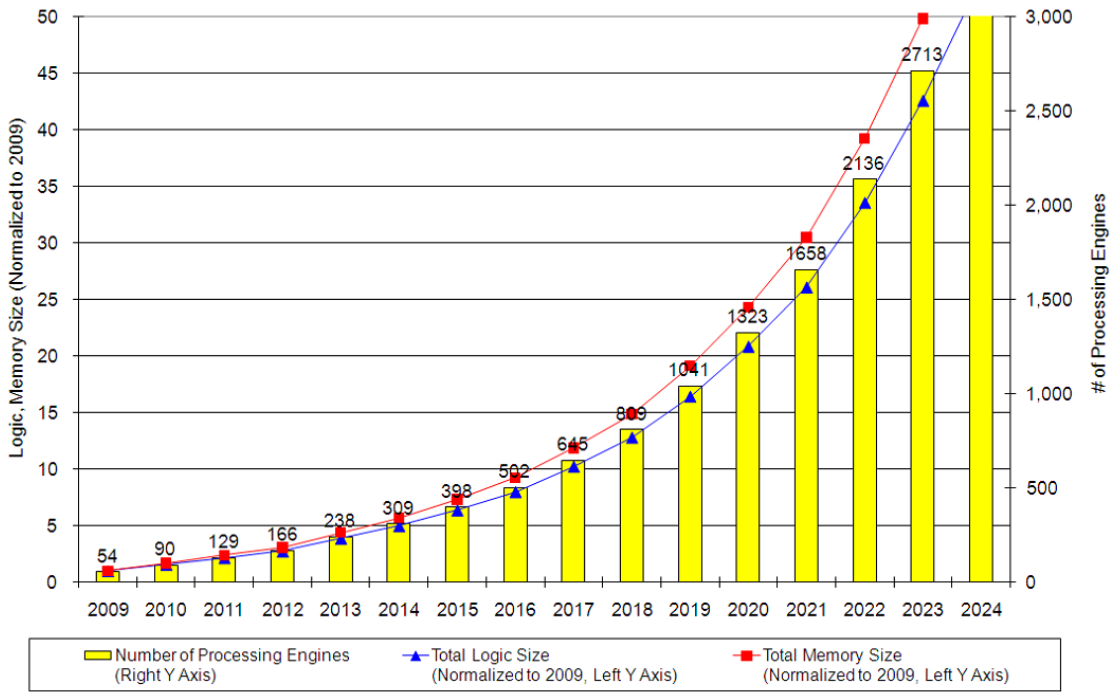


Figure 1.1: This trend shows how the complexity of portable devices is increasing over the next few years. Not only is the logic and memory size increasing, but also the number of processing engines following the exponential trend [1].

1.2 The Importance of Security Testing

Smart Cards are used in many security critical environments for example in banking, pay-TV and passports. To proof the security level of such smart cards, the smart cards have to be evaluated by a laboratory in order to achieve certification by a certification authority. The certification is often done following the common criteria standard. This evaluation can only be done by special laboratories. They test the security of the smart card against different side channel attacks and fault attacks. In Figure 1.4 [4] it can be seen that an EAL4 certification cost approximately 300.000 dollars. These costs can increase if faults are found, because such faults often require a redesign of the chip or the software. Due to of this high cost the companies want these problems to be resolved before these certification tests are done on the real chip, to test the system against fault attacks whilst in the development stage. These tests can be split into two main groups. The first one is the random test where the system is attacked at random times and locations to check if an attack has any effect on the system. The second method is to attack the system at a specific location because a developer thinks that there might be a weak point in the design. These two methods vary in the method used for the fault injection. The second method can be best tested with a simulation with exactly knowing the parameters of the expected attack and in this case a slow simulation time has no bearing. For random tests this simulation is often too slow. Therefore, the fault emulation is a proper method.

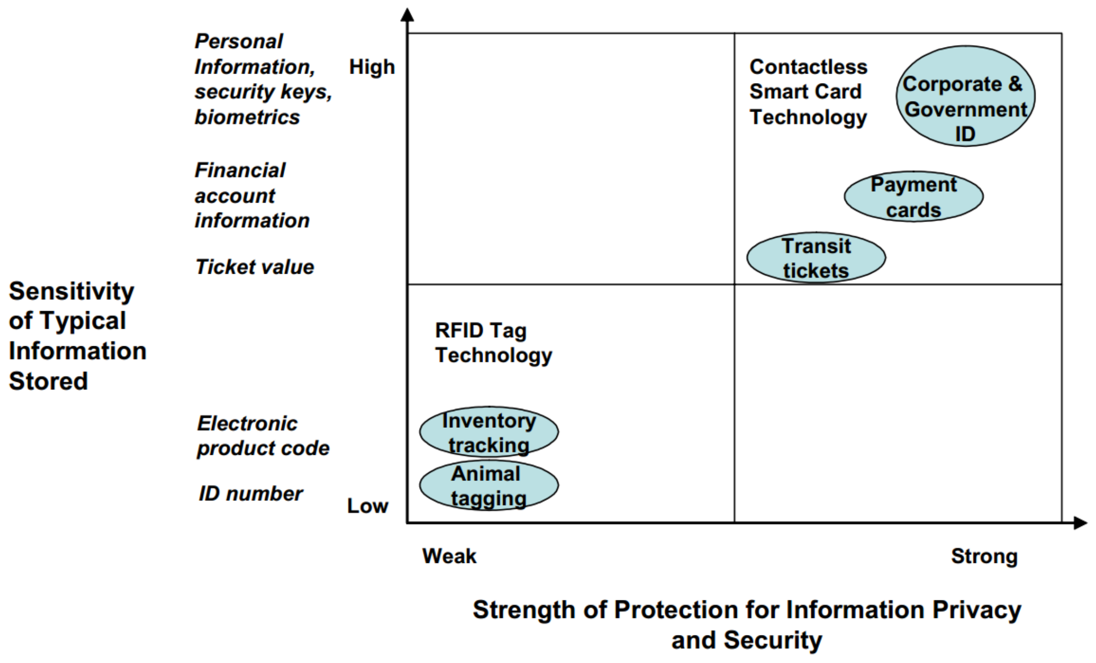

Personal Information, security keys, biometrics
Financial account information
Ticket value
Electronic product code
ID number
High
Low
Sensitivity of Typical Information Stored
Contactless Smart Card Technology
Corporate & Government ID
Payment cards
Transit tickets
RFID Tag Technology
Inventory tracking
Animal tagging
Weak
Strong
Strength of Protection for Information Privacy and Security

Figure 1.2: This figure shows the difference of RFID cards and smart cards [2].

1.3 Smart Fault Injection Method

The smart fault injection method, presented in this thesis, has a goal to improve the commonly used fault injection methods used to accelerate fault attack testing. This can be done by reducing the required time for one fault injection campaigns, by reducing the effort for the verification engineer and by reducing the number of fault injection campaigns. Before the novel fault injection method can be explained, the questions must be asked about the need to accelerate the fault injection process.

1.3.1 Problem Statements

- **Increasing complexity of SoC:** The complexity of SoC has increased extremely over the last few years. This lead to a high test effort for security testing.
- **Increasing number of Smart Cards:** The number of smart cards has increased over the last two years more than 25 percent. This growth in the number of smart cards, increases the risk that the user might have a smart card that can be hacked.
- **Increasing security requirements:** The number for security requirements of smart cards is increasing because the usage of smart cards becomes more common in different environments. Especially, for passport security as it is extremely important that it is not possible to hack the smart card chip, because they store biometric data.
- **Increasing test effort:** The effort to test such smart cards is also increasing because they are becoming more complex and the security and encryption is improving.

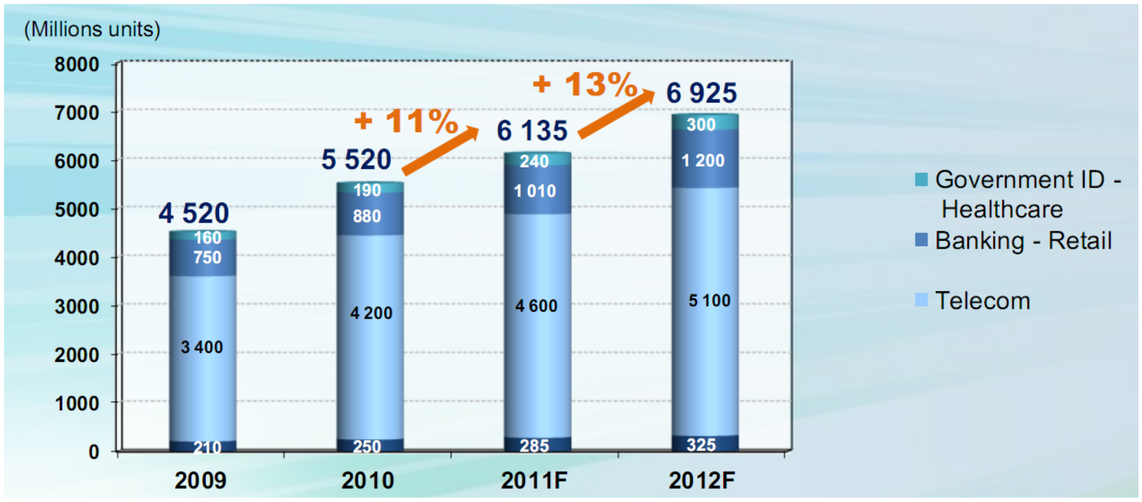


Figure 1.3: The number of smarts cards sold per year worldwide goes into the billions. This number is increasing yearly by a double digit percentage value [3].

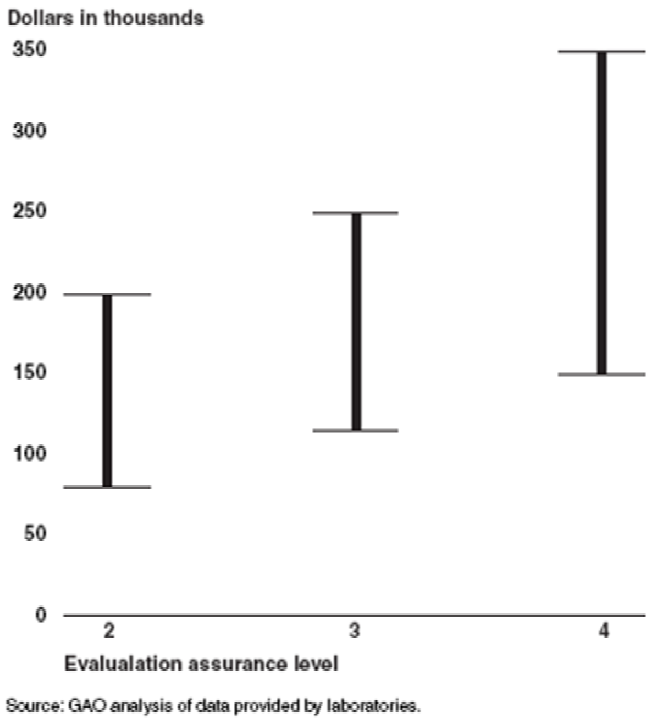


Figure 1.4: This figure shows the cost of the common criteria certification process of smart cards depending on the Evaluation assurance level (EAL) [4]

- **High cost of certification:** The certification process cost rises depending on the complexity and the security level.

1.3.2 Power-Modes Project

This thesis is part of the Power-Modes project (POWer EmulatoR and MOdel based DEpendability and Security evaluation platform)[1]. The goal of this project is to design and implement a fault emulation platform which can be used for dependability and security evaluation by including power information.

1.4 Thesis Structure

The structure of this thesis is shown in this section. At the beginning of this thesis an introduction into the field of fault injection and smart card security is given in Chapter 1. After the background is explained the most important related works are discussed in Chapter 2. Chapter 3 shows the novel approach about how the fault injection method can be accelerated by using the smart fault injection method. The performance of this method is shown in Chapter 4. Chapter 5 shows the conclusion and the future work. Finally the publications which describe this method in more detail are shown in Chapter 6.

[1]Project Partners are Infineon Technologies Austria AG, Austria Card Plastikkarten und Ausweissysteme GmbH and Institute of Technical Informatics. The project is funded by the Austrian Federal Ministry for Technology under the FIT-IT contract FFG 825749

Chapter 2

Related Work

The testing of SoC's is a time consuming process. Especially for fault attack testing the time to test if a system is secured against attacks on the hardware is enormous. This time is also increasing exponentially to the complexity of the system-under-verification (SuV) [17, 18, 19]. Therefore, a large amount of research is done on how to decrease the test time. This research can be split into the following three main topics:

- improving the fault injection method
- reducing the required fault injection campaigns
- automatizing the analysis after a fault injection is performed

2.1 Improving the Fault Injection Method

The fault injection methods can be split into the following three main methods:

- **Fault Simulation:** The fault simulation approach is the most common method for fault testing. Compared to other fault testing methods the time required for verifying a SuV is much higher [20]. This slower fault testing time would be compensated by the flexibility of this approach. In [21] an early example of this approach it is shown where simple simulation commands are used for a full fault injection simulation.

  An example for a high-level fault injection simulation is the paper from Rothbart et al., where faults are injected into a SystemC model of the design [22]. An overview of the most common fault injection simulation methods are given in [23].

  Novel fault simulation approaches allow the simulation of SoCs which are written in different hardware description languages. A good example for such an approach is [24] where a system is tested and designed in SystemC and VHDL.

- **Physical Fault Tests:** This approach uses physical manipulations of a real chip to check if the system can resist fault attacks [17]. Therefore, a good knowledge about the design is required. Also the verification engineer needs to perform the fault injection using special equipment [25] which could be very expensive. Another disadvantage is that if a fault is detected a redesign of the system is required which leads to high costs.

- **Fault Emulation:** To solve the problems of a long simulation time and the expensive physical tests, the fault emulation approach was developed. This approach maps the functionality of the chip on an FPGA. On this FPGA the behavior of the SoC can be emulated [26, 27, 28, 29]. To inject faults in this emulated chip three different methods are known.

  - **Mutants:** Mutants are manipulated blocks of the SoC which have the same functionality as the original block of the SoC until they are activated [30]. When they are activated they act as a manipulated block. The disadvantage of this module is that the effort to produce such a mutant is very high, because the functionality of this module has to be modified. Another disadvantage is that the hardware description of the system has to be manipulated. The big advantage of this mutant is that complex fault models can be reproduced because every manipulation to this system can be implemented.
  - **Saboteurs:** Saboteurs are blocks which are placed between the source and the sink of a signal. While they are not active they act as a transparent block. When they are activated they can manipulate the signal in many different ways (e.g. stuck-at faults, delays, ...) [30]. The disadvantages of this module is that they have to be placed into the hardware description of the SuV. This is a time consuming and error prone process. The advantage is that with saboteurs, fault effects can be injected into every signal in the design.
  - **Partial Reconfiguration:** The partial reconfiguration is a very new approach for fault injection campaigns. This approach uses the functionality of new SRAM FPGAs where it is possible to manipulate the look-up-tables (LUTs) of the FPGA while the system is running. For this approach many different publications have been written over the last few years [31, 32, 33, 34, 35]. The main disadvantage of this approach is that it only works with the new SRAM FPGAs and the tooling to use this fault injection method is not optimal.

Independent to how the faults are injected exist two different fault basic models on how faults can be injected. These models should be known because not every injection method support all the injection models. Therefore, some of the work is done by [36, 30] splitting the injection models into:

- **Permanent fault model**: Permanent faults are manipulations of the system which are active forever. Such permanent faults are often also called destructive faults because to produce them often parts of the chip are destroyed. These faults are easy to test with every one of these fault injection methods.

- **Transient fault model**: Transient faults are manipulations of the system which are only active for some amount of time. This fault model is often used for memory fault attack emulation, because an attack to the memory often only manipulates the content of a memory cell until the next write access.

2.2 Reducing the Required Fault Injection Campaigns

The requirements to reduce the number of fault injection campaigns has increased over the last few years because the complexity of SoC has increased. It is not possible to test every possible fault effect on the system. Therefore, the number of fault injection campaigns have to be reduced to the most critical fault attacks. The main problem is to find the critical fault attack targets.

In the literature two different ways to deal with the problem exist:

- **Statistical approaches:** Statistical approaches use mathematical methods to find worthy attack targets. The main problem with this approach is that after the fault injection the verification engineer does not know if all the critical regions were tested. Therefore, Leveugle et al. tried to solve this by generating a confidence quantifier for this approach [37, 38]. Their results show that it is possible with statistical methods to obtain very precise results about the robustness of a circuit. Therefore, only a limited number of fault injection is required.

- **System analysis approaches:** The target of this approach is to automatically find security critical regions in the design by analyzing the source code or by analyzing the activity of system components. The analysis of the source code can be done without running a golden model. In [39, 40] a method is presented on how the source is analyzed to find memory information. They extract from the ASM code every access to the memory and use this information do reduce the fault injection campaigns against the memory to the real write and read accesses. This reduces the required amount of fault injection campaigns against the memory.

  The approach presented in [41] works in a similar way but in this approach a golden model is used to generate the information required for reducing the number of required fault injection campaigns. They check every access to the registers and use this information to reduce the number of fault injections. This reduces the number of required fault injection campaigns to the registers to about 12 percent of the normal run.

2.2.1 Automatizing the Analysis after a Fault Injection is Performed

The analysis of the results is an important task to help the verification engineer find the successful attack within the millions of test scenarios. One method to solve this problem is shown in [42]. They uses failure mode effects analysis to calculate the effect of a fault to the system using a high level system model of the SoC. In [43] different fault models are explained and how it is possible to detect these faults using this fault model. The fault effects can be split into multiple types:

- **Correct:** This means the fault attack has no effect to the system.

- **Silent:** Silent means the fault attack had a effect on the system but the effect is not visible in the output of the system.

- **Latent:** A latent fault is a manipulation of the system which does not have an effect on the system at that moment, but it might produce a fault effect in the future.

- **Detected:** Detected faults are attacks which are already found by the system.
- **Failure:** Failure are faults which already had an effect on the output of the system.

2.3 Difference to Related Work

The differences between the related work and this thesis can be spit into three main parts:

- Automation of the system manipulation required for a saboteur based fault emulation process. In the literature no method is shown how the hardware description of a SuV can be manipulated automatically to generate a fault emulation system which can be used for long term fault testing.
- Using more activity information for reducing the fault injection campaigns to an absolute minimum. In this thesis additional information of the system are used to reduce the required fault injection campaigns. This information is about the memory access information and the power information.
- Analyzing the effect of the fault injection automatically on the emulation platform. In the literature the fault effect analysis is done mostly on the host PC, the difference between this work and the related work is to bring this directly onto the emulation platform to reduce the communication overhead between the emulation platform and the host PC.

2.4 Contributions and Significance

The contribution of this work is to accelerate emulation based fault resistance verification for smart card operating system development. This acceleration is required because without an improvement it is not possible to cope with the increasing complexity of system-on-chips (SoC), because the number of possible fault targets rises exponentially compared to the complexity of the SoC. To accelerate the fault verification it is a requirement to improve many different steps of the fault verification.

This starts by improving the fault injection method, which means how the faults are injected and how the required system manipulations are introduced into the system. In this thesis a new fault injection controller is presented to cope with this problem. Because the system manipulation is an error-prone and time-consuming process, a method on how such manipulation can be done automatically is presented.

The second step is to improve the fault verification flow by analyzing the system to find security critical regions in the design. Therefore, a new novel analysis method is presented which can analyze meaningful attack targets directly on the FPGA. With this method it is possible to reduce the required test campaigns by a factor between ten and one thousand.

The final step for accelerating the fault verification is to reduce the effort for the verification-engineer to detect what attack was a successful attack. This is required because an engineer cannot check millions of fault attack scenarios. Therefore, a novel analysis block was introduced to test this directly on the emulation platform if the control-flow, memory content or the communication to the environment are disturbed.

Chapter 3

Smart Fault Injection Method

In this section the smart fault injection method is summarized. The structure of this method can be split (as seen in Figure 1) into 4 working packages. Before the working packages have been specified the target of this whole concept has to be described. The target is to test the resistance of the system against fault attacks as fast as possible. Therefore, several intermediate steps are required. The whole work is based on the fault injection method shown in detail in Chapter 6.1

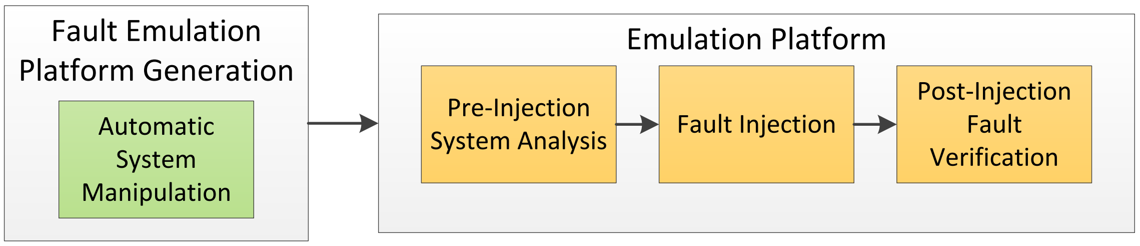

Fault Emulation Platform Generation
Automatic System Manipulation
Emulation Platform
Pre-Injection System Analysis
Fault Injection
Post-Injection Fault Verification

Figure 3.1: Overview of a smart fault emulation tool.

3.1 Challenges

The challenges this work has to cope with are described in the following block. The main aim behind these challenges is to accelerate the fault emulation, reducing the effort for the verification engineer and make the fault injection usable without exact knowledge about the internal structure of the SuV.

3.1.1 Smart Attack Scenarios for Fault Emulation

For fault testing we have an advantage against the attacker by using additional information of the system to find areas, time slots, memory access and so, we can find out where the system is vulnerable against fault attacks. A simple example is that an attack to a multiplier has only an effect to the system if the multiplication is used. This information can be used to reduce the fault emulation to the times where the multiplication is used. Also power information can be used to find when the system leaks power information, which can be used by attackers for differential power analysis.

3.1.2 Abstraction of the Attack Target for Fault Emulation

A big problem of common fault emulation tools is that the verification engineer must know exactly what signal should be attacked and at what time. But this level of knowledge is difficult in some situations. This fault injection technique should be used by software developers to test if their software is resistant against fault attacks before the physical test at a certification laboratory is done. The software developers do not and sometimes should not know which exact signal to disturb to produce a specific fault effect. For the software development it is only important to produce the fault effect to check if the software can handle this attack. Therefore, an abstraction of the attack targets is be realized.

3.1.3 Multi-bit Attack Scenarios

Because of the increasing knowledge of attackers and cheaper equipment the number of fault attacks has increased over the last few years. Also the number of known multi-bit attacks has increased. Multi-bit attacks are fault attacks where an attacker interferes with multiple regions of a chip during a single attack scenario. A simple example would be to firstly deactivate the fault detection unit and secondly to attack the security critical component.

3.1.4 Automatic System Manipulation

Fault emulation requires, if saboteurs are used as fault injection elements the adaption of the system with triggers, saboteurs and a control logic. This system manipulation is a time consuming and error-prone process. To cope with this problem an automatic system manipulation step was developed (see Chapter 6.2). This approach allows to place hundreds of saboteurs, the control logic, and the triggers automatically into the hardware description of the SuV. It is only required that a verification engineer specifies the location where the saboteurs should be placed.

3.1.5 Injection Campaign Generation based on System Information

Because of the large number of possible fault attack campaigns the number of these campaigns has to be reduced to an absolute minimum. This can be realized by analyzing the system information which is generated with a golden model run. With this method security critical regions of the design should be detected and so it is possible to attack the most important regions of the chip.

3.1.6 Fault Verification Directly on the FPGA

Verification engineers must verify if a fault injection campaign has manipulated a system in such a way that the secure information can be extracted from the SuV. This process is a time consuming and error-prone process which should be reduced by fault verification directly on the FPGA. This has the aim to reduce the effort for the verification engineer by analyzing the fault injection results. The analysis directly on the FPGA also reduces the communication overhead to the host PC which increases the overall speed of the verification process.

3.1.7 Standard Communication Interface

The standard communication interface is required because the system should be used on different emulation platforms.

3.2 Fault Injection Method

The fault injection method is the back bone of this thesis. This is the method where faults are injected into the system to manipulate its behavior (for a more detailed description see Chapter 6.1). These fault injection methods should fulfill the following requirements:

- **System independent:** The method should not be limited to a specific product or FPGA.
- **Flexible:** The concept should be expandable if new features should be added.
- **Multi-bit attack support:** Because multi-bit attack get more and more common the fault emulation tool should support such a feature.
- **Simple configuration:** The interface for the test environment should be as simple as possible to allow its methods to be used on multiple platforms.
- **Attack abstraction:** The attack abstraction is one key requirement of the MFI because for software development the software developer should not or maybe must not know what signal should be attacked. For them only the effect of the attack to the software is important.
- **Synchronize the fault injection with the SuV:** For fault attack it is important that the system can be attacked exactly at a specific time and location.
- **Repeatable:** The attacks should be repeatable.

To fulfill all these requirements a modular system is the choice to use. This system consists of a central core with a flexible interface where the fault injection and the trigger elements can be connected. This controller is called the Modular Fault Injector (MFI).

3.2.1 The Modular Fault Injector

The modular fault injector (MFI) is a system which consists (as seen in Figure 3.2) of multiple elements.

- **Control Logic:** The control logic is configured via the GPIO interface and activates the saboteurs depending on the trigger values and the defined attack.
- **Internal Fault Pattern Memory:** The memory stores the attack campaigns that should be performed.
- **Memory Control:** This block is a control logic which allows access to the Internal Fault Pattern Memory from the GPIO Interface.

- **GPIO Interface:** This interface is the connection to the environment. With this interface it is possible to configure the MFI and read out debug information.
- **Saboteur Interface:** The saboteur interface consists of two buses. The first bus is shared from every saboteur and defines which attack type the saboteurs should emulate. The second bus contains for every saboteur exactly one bit which defines if the saboteur should be activated or not.
- **Trigger Interface:** The trigger interface is an interface which is used to configure the trigger and to check if a trigger is active or not.

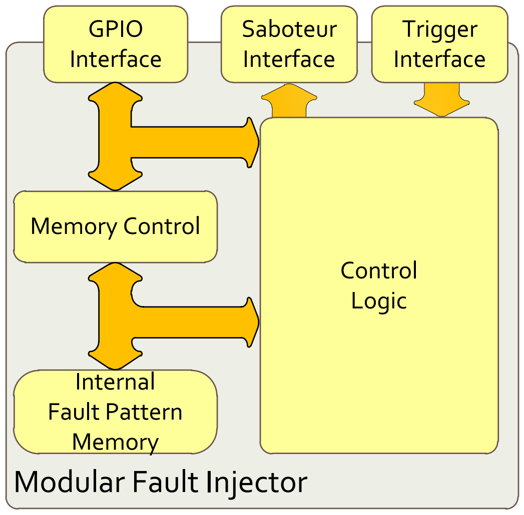


Figure 3.2: Overview of all components of the MFI with their interface to the environment, the saboteurs and the triggers (adapted from [5]).

3.2.2 Saboteur as Fault Injection Element

Saboteurs are small hardware blocks which are placed between the source and the sink of a signal. If not activated they do not influence the behavior of the signal. If they are activated they can manipulate the signal. The saboteurs are clock interdependent which means that there is nearly no delay between the activation signal and the manipulation of the manipulated signal. Only the delay generated by the combinatoric of the saboteur produces a delay. As seen in Figure 3.3 the saboteurs have several different control signals. The Activate signal is used to activate the fault injection (for a more detailed description see Chapter 6.3).

3.2.3 Synchronized Fault Injections and MFI with Triggers

Triggers are blocks which are used to synchronize the SuV with the fault injection controller. Therefore, these blocks have to be connected to, for the fault injection, relevant regions for example the program counter, memory address line, op code and so on. The

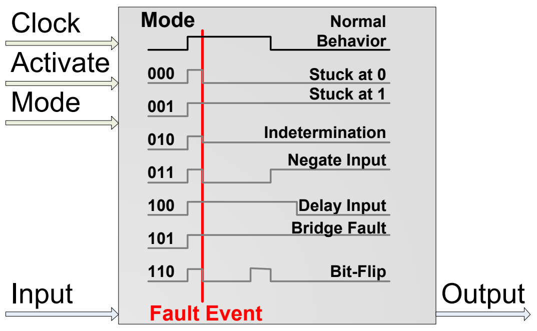


Figure 3.3: Overview of the functionality of a Saboteur (obtained from [5])

trigger will be configured over a specific trigger configuration interface. The triggers can be split up into two main types. The timing and the event triggers. The event triggers are used to trigger on state values of the SuV (e.g. program counter). The timing trigger is used to activate a fault after certain clock cycles. As seen in Figure 3.4 these triggers can be combined to allow counting clock events after an event trigger is activated. For example this can be used if the event trigger is set to trigger when the first program counter value of a critical function is reached. After this trigger is activated the counter begins to count. With this method it is possible to attack every specific clock cycle of the critical function only by counting from the beginning of this function until the specific clock cycle when the fault should be injected is reached.

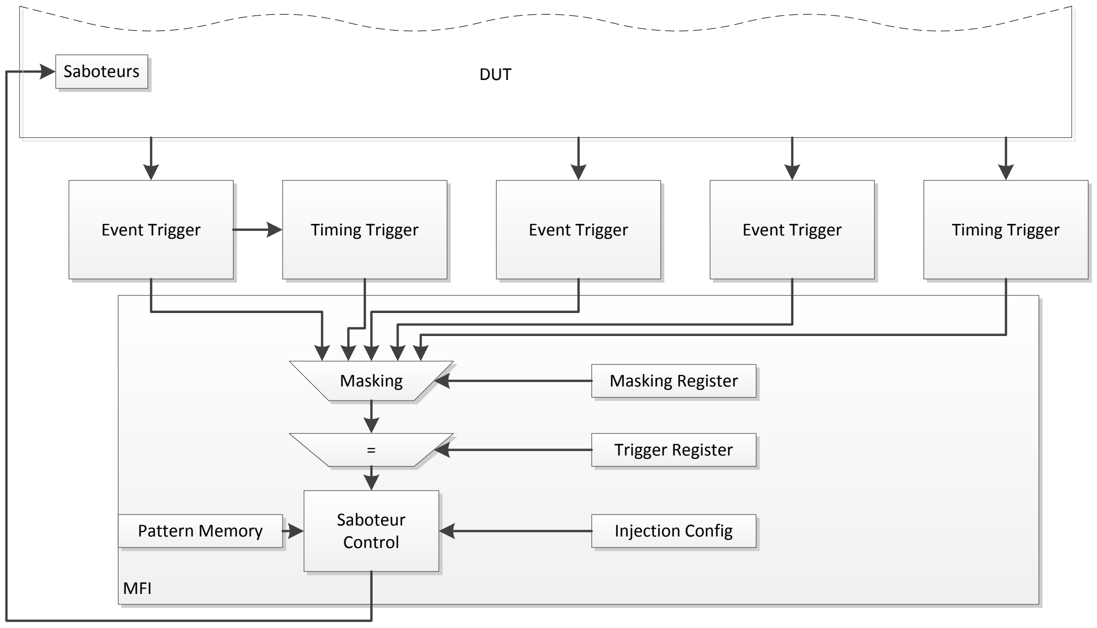


Figure 3.4: Overview of the Trigger Concept

3.3 Automatic System Manipulation

The automatic system manipulation is an essential part in accelerating emulation-based fault injection, because the modification of the hardware description language of the SuV is an error-prone and time consuming process. The system manipulations modify the SoC in such a way that all modules required for producing a specific fault effect are placed automatically. These modules are the MFI, the triggers, the saboteurs and an interface to the environment. For this system manipulation the system has to be analyzed to get the structure of the SoC. After the analysis a verification engineer has to specify which parts of the system should be attacked. Therefore, a GUI was designed (for a more detailed description see Chapter 6.2).

3.3.1 Method

As seen in Figure 3.5 the method can be split into three parts:

- **VHDL analysis:** Figure 3.6 shows the parsing and analysis methodology. The target of this method is to generate a tree which represent the structure of the SoC's hardware description. With the help of this tree it is possible to root signals from the saboteur location to the top-level file of the SoC. This is required because the control signals of the saboteurs must be manipulated. This process is split into three steps:
  - parsing the VHDL files
  - linking of detected modules
  - building of a tree representation

- **Saboteur Placement:** With this step a verification engineer has to specify the attack location. Therefore, the verification engineer has to know exactly where to place the saboteurs. To help the verification engineer place the saboteurs a GUI is provided which shows a project tree. The engineer can use many different categories to decide were to place the saboteurs. The categories are:
  - Dependable relevance of the signal
  - Security relevance of the signal
  - Influence of the signal on the power profile
  - Clock signals
  - Critical signals of fault detection blocks
  - Critical signals of fault recovery blocks
  - Memory interfaces
  - Register ports
  - Data and address buses

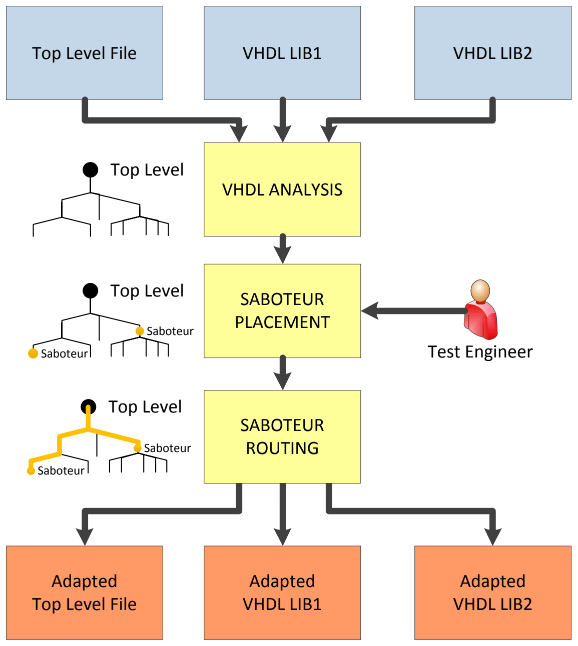


Figure 3.5: System manipulation steps (obtained from [6]

- **Saboteur Routing:** This step routes the signal of the saboteurs and the trigger signals to the top-level of the SoC. Therefore, the control signal of the saboteurs has to be added to the entity, components and declarations of every module which contains the saboteurs. The saboteurs and the triggers are connected to the modular fault injector (MFI).

3.4 Pre-Injection System Analysis

The target of the pre-injection system analysis is to reduce the number of required fault injection campaigns because it is not possible to test every thinkable fault. This reduction can be reached by analyzing the system while a golden model run, to find security relevant regions. This pre-injection system analysis method can be split into two different methods. The memory access analysis and the module activity analysis. (for a more detailed description see Chapter 6.5)

3.4.1 Memory Access Analysis

The memory access analysis tries to find memory cells which are used by a security critical function. This is required because the size of the memory has reached a size where its not possible to test the effect of a fault attack on every memory cell with every possible pattern. Therefore, the number of fault attacks has to be reduced to the most important. The effect of this reduction can be calculated by the Equation 3.1. The time (T) required to test the effect of the fault injection depends on the number of scenarios and the time per scenario. In many cases the time per scenario cannot be improved only the reduction of the test scenarios can reduce the overall test time. For example an emulation of a

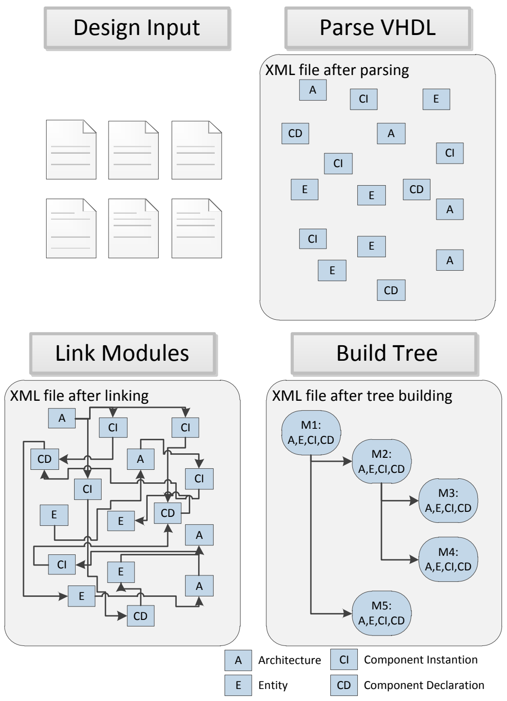


Figure 3.6: System manipulation Method (obtained from [6]

smart card with the parameters shown in Table 3.1 requires, to inject in every memory cell exactly one fault ever 1000 clock cycles, 33 million seconds. This is nearly a whole year.

Memory size	= 100KB
Run-time of the OS	= 100ms
Frequency	= 33MHz

Table 3.1: Example values of a smart card and their operating system.

$$T = n_{scenarios} * t_{scenario} \tag{3.1}$$

To reduce this amount of required attacks an analysis of the memory accesses can be done. Therefore, all memory accesses, while a security critical code is executed, are analyzed and the gained information is used to find security critical memory cells. As seen in Figure 3.8 only a limited set of memory cells are used from the security critical code.

For reducing the amount of fault attack scenarios the information if a memory access is a read or write access is relevant. This knowledge allows emulation of the transient fault attacks. Transient fault attacks on memory are attacks which are only active until the next write access to the same memory cell. This can be seen in figure 3.9. With the knowledge of the time and the access type the following rules can be defined to emulate transient memory attacks:

1. Inject a fault to a memory cell before every read access until the next write access.

2. Do not inject a fault before a write access.
3. Inject a fault before the last write access.

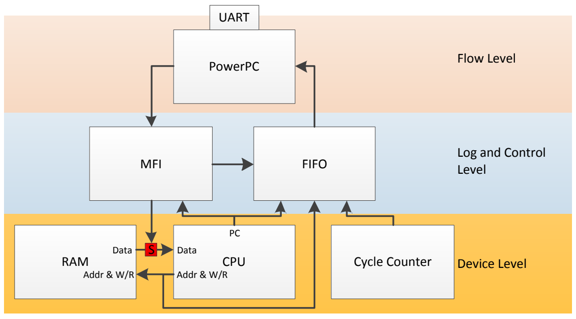


Figure 3.7: Hierarchical model of the pre-injection memory access analysis hardware structure (obtained from [7]

The system to perform such a memory log is build as shown in Figure 3.7. This system consists of the following blocks:

- **RAM:** The memory of the SuV.
- **CPU:** The CPU of the SuV.
- **MFI:** The modular fault injector.
- **Cycle Counter:** A counter which counts the clock cycles beginning after a SuV reset.
- **FIFO:** This FIFO stores all memory accesses from the SuV CPU to their memory.
- **Flow Controlling CPU (in this case the PowerPC):** This CPU is used as the connection between the verification engineer and the fault injection method.

This system allows for logging all memory accesses of an SuV and analyzes them to find security critical memory cells. Therefore, every memory access, while a critical code section is executed, has to be stored into a FIFO and analyzed. As seen in Figure 3.8 an SuV only uses a small part of the whole memory while critical code is executed. If the verification engineer wants to test the security of the critical code region it is enough to attack exactly the memory cells which are used within these critical code regions. This reduces the amount of required fault attacks enormously.

In Figure 3.9 it is also shown that with the generated information it is easily possible to emulate different attack models, as well as knowing the exact time of a read or write execution on a specific memory cell. From this we can emulate transient or permanent faults. For transient fault the manipulation of the memory cell is only active until the next write access to the memory cell.

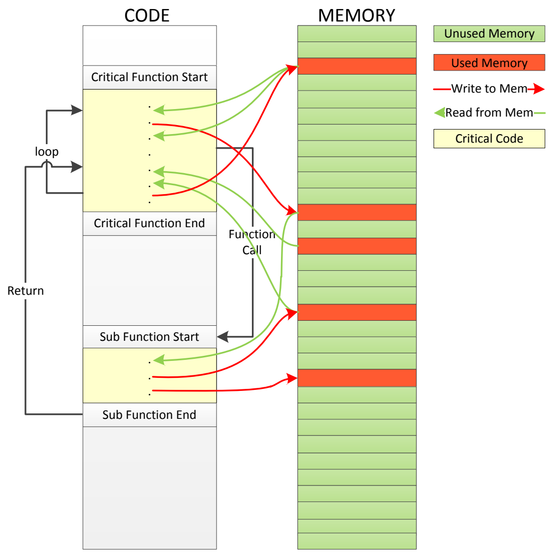


Figure 3.8: This figure shows the memory analysis problem (obtained from [7])

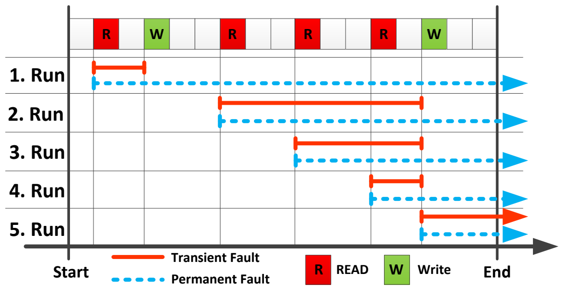


Figure 3.9: In this figure it can be shown how a memory attack can be performed (obtained from [7])

3.5 Post-Injection Fault Effect Analysis

The post-injection fault effect analysis (PIFEA) is a method about how the effects of fault attacks can be analyzed directly on the FPGA. The aim behind this analysis is to give a verification engineer a hint which fault injection scenarios have a security critical effect on the system-under-verification (SuV). Therefore, it must be defined what effects are security critical. This work is focused on smart card security, so we can define that a successful fault attack has to manipulate the output of the device or manipulate the content of the non-volatile-memory (NVM). With this method every successful fault attack can be detected, but to get a better insight into about what happened after the fault was injected, the PIFEA tool is used to check if the control flow was manipulated (for a more detailed description see Chapter 6.4).

To solve these problems the hardware shown in Figure 3.10 was designed. As seen in this figure the communication of the SuV to the environment, the control flow and the memory content are connected to the PIFEA module. To control the PIFEA module several interfaces have been designed.

3.5.1 The Post-Injection Fault Effect Analysis (PIFEA) Block

The structure of the PIFEA module consists of five main components (see Figure 3.11).

- **Memory:** The memory stores the communication of the SuV to the environment, the control flow and the memory content while the PIFEA module is activated.
- **Golden Model Memory:** Stores also the communication of the SuV to the environment, the control flow and the memory content but only while the golden model is running.
- **Comparator:** The comparator checks if there are differences between the golden model run and the fault injection run.
- **Secure State Definition Memory:** In this memory secure states can be defined. For example a secure state could be that the system resets or a fault detection process is activated.
- **Secure State Checker:** This checker proves if the system is in a secure state after a fault injection or not.

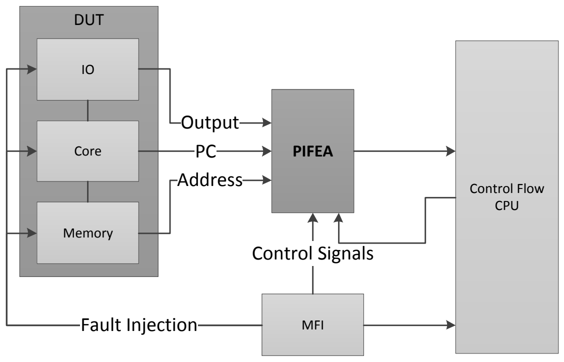


Figure 3.10: Overview of the connection of the PIFEA to the environment (obtained from [8]).

3.5.2 Fault Effect Analysis with Power Information

A help for the verification engineer is the power leakage information of a chip after a fault attack. Therefore, a power-emulator is used. The functionality of this emulator is shown in Figure 3.12. As seen in this figure the power information is calculated on the FPGA using activity signals of active blocks which are weighted by a constant, which represent the power consumption of the block. These constants are estimated while a characterization process which is based on modelsim and physical measurements (for a more detailed description see Chapter 6.6 and Chapter 6.7).

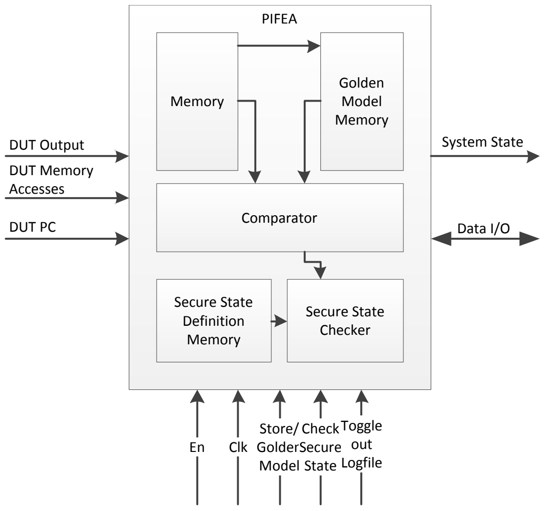


Figure 3.11: The internal structure of the PIFEA module (obtained from [8]).

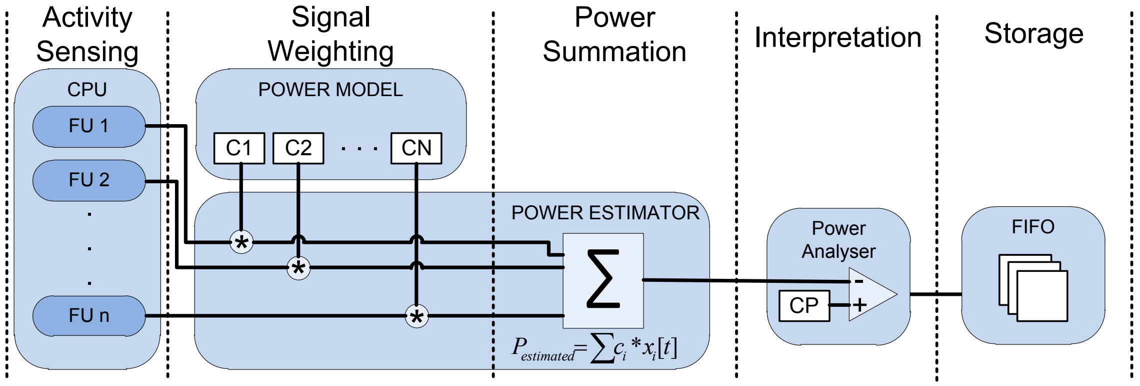


Figure 3.12: Power Emulation - Principle Architecture (obtained from [9])

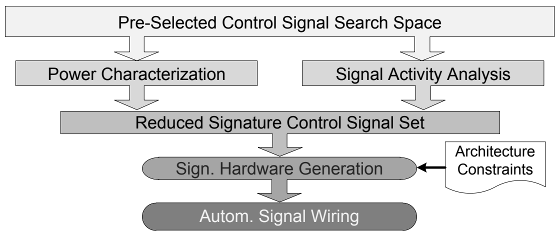

Pre-Selected Control Signal Search Space
Power Characterization
Signal Activity Analysis
Reduced Signature Control Signal Set
Sign. Hardware Generation
Architecture Constraints
Autom. Signal Wiring

Figure 3.13: Signature architecture augmentation methodology (obtained from [10]).

3.5.3 Fault Effect Analysis with Signatures

The last method presented in this thesis which helps the verification engineer to find which fault injection campaign has lead to a security critical state of the system, is the signatures approach shown in Chapter 6.7. This approach uses a golden model to store information while the execution of the software the activation time of certain control

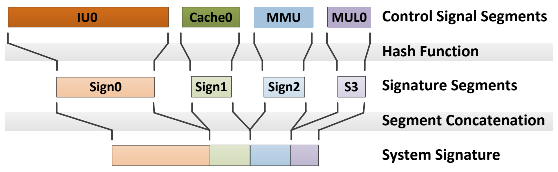


Figure 3.14: Signature segmentation approach (obtained from [10]).

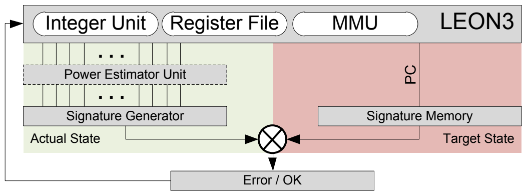


Figure 3.15: Signature generation and comparison architecture (obtained from [10]).

signals. If a successful fault attack occurs this flow should be manipulated (see Figure 3.13). The other advantage of these signatures is that with the signature position the verification engineer can see what blocks are manipulated (see Figure 3.14). If the system is manipulated (see Figure 3.15) then automatically a system control unit on the FPGA detects this manipulation and the verification engineer knows that this injection has an effect on the system(for a more detailed description see Chapter 6.8).

Chapter 4

Evaluation and Case Study

In this chapter an overview about the experimental results is given.

4.1 Test Environment

In this section the platform for the emulation and simulation are presented. Additionally the test target is shown.

- **Emulation Platform:** As FPGA platform the Xilinx ML507 [44] evaluation board with a Virtex5 FPGA is used. The synthesis required for the generation of the netlist for the FPGA is generated on an AMD X6 based PC with a 3.2 GHz processor and 8GB RAM.
- **Simulation Platform:** For the simulation results the Modelsim software which is part of the ISE software package provided by Xilinx are used. The simulation is running on an AMD X6 based PC with a 3.2 GHz processor and 8GB RAM.
- **Test System:** The test system which is used for showing the effect of the fault emulation is the LEON3 Processor which is a SPARC V8 conformal processor of Gaisler Research [45].

4.2 Fault Injection Method

The first tests show the performance of the fault injection method. For a detailed description of the test see Chapter 6.1. The configuration of the fault injection setup is shown in Table 4.1. As seen the configuration is a single-bit saboteur approach with a bit-flip fault model. The attack targets where the saboteurs are placed are: the integer pipeline, the cache controller, the register file, the multiplier unit and the divider unit.

4.2.1 LEON3 Platform Setup

The configuration of the LEON3 processor is a single-core configuration with the default platform settings for the ML507 evaluation board. The PowerPc is used to communicate with the MFI over a GPIO interface.

Table 4.1: Fault injection setup

Saboteur type	Fault mode	Fault target type	Fault injection targets
Single-bit	Bit-flip	Control logic	Integer pipeline
			Cache controller
			Register file
			Multiplier unit
			Divider unit

Table 4.2: Simulated fault injection performance

Saboteurs	Inj. Patterns	Time [sec]	Patterns/sec.
8	256	274	0.93
16	256	282	0.91
24	256	286	0.90
32	256	299	0.86

Table 4.3: VHDL synthesis results

Saboteurs	Look-up-tables	Overhead[%]	Slices	Overhead[%]
0	14897	-	10423	-
8	14950	0.36%	10513	0.86%
32	15107	1.41%	10642	2.10%
96	15194	1.99%	10716	2.81%
170	15244	2.33%	10774	3.37%
326	15478	3.9%	11088	6.34%

4.2.2 Principle Simulation Results

To ensure that the fault injection method worked as intended a simulation using the ModelSIM software package was used. The results of these injection campaigns are shown in Table 4.2. As above, the pattern injection rate is nearly constant when the number of saboteurs is smaller than 32. If this number increases, additional configuration overhead of the MFI must be added which leads to a decreased performance.

4.2.3 VHDL Synthesis Results

After the simulation was run correctly the fault injection controller can be synthesized and the results are shown in Table 4.3. As seen in this table the size of the saboteurs has only a small overhead to the overall system, with 326 saboteurs and only an overhead of 3.9 percent is generated.

Table 4.4: Emulated fault injection performance

Saboteurs	Inj. Patterns	Time [sec]	Patterns/sec.	Pattern Blocks/sec.
8	100M	46.76	2.17M	2.17M
32	100M	46.76	2.17M	2.17M
96	100M	114.48	873.5k	2.17M
170	100M	156,4	639.4k	2.17M
326	100M	282.16	354.4k	2.17M

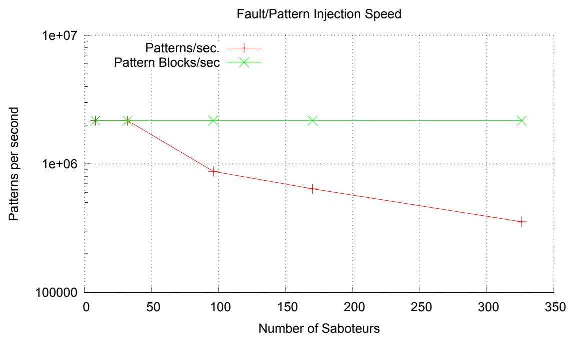


Figure 4.1: Speed of the fault injection depending on the number of saboteurs (obtained from [5]).

4.2.4 Fault Injection Performance

One of the most important values to measure the quality of the fault injection approach is the speed of how fast faults can be injected. As seen in Table 4.4 the speed is influenced by the number of saboteurs which must be changed between the fault injection campaigns. The number of saboteurs which can be manipulated in parallel are 32. The number of saboteurs which have to be changed is called patterns. If the pattern size is less than 32 it is possible to write 2.17 million patterns per second (see Figure 4.1).

4.2.5 Case Study: Long-time Test for Light Attack Emulation on the Memory of a Smart Card

To show that this fault injection method can be used for real products we tested this method together with the companies Austria Card with their operating system development for smart card systems. The detailed test is shown in Chapter 6.3. This test checks if the MFI can handle fault injection campaigns with a long duration. For this test we chose a light attack which is described in [46]. Faults are injected into the memory of the emulated smart card and prove that the software security features can cope with this faults.

As seen in Figure 4.2 not every region of the memory is security critical over the whole run-time of a system. Using the knowledge of the verification engineer of Austria Card

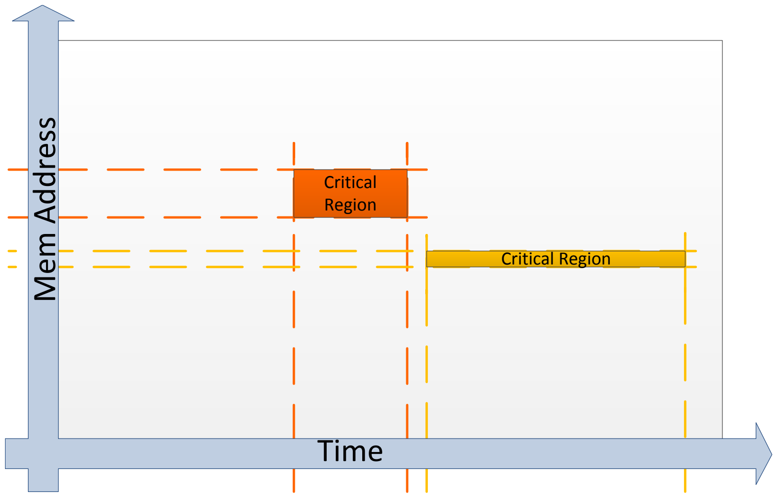


Figure 4.2: Critical memory regions of a smart card system (obtained from [11]).

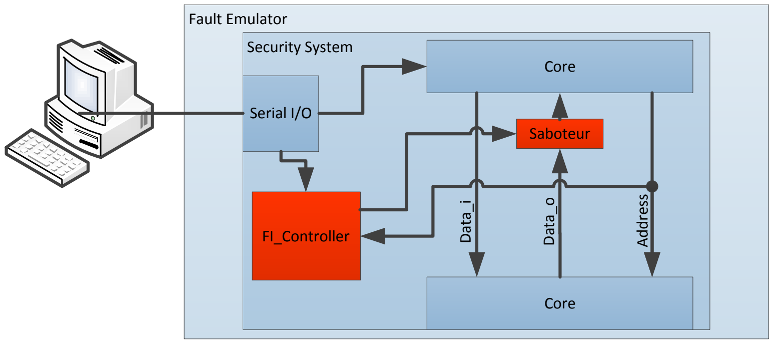


Figure 4.3: System modifications for the RAM fault emulations (obtained from [11]).

we chose memory cells for the attacks where security critical information is stored (e.g. access rights). With the Equation 4.1 it is possible to calculate the required fault injection campaigns to test the system against fault attacks. For this test we attack 1536 memory addresses at 102 specific clock cycles with one attack patterns. As seen in Equation 4.2 this leads to a total of 156672 injection campaigns.

$$N_{tests} = N_{memoryaddresses} * N_{clkcycles} * N_{attackpatterns} \tag{4.1}$$

For this test we place the saboteur as seen in Figure 4.3 between the Core and the Memory of the system.

$$N_{tests} = 1536 * 102 * 1 = 156672 \tag{4.2}$$

Table 4.5 shows the results of this fault test. 121570 memory cells are never accessed after the fault injection is performed. In 35102 cases the fault was injected into a memory cell which was used after the attack. In 30373 of these cases the fault has no effect on the operating system. Only 4729 fault injections had an effect to the smart card, but all

Table 4.5: Results of the light attack emulation on the memory

	N	%
Total number of injections	156672	100%
Memory cell never accessed	121570	77,6%
Fault injected	35102	22,4%
No effect	30373	19,4%
Fault detected by smart card	4729	3,0%

of them were detected by the software security features. There was no successful attack against the smart card operating system developed by Austria Card.

4.3 Automatic System Manipulation

The automatic system manipulation has the aim to reduce the effort for the verification engineer. Therefore, the time consuming and error-prone process of the manually manipulation of the hardware description of the SuV is replaced by an automatic approach. The modules which have to be added to the hardware description are: MFI, triggers and saboteurs.

The test results of the automatic saboteur placement are shown in detail in chapter 6.2. The most important results are summarized in this section.

The fault injection which should be emulated to show that the automatic system manipulation is working correctly is a method shown in [47] where it would be assumed that if while a RSA authentication only one bit of a multiplication is disturbed, it is possible to extract the private key. In Figure 4.5 it shows where the saboteurs should be placed to emulate this attack.

To reproduce this fault attack saboteurs have to be placed at the hardware multiplier of the SuV. This can be done easily by using the automatic system manipulation tool. Figure 4.4 shows how the GUI looks like, which is used for the saboteur placement. For a more detailed explanation of the results see Chapter 6.2

Table 4.6 shows the required time for the system manipulation. As seen in this table the system manipulation is split into two parts: the design analysis and the design adaption. The design analysis has to be performed once during the whole system manipulation step. For the test system this analysis requires 70s. The system manipulation step requires for each saboteur exactly 200ms. This means if 100 saboteurs are placed, 20s are required. Both times together give the time required by the tool to manipulate the system.

Table 4.7 shows the synthesis results of the manipulated design. The manual and the automatic approach requires the same amount of slices which shows that the automatic approach has no disadvantages against the manual approach. It also shows that the overhead is rising depending on the number of saboteurs placed into the design.

4.4 Pre-Injection System Analysis

The automatic pre-injection system analysis has the aim to reduce the required fault injection campaigns to an absolute minimum. Therefore, the system has to be checked

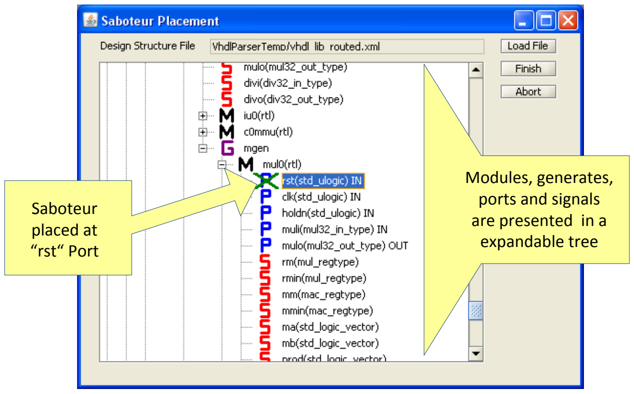


Figure 4.4: The saboteur placement GUI allows the verification engineer to specify the location of the saboteurs.

Table 4.6: VHDL design analysis and saboteur placement time

VHDL design analysis	
Parsing	48565ms
Linking	5373ms
Routing	16072ms
Sum	**70010ms**
VHDL design adaption time per saboteur	
Saboteur placement	20ms
Signal routing	180ms
Sum	**200ms**

Table 4.7: Synthesis time and hardware requirements depending on the number of saboteurs.

Nr. Saboteurs	**0**	**10**	**10**	**50**
Routing method	n.a.	manual	auto.	auto.
Synthesis Time[s][a]	1536	1543	1543	1559
Synthesis Time OH[a,b][%]	-	0.46	0.46	1.5
Slices	14780	14950	14948	15172
Slices OH[a][%]	-	1.15	1.14	2.65

during a golden model run. The detailed experimental results of this approach are shown in Chapter 6.5. The most important results are summarized in this section. Table 4.8 shows the detailed configuration of the Leon3 processor for this experiment. For this test the saboteurs are placed between the core and the cache of the Leon3 processor.

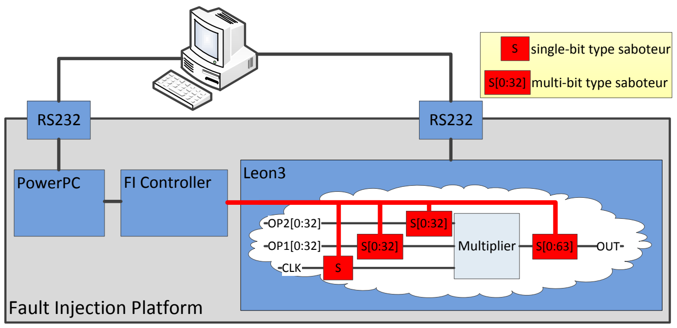


Figure 4.5: Evaluation of the fault injection flow test setup (adapted from [6])

Table 4.8: LEON3 Configuration

Operating Frequency [MHz]	40
Instruction Cache Sets	2
Instruction Cache Set Size [kB]	4
Data Cache Sets	1
Data Cache Set Size [kB]	4

Table 4.9: Fault injection setup

	Test System	Test System & Per-Injection Memory Analysis	Overhead [%]
Number Slices	11401	12137	6.5
Number LUT	20832	20954	0.6
Number BRAM's	38	103	171
Total Memory(KB)	1332	3654	174

4.4.1 Hardware Overhead

As seen in Table 4.9 the hardware overhead of the pre-injection system analysis tool is not very high. Only the number of BRAM to store the golden model information is high. This is not a really big problem because the BRAM can be replaced by external memory. Normally an emulation platform has enough memory placed on the board to allow the storage several seconds of golden model run.

4.4.2 Proof of Concept

To prove that the pre-injection system analysis is working a short program was executed and analyzed by the pre-injection system analysis concept. The program is showing in Listing 4.1. Table 4.10 shows the memory access measured with the tool and the estimated memory accesses. The estimated values are generated by analyzing the assembler code of the test program. As seen the number of accesses is the same. Table 4.11 shows the

Table 4.10: Memory accesses of the proof of concept example.

	estimated value	measured value
Read	23	23
Write	22	22
Total Memory Accesses	47	47

Table 4.11: Results for the proof of concept example.

	Simulation [s]	Emulation [s]	Speed-up factor
per run	603	0.224	2692
25 Test Cases	15065	5.6	2690
10000 Test Cases	est. 6030000	2260	2668

required time for simulating and emulating this proof of concept example.

```
int test(){
        volatile int i=0,h=0;
        for(i=0;i<10;i++){
                h+=i;
        }
        return h;
}
```

Listing 4.1: Test Program for Proof of Concept

4.4.3 AES Example

This approach also works on a larger test program, therefore, an AES implementation from [48] is used. Only the encryption of a 16 Byte plain text was attacked. Table 4.12 and Table 4.13 shows the results of this experiment. As seen in these tables the speed-up between the simulation and the emulation approach is approximately 2600 and the reduction of required fault injection scenarios generated by the pre-injection system analysis tool is approximately 1000.

4.5 Post-Injection Fault Effect Analysis

The PIFEA module is used to analyze the effect of a fault injection to the system. For detailed information see Chapter 6.4. In this section the most important results are summarized. To prove the functionality of the PIFEA tool saboteurs are placed into the memory interface of the Leon3 processor and a small program is executed on the Leon3 (see Listing 4.2). With this program it is possible to test if a fault injection can manipulate the system in a way that the program flow, the output or the none volatile memory is manipulated. If one of these parameters was disturbed the tool reports a security critical behavior of the SuV which indicates the verification engineer that this fault injection is possibly security critical. The hardware overhead of the PIFEA tool is shown in Table 4.14. As seen the biggest hardware overhead to the total system of the PIFEA tool is the

Table 4.12: Data memory accesses during AES encryption

Read	250
Write	111
Total Memory Accesses	361
Rom Addresses	121
Ram Addresses	42
Total Addresses	163
Total Clk Cycles	2584
Test Cases (without our approach)	249873
Test Cases (with our approach)	249
Speed-up factor	1003

Table 4.13: Simulation and emulation time for the AES example

	Simulation [s]	Emulation [s]	Speed-up factor
per run	480	0.5	960
249 Test Cases	est. 119520	120	996
249873 Test Cases	est. 119939040	119050	1007

Table 4.14: Required number of FPGA slices

	Test System	MFI	PIFEA
Number Slices	7811	2291	238
Number LUT	14081	2043	358
Number BRAM's	30	0	90
Total Memory(KB)	1080	0	3132

required memory. Due to the emulation platforms have a large amount of memory this does not lead to a problem.

```
function test begin
  variable i=0,result=0,flow_contol=0;
      variable array[10]=0;
  for  i=1 to 10 do           //manipulate i
    if flow_contol==0 then    //manipulate flow_contol
      result+=array[i];       //manipulate i
    else
      ...                     //unallowed code
  end for
  output(result);             //manipulate result
end function
```

Listing 4.2: Test Program for Proof of Concept

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The enormous increase of complexity and usage of smart card systems leads to a high effort to verify if these cards fulfill the security requirements. Therefore, several methods have been developed over the last decades to prove these requirements. Because smart cards are often used in high security environment, certification schemes like common criteria are often utilized. Specialized protection profiles and security targets define rules that smart cards have to fulfill in order to achieve common criteria certification. A typical part of the evaluation process is the physical fault attack verification, where e.g. a laser is used to manipulate the smart card chip. A security evaluations are very expensive, the manufacturers are aware to remove as many design bugs as possible before the evaluation in an external laboratory starts. Therefore, the system must be tested directly while the system is being developed. For this issue fault emulation could be used, but the time for testing a smart card against all possible fault attacks is enormous.

In this thesis a method is presented on how to accelerate the fault injection process by improving the fault injection method, reducing the required test scenarios and improving the analysis of the results of the fault injection. The improvement of the fault injection method is done by developing a modular fault injection concept which allow for automatic system manipulation which reduces the time consuming and error-prone process of manual system manipulation. This step does not accelerate the verification process in such a way that a whole system can be tested against fault attacks. Therefore, the number of required fault injection scenarios has to be reduced. For the reduction of the scenarios the system has to be analyzed while a golden model run. While this run is happening the the usage of the memory is analyzed, and now it is possible to inject a fault into the memory, but only when this area is active. With this improvement it is possible to reduce the required scenarios from over some billions to a handleable number of some millions. To check the results by a verification engineer manually the number of some million scenarios is also too high. Therefore, a module was developed which analyze directly on the emulation platform what effect a fault injection have to the system. This allows to highlight fault injections which manipulate the system in a not allowed way.

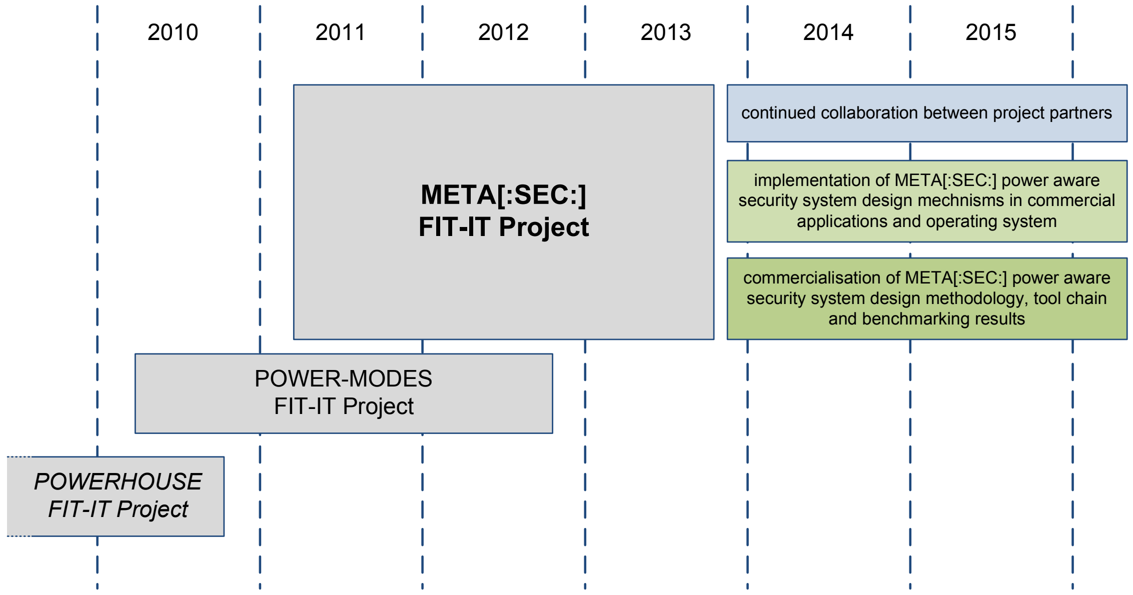

2010
2011
2012
2013
2014
2015
META[:SEC:] FIT-IT Project
continued collaboration between project partners
implementation of META[:SEC:] power aware security system design mechnisms in commercial applications and operating system
commercialisation of META[:SEC:] power aware security system design methodology, tool chain and benchmarking results
POWER-MODES FIT-IT Project
POWERHOUSE FIT-IT Project

Figure 5.1: Long term perspective of the Power-Modes Projects, obtained from [12]

5.2 Future Work

5.2.1 Analyze High Order Fault Attacks

High order fault attacks are a combination of normal fault attacks and side-channel attacks. A fault attack is used to manipulate a system in such a way that the system leaks side-channel information (e.g. power and timing) in such a way that this information can be used to extract security critical information. To cope with this problem the system has to analyze the side channel information during the fault injection process. Therefore, a method has to be developed to analyze leakage information automatically on the emulation platform.

5.2.2 Connection to other Research Projects

This project is one part of a large cooperation between Infineon Technologies Austria AG, Austria Card Plastikkarten und Ausweissysteme GmbH and RF-iT Solution which is split into multiple parts. In Figure 5.1 an overview of the ongoing and previous projects is shown.

The POWERHOUSE[1] (POWER-aware, Hardware-supported Operating System and Ubiquitous application Software development Environment) Project has been established to focus on a power emulation during operating system development for smart card systems. This power emulation method was used in the Power-Modes Project to generate power information together with the fault verification.

[1]POWER-aware, Hardware-supported Operating System and Ubiquitous application Software development Environment. Project Partners are Infineon Technologies Austria AG, Austria Card Plastikkarten und Ausweissysteme GmbH and TU Graz. The project is funded by the Austrian Federal Ministry for Technology under the FIT-IT contract FFG 815193

The META[:SEC:][2](Mobile Energy-efficient Trustworthy Authentication System with Elliptic Curve based SECurity) Project brings the power emulation from the POWER-HOUSE Project and fault injection method from the Power-Modes Project to a system view. The target is to generate power and security information with the development of reader card systems.

[2]Mobile Energy-efficient Trustworthy Authentication System with Elliptic Curve based SECurity. Project Partners are Infineon Technologies Austria AG, RF-iT Solution and TU Graz. The project is funded by the Austrian Federal Ministry for Technology under the FIT-IT contract FFG 829586

Chapter 6

Publications

This chapter contains all the relevant publications that were published whilst writing this PhD thesis. Figure 6.1 shows an overview of all publications and how the publications can be mapped to the work packages. The work packages of this thesis are the fault emulation platform generation and the emulation platform itself. The platform can also be split into several parts. The Pre-Injection System Analysis which handles the generation of information about the system before the fault injection campaign are stated. This information is used to reduce the required fault injection campaigns to a minimum. The next block is the fault injector. This block uses the information generated at the Pre-Injection System Analysis and manipulates the behavior of the system in such an manner that a special fault effect can be produced. Finally the effect of the fault to the overall system has to be analyzed. Therefore, the post-injection Fault Verification block is designed. This block checks if an attack caused unindented behavior to the system-under-verification. Therefore two different ways which work in parallel are developed. The first one is the PIFEA module which detects control flow changes. The second is the signature based approach which detected successful attacks to the system because the attack signatures were manipulated.
Publication 1: *Modular Fault Injection for Multiple Fault Dependability and Security Evaluation*, 14th Euromicro Conference on Digital System Design, Oulu, Finland, 2011

Publication 2: *Automatic Saboteur Placement for Emulation-Based Multi-Bit Fault Injection*, 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip, Montpellier, France, 2011

Publication 3: *Case Study on Multiple Fault Dependability and Security Evaluation*, Elsevier Microprocessors and Microsystems, in press 2013

Publication 4: *Efficient Fault Emulation based on Post-Injection Fault Effect Analysis (PIFEA)*, 55th International Midwest Symposium on Circuits and Systems, Boise, USA, 2012

Publication 5: *Efficient Fault Emulation using Automatic Pre-Injection Memory Access Analysis*, 25th IEEE System-on-Chip Conference, Niagara Falls, USA, 2012

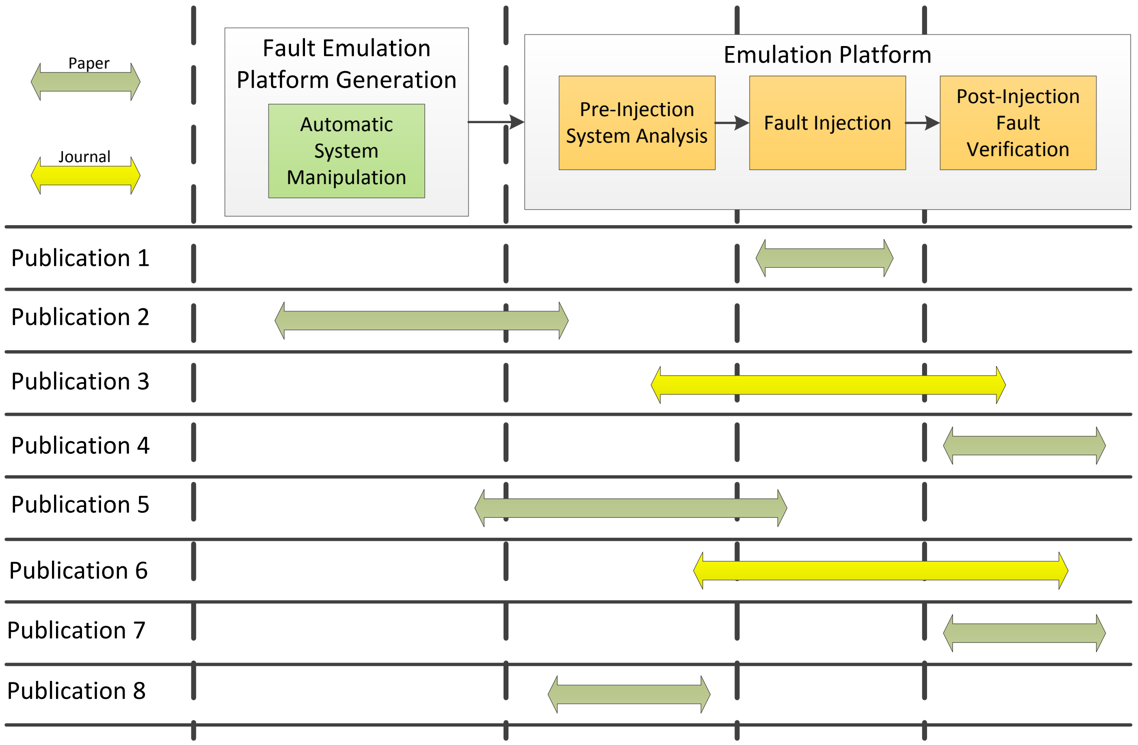


Figure 6.1: Overview of the publications done for the acceleration of the system-on-chip fault emulation tool.

Publication 6: *POWER-MODES- POWer EmulatoR and MOdel based Dependability and Security evaluation*, ACM Transactions on Reconfigurable Technology and System, 2012

Publication 7: *Characterization and Handling of Low-Cost Micro-Architectural Signatures in MPSoCs*, IEEE 25th European Test Symposium, Annecy, France, 2012

Publication 8: *Acceleration of Fault Attack Emulation by Consideration of Fault Propagation*, International Conference on Field-Programmable Technology, Seoul, Korea, 2012

2011 14th Euromicro Conference on Digital System Design

Modular Fault Injector for Multiple Fault Dependability and Security Evaluations

Johannes Grinschgl, Armin Krieg,
Christian Steger and Reinhold Weiss
Institute for Technical Informatics
Graz University of Technology
Graz, Austria
{johannes.grinschgl, armin.krieg,
steger, weiss}@tugraz.at

Holger Bock, Josef Haid
Design Center Graz
Infineon Technologies Austria AG
Graz, Austria
{holger.bock, josef.haid}@infineon.com

***Abstract*—The increasing level of integration and decreasing size of circuit elements leads to greater probabilities of operational faults. More sensible electronic devices are also more prone to external influences by energizing radiation. Additionally not only natural causes of faults are a concern of today's chip designers. Especially smart cards are exposed to complex attacks through which an adversary tries to extract knowledge from a secured system by putting it into an undefined state. These problems make it increasingly necessary to test a new design for its fault robustness. Several previous publications propose the usage of single bit injection platforms, but the limited impact of these campaigns might not be the right choice to provide a wide fault attack coverage.**

This paper first introduces a new in-system fault injection strategy for automatic test pattern injection. Secondly, an approach is presented that provides an abstraction of the internal fault injection structures to a more generic high level view. Through this abstraction it is possible to support the task separation of design and test-engineers and to enable the emulation of physical attacks on circuit level. The controller's generalized interface provides the ability to use the developed controller on different systems using the same bus system. The high level of abstraction is combinable with the advantage of high performance autonomous emulations on high end FPGA-platforms.

***Index Terms*—fault emulation, fault injection controller, saboteurs, multi-bit faults, automatic test pattern injection**

I. Introduction

The continuing success of the semiconductor industry regarding the progress of structure downscaling has led to highly integrated but also highly sensitive devices. External radiation effects, thermal and electric degradation have become common problems for the dependability of a system [1].

Through these effects transient or even permanent faults are introduced which lead to a change of the system's behavior. These faults happen randomly in contrast to ones resulting out of so-called fault attack scenarios. In this case an attacker deliberately injects faults into a system to change the system behavior. If no precautions are taken it is possible that such an attack is successful [2].

Therefore in the last years intensive research has introduced several different tools to simulate or emulate possible fault scenarios during the design phase. Especially fault emulation

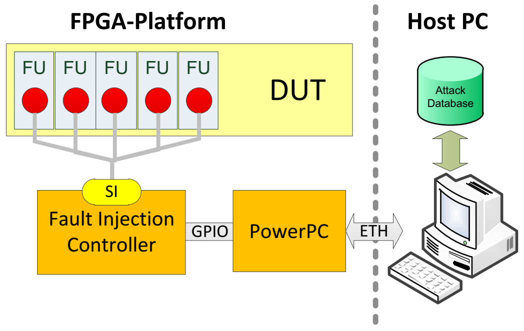


Fig. 1. Schematic view of the proposed fault injection system

proved to be a very effective way of testing a system under influence of a fault source. The usual platform to emulate such a faulty system is an FPGA because of its flexibility. An example of such an FPGA fault injection platform is shown in Figure 1. There are different ways of injecting such a fault into a circuit. One would be the use of partial reconfiguration features of the used FPGA [3]. This design approach also heavily limits the platform choice as there are only few candidates such as the Virtex families from Xilinx [4]. Another way is to instrument the given circuit either with manipulated logic elements or with integrated controllable fault elements. In the latter case it can be distinguished between saboteurs and mutants [5]. Saboteurs are small circuit elements which do not affect system behavior under normal conditions. If activated they directly inject faults into the targeted submodule by disturbing the internal signals. To disturb signals saboteurs have to placed between there source and there sink. Mutants are modified submodules that also do not affect system behavior under normal conditions but if activated behave like a faulty version of the original. To simulate or emulate fault attacks with mutants the submodule have to be replaced by a mutated submodule.

To accomplish such a test setup, it is necessary to have access to the hardware description or a standardized test interface. Such a test interface could be test chains [6]. Another important step in creating an effective fault injection platform is the selection of a proper fault model. For dependability evaluations a single event upset (SEU) fault model is often sufficient. In case of faults caused by radiation or degradation it can be safely assumed that only a single random fault will happen at a time [7]. Security evaluations on the contrary consider intentional faults. Therefore it is possible that an attacker introduces several faults at once to achieve the desired effect of secret exposure.

Such a multi-bit fault model results in much more complex fault relationships than SEU ones, as time and space of such faults cannot be assumed of being random. This model has to reflect the dependency of several SEUs among each other. In case of dependability analyses this means that similar types of sensitive sub-circuits will fail at the same time. In case of security evaluations this dependency is defined by the attack scheme of the adversary.

Finally, the controlling element itself has to be designed very carefully. The scope reaches from a very simple implementation completely controlled by an external source like a personal computer to very sophisticated integrated designs. Simple implementations have the advantage of being very adaptable to system changes, but they are also highly dependent on the used communication interface. Therefore they can be considered as slow. Such an implementation based on a PCI interface is shown in [6]. Sophisticated solutions allow for very high fault injection rates while being difficult to adapt to system changes [8].

In the past years implementations of fault injection emulation controllers are either simple and limited by slow interface performance or they are highly complex and therefore not portable from one design to another.

In this paper we propose a modular fault injector(MFI) that combines the advantages of a simple portable design and fast highly complex implementations. It also helps to provide a common platform for fault injection campaigns on different target architectures. Another point that is often neglected in previous work in this field is the consideration of fault attacks.

Finally the consideration of multi-bit fault patterns instead of simple SEUs can be seen as an important contribution of this work. This is especially important for security evaluations. It has also to be mentioned that this fully synthesizable controller can also be used as an online-testing implementation if desired.

The main goals of this work can be summarized as follows:

- Fully modular fault injector design
- Multi-bit fault injection to support fault attack emulation
- Online-testing support

This paper is structured as follows. Section II briefly shows the state of the art on work related to emulated fault injection using integrated controllers and different reconfiguration techniques. In Section III the design of the fault injection controller is described. Section IV shows some experimental results of this MFI by means of the LEON3 processor [9]. Finally, conclusions are drawn in Section V.

II. Related Work

The introduction of fault emulation on FPGA platforms has led to several publications especially concerned with the emulation of SEUs in space applications. It has to be distinguished between approaches using direct circuit manipulation like the solution proposed in this paper and techniques using reconfiguration features of certain FPGAs.

The former possibility can again be divided into two subcategories. One using specialized hardware units to instrument saboteurs or mutants as shown in [8].

The second variant is to use available processor cores for fault injection automation similar to the solution provided by [10]. This solution promises a high emulation performance because of fast on-chip communication channels, but it is more difficult to port to other platforms.

In the last years partial and complete runtime reconfiguration of FPGAs got a common technique to implement fault injection in a flexible way. Reconfiguration is possible on special FPGA series using an external communication interface to directly feed different fault configurations into the FPGA as shown in [11]–[13]. If this external interface is too slow it is also possible to use existing processor resources for the reconfiguration task as presented in [14], [15]. While the reconfiguration approach is tempting (and has been used by several research groups) it also limits the maximum reachable fault injection performance and the designer's platform selection. Similar to reconfiguration approaches is the work presented in [16], in which the synthesized netlist is augmented to preserve the original structure of the system.

In the field of security evaluations work has been done by the authors of [17]. The presented work is done using a proven fault injection platform presented in [18]. This investigation confirms the strong need for multiple fault injection campaigns in the design process. However in their experiments only a small fault multiplicity (six) was used to prove this point.

III. Modular Fault Injector

The following reasons summarize the main drivers behind the development of this modular fault injection controller (MFI):

- Support for fault patterns to support multi-bit fault injection
- Multi-mode saboteur support
- Scalable interfaces to support automatic saboteur placement
- Standardized communication interface
- Internal memory for automatic fault injection

To support an easy application of the fault injection system to a new design under test, a standardized communication interface is needed. The General Purpose Input/Output (GPIO) communication interface enables the developer to use the proposed controller in a wide selection of different architectures. The GPIO interface consists of pins which can be configured via software commands. These pins allow for controlling the MFI without disturbing the device-under-test.

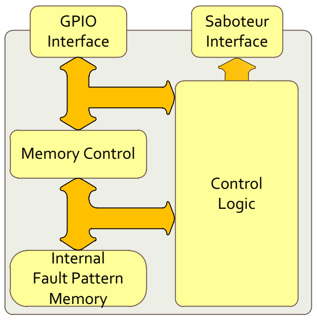


Fig. 2. Schematic view of the fault injection controller

Extensive fault injection campaigns are only possible using a large number of active saboteurs. It is not possible to route such a large number manually, therefore internal interfaces have to be scalable to enable automated injection processes.

An effective separation of the design and test-engineers tasks can only be guaranteed through a generalized view of the fault injection system. This is done using so called fault patterns, high level representations of the underlying saboteur distribution. These patterns are also necessary for efficient physical fault attack emulation.

High speed automated operation is enabled through a flexible internal memory system. In the design phase of the proposed fault injector, future advances of the evaluation platform have to be considered. This includes support of large fault patterns and burst loading from the bus-system.

Different fault and attack scenarios call for different saboteur types. To simplify the reconfiguration process, different saboteur modes are provided by a single flexible saboteur design.

A basic block diagram of the proposed fault injection system is shown in Figure 1. The flow consists of the fault injection controller which specifies the attack scenario, the saboteurs which disturb signals and a PowerPC which is used as the interface to the PC.

A. *Fault Injection Controller*

The fault injection controller is the main part of the MFI. It controls the mode and the activation time of the saboteurs. As seen in Figure 2 the fault injection controller consists of two interfaces. The first one is the saboteur interface. This interface is a bus where for each saboteur an own activate signal is included. It also contains mode signals for controlling the mode of the saboteurs. The second port is the General Purpose Input/Output (GPIO) port. This interface is used to control the fault injection controller. Via the GPIO port the internal fault pattern memory is filled. This memory is used to specify when which attack pattern shall be activated.

Thanks to the GPIO interface of the MFI it is possible to automate fault injection campaigns. The easiest way to control the MFI via GPIO commands is through the FPGA-internal PowerPC. Through this solution a simple generic software interface to the generic hardware interface is provided. This has the advantage that a test-engineer does not need specific knowledge about the fault injection system itself. Especially in case of security evaluations the test-engineer will be mostly concerned with a good mapping of a real attack scenario to the saboteur matrix inside the system of interest. The GPIO interface of the MFI can be easily changed to every other interface which allows to write the internal memory of the MFI. This flexibility makes the whole flow independent to the used test environment.

TABLE I
SABOTEUR OPERATION TYPES AND THEIR DESCRIPTION

Saboteur mode	Fault type	Description
Stuck-at-Zero	Permanent	Signal value of '0' until reload
Stuck-at-One	Permanent	Signal value of '1' until reload
Indetermination	Permanent	Undefined signal state until reload
Bridging fault	Permanent	No output propagation until reload
Negation of input	Permanent	Undefined signal state until reload
Bit-flip	Transient	Output inverts input for one cycle
Artificial delay	Transient	Input to output propagation delay

Implementation: The fault injection controller is split into five main parts.

- Saboteur interface: The saboteur interface was implemented to allow for easy expandability to support a higher number of saboteurs. The size of the saboteur interface bus is equal to the number of saboteurs. The interface also includes mode signals to control the mode of the saboteurs.
- GPIO interface: The GPIO interface is the interface of the MFI to the controlled processor. Via this port all attack scenarios can be loaded into the MFI.
- Memory control: The memory control is used to place the fault patterns into the internal fault pattern memory. It also generates signals for the control logic if a special command is sent via the GPIO port.
- Internal fault pattern memory: This memory stores the fault patterns which define the attack scenario.
- Control logic: The control logic activates the saboteurs depending on the information stored in the internal fault pattern memory.

B. *Saboteurs*

To model a wide selection of possible faults we introduce a configurable saboteur. According to [19] and [20] saboteurs can be distinguished into several different types. Depending on their location it can be differentiated between serial and parallel saboteurs. Depending on the directionality the division can be made between uni- and bidirectional ones. Finally depending on their complexity it can be distinguished between simple and complex saboteurs. The saboteur used in this work can be classified as an unidirectional serial simple saboteur. This means it only works in one direction, it is located directly into the connection signal and affects only one port at the input and output side. The supported fault models of such an injection element are shown in Table I.

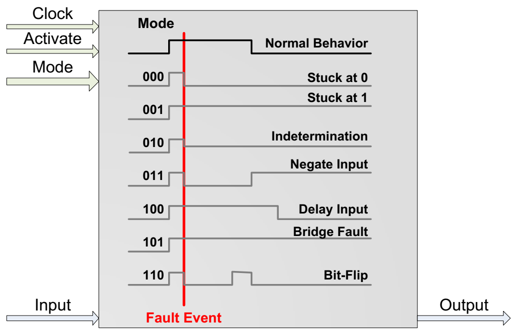


Fig. 3. Block diagram of the proposed saboteur element

The proposed saboteur can be used to inject faults into single bit lines or even complete buses. This allows the emulation of bus attacks with e.g., unfocused lasers or influences by strong external energy sources. The first four saboteur configuration modes are equivalent to direct circuit modifications. Bit-flips could also be forced by short intensive external pulses. Delay faults emulate circuit behavior in case of operating voltage changes. A schematic block diagram and fault effect visualization of the proposed single-bit saboteur is shown in Figure 3.

Implementation: The complexity of the implementation of the saboteurs depends on the features a saboteur shall support. Therefore the required number of slices for one saboteur mainly depends on their complexity. If the interface of the saboteurs is not changed only the architecture of the saboteur has to be modified to support more features. So if the saboteurs are placed the test-engineer can adapt the saboteurs by only changing one file. This makes the saboteur placement concept very flexible.

C. Fault pattern support

To evaluate a device-under-test for its fault attack sensitivity it is important to know the exact location of security relevant silicon regions. At this locations saboteurs have to be placed by a test-engineer. A fault pattern approach is used for an effective mapping of saboteurs to their corresponding fault locations. Of course, not every possible pattern combination has to be transmitted into the MFI. Every used pattern represents a very likely fault scheme determined in previous physical experiments. The basic concept of fault pattern to physical implementation mapping is shown in Figure 4.

Implementation: In this example only a random number generator is used as pattern generator to show the function of the fault MFI. The complexity of such a pattern generation does not increase the required slices of the MFI, because the patterns are not generated directly in the MFI.

D. Automatic Saboteurs

For a high amount of saboteurs an automatic saboteur placement approach is required, because manual saboteur placement is a very time consuming process. In [19], a theoretical method

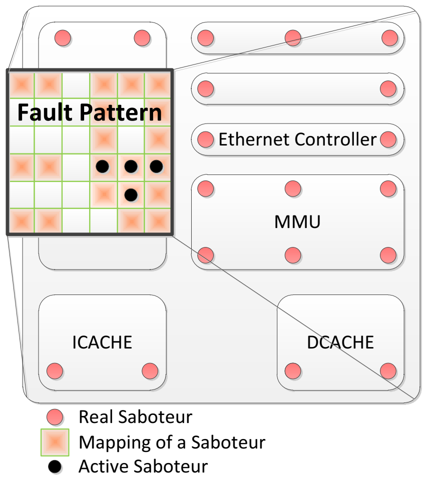


Fig. 4. Schematic view of an extracted fault pattern

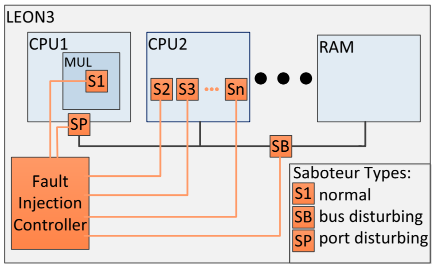


Fig. 5. Automatic routed saboteurs (adapted from [21])

is discussed how to place saboteurs automatically into a VHDL design. Our approach works very similar to that approach. In Figure 5, a schematic example of an automatic saboteur placement is shown. The saboteurs placement tool is based on the vMAGIC VHDL parser library [22]. VMagic is a Java API which allow to parse and adapt VHDL code in an automatic way. The flow can automatically add hundreds of saboteurs [21]. Then the VHDL code is automatically adapted with saboteurs and the fault injection controller. A method to modify VHDL code automatically is shown in [23].

E. Attack Scenario

As seen in Figure 6 the attack flow is based on a golden model run. The golden model runs from one reset to the next reset of the DUT. All information of the golden model is stored (e.g., the output, none volatile memory (NVM), timing, ...).

The next step is to reset the whole system. Then all attack patterns are stored into the internal fault pattern memory via the GPIO interface. Also the attack mode has to be transmitted via GPIO. The mode is used to define the attack type of the saboteurs (e.g., bit-flip). After a pre-determined time the specified fault attack will be injected and then the saboteurs are activated depending on the fault pattern. Now the faulty DUT will run until it is reset or a pre-defined timeout is reached.

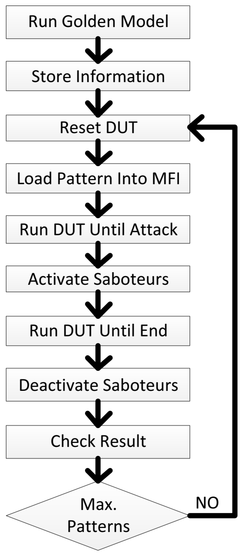


Fig. 6. Flow diagram shows a typical fault injection run

This timeout is required because the device could run into an infinite loop after the fault injection.

After the fault injection run is finished the saboteurs are deactivated and the result is compared to the results from the golden model run. If the results are not equal a fault analysis process has to be launched. This procedure is continued until all attack patterns are tested.

The fault pattern generator creates attack patterns for the fault injection. In this example only a random number generator is used as pattern generator to show the function of the fault MFI. But because the patterns are not generated directly in the MFI the complexity of such a pattern generation does not increase the required slices of the MFI.

IV. Experimental Results

To prove the effectiveness of our approach we chose to implement the proposed controller on a widely used platform,the LEON3 SPARC V8 conformant processor of Gaisler Research [9].

The test setup is implemented on a Virtex5 FPGA using the Xilinx ML507 evaluation board. Simulation results are gained using the Modelsim software which is part of the ISE software package provided by Xilinx.

The fault injection setup is configured as shown in Table II.

A. LEON3 platform setup

The LEON3 is configured for a single-core configuration using default platform settings for this particular evaluation board (ML507). The MFI is configured via GPIO and can be accessed using the PowerPC.

TABLE II
Fault injection setup

Saboteur type	Fault mode	Fault target type	Fault injection targets
Single-bit	Bit-flip	Control logic	Integer pipeline Cache controller Register file Multiplier unit Divider unit

TABLE III
Simulated fault injection performance

Saboteurs	Inj. Patterns	Time [sec]	Patterns/sec.
8	256	274	0.93
16	256	282	0.91
24	256	286	0.90
32	256	299	0.86

B. Saboteur configuration

The random approach of fault injection proposed in this publication allows to find global dependencies between local fault occurrences. Of course, with rising number of saboteurs possible combinations rise until an analysis of the results is not possible in a reasonable time frame. Therefore a small number of saboteur locations has been chosen for these fault injection experiments.

C. Principle simulation results

To ensure the correct behaviour of the fault injector and its saboteur, the whole LEON3 system has been thoroughly simulated using the ModelSIM software package. It has to be mentioned that the simulation includes not only the fault injection process but also start and calculation instructions. Results of these injection campaigns are shown in Table III.

The pattern injection rate is constant for patterns smaller than 32, because of the working principle of the MFI. For larger patterns additional GPIO transfers are required. However with increasing pattern size the amount of concurrently injected faults rises accordingly.

D. VHDL Synthesis Results

To estimate the necessary FPGA area requirements a synthesis of a typical platform configuration has been performed. The results of several synthesis runs using different saboteur configurations is shown in Table IV.

It can be seen that the MFI and its saboteurs has a negligible effect on the size of the resulting synthesized design. Basing on this routed results, it can be expected that it should be possible to implement a large amount of saboteurs on this FPGA-platform.

E. Fault Injection Performance

In this subsection results of several fault injection campaigns are presented to show that higher structural flexibility does not come with disadvantages concerning emulation speed. It also

TABLE IV
VHDL SYNTHESIS RESULTS

Saboteurs	Look-up-tables	Overhead[%]	Slices	Overhead[%]
0	14897	-	10423	-
8	14950	0.36%	10513	0.86%
32	15107	1.41%	10642	2.10%
96	15194	1.99%	10716	2.81%
170	15244	2.33%	10774	3.37%
326	15478	3.9%	11088	6.34%

TABLE V
EMULATED FAULT INJECTION PERFORMANCE

Saboteurs	Inj. Patterns	Time [sec]	Patterns/sec.	Pattern Blocks/sec.
8	100M	46.76	2.17M	2.17M
32	100M	46.76	2.17M	2.17M
96	100M	114.48	873.5k	2.17M
170	100M	156,4	639.4k	2.17M
326	100M	282.16	354.4k	2.17M

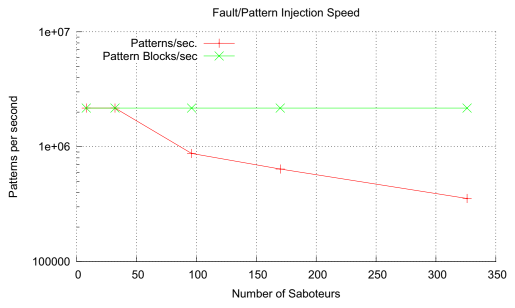


Fig. 7. Speed of the fault injection depending on the number of saboteurs

needs to be mentioned that such a multi-bit injection campaign leads to complex results in system behavior and therefore the analysis modules will most likely be the slowest link in the emulation chain. The results are summarized in Table V. It can be seen that the number of saboteurs has an influence on the number of patterns which can be injected per second. The reason is that the GPIO interface only support the transmission of 32 bits of the pattern simultaneously. If the size of the pattern is increased the time for the pattern transmission is increasing. The last column shows that the transmission of one pattern is interdependent to the number of saboteurs. It is not required to send for each attack the whole pattern. If only one pattern block has to be changed only one GPIO transmission is required. But not always only the maximum number of faults is important because not all saboteurs are activated. The more important numbers are the patterns injected per second. The speed of the pattern injections per seconds is not independent of the number of saboteurs. But the transmission of on fault pattern is constant. This is presented in Figure 7.

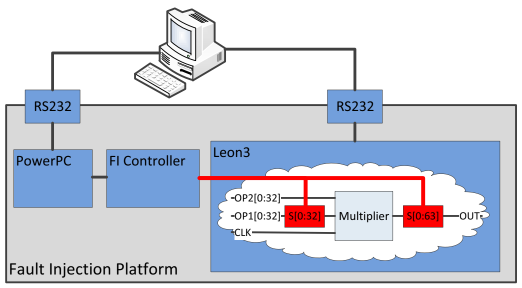


Fig. 8. Overview of the test environment.

F. Case Study: Attack on RSA Authentication

The MFI is used to show a case study of a fault attack on RSA authentication [24]. The RSA authentication process signs a message with a private key. To generate such an signature the RSA algorithm uses multiplications. In [24] it has been concluded that fault attacks on the multiplication unit can be used to attack the RSA algorithm. For the described fault attack the authors assume that only one bit of the multiplier output switches. In this case study we show that our approach can emulate exactly such fault attacks.

For this test we chose the LEON3 processor in single core configuration. For the attack target the output signals and one of the operand registers of the multiplier have been chosen.

An overview of the test environment is given in Figure 8. Listing 1 shows pseudo code describing the fault injection flow. The saboteur placement methodology is described in [21]. The FPGA-internal PowerPC is running a Linux kernel to allow for programming the fault injection flow in an comfortably way.The fault injection control software is running on the PowerPC. The host PC evaluates the results of the fault injection. During this attack the saboteurs are configured in bit-flip mode.

```
run_golden_model();
store_all_information();
for(i=0;i<n_patterns;i++){
        reset_dut();
        load_pattern(pattern[i]);
        run_DUT(pattern_time[i]);
        activate_saboteurs();
        run_DUT(max_time);
        deactivate_saboteurs();
        check_result();
}
```

Listing 1. Pseudo code of the fault injection flow

In Table VI and Figure 9 the results are shown. As seen the MFI can emulate a fault attack on the multiplier interface. Because the MFI supports a large amount of saboteurs it is possible to place the saboteurs not only at the output. With this it is possible to test also the effect of a fault injection to the input of the multiplier. As assumed, attacks to the output of the multiplier can be used to emulate attacks as shown in [24]. Attacks to the input of the multiplier mostly disturb more than one output bit. But in [24] the authors assume for there attack that only one bit of the multiplication has to be disturbed.

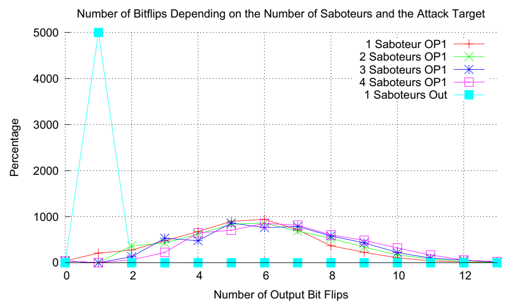


Fig. 9. Influence of saboteurs on the multiplication unit. Bit-flips of the output depending on the number of active saboteurs at the input/output.

TABLE VI
NUMBER OF FAULTS DETECTED DEPENDING ON THE ATTACK PATTERN

Attack Target	OP1	OP1	OP1	OP1	Out
Number of Saboteurs	1	2	3	4	1
Nr. of Bit-flips					
0	37	37	37	37	0
1	208	0	0	0	5000
2	275	365	129	59	0
3	477	428	532	224	0
4	676	624	478	646	0
5	899	836	862	704	0
6	942	862	762	852	0
7	714	686	787	817	0
8	369	529	580	604	0
9	226	337	432	487	0
10	105	170	221	319	0
11	40	82	99	167	0
12	20	22	56	58	0
13	9	13	13	24	0

G. Comparison to existent fault injector solutions

During the last years many high performance fault injection emulation platforms have been published. While fault injection speed is only one parameter to quantify the performance of a fault injector solution, it is important to measure how many faults can be injected in a reasonable time. This is especially important for complex system-on-chip designs with a high amount of interesting injection points. In [6] fault injection campaigns using different benchmark configurations have been compared to typical simulation solutions. To cover all considered design approaches also reconfiguration techniques are presented in [12]. This publication shows two implementations, one using autonomous emulation (AE) and one using partial reconfiguration (PR). The test object in this case is the freely available CORDIC16 core.

It has to be mentioned that our fault injector solution profits from a higher clock frequency and state-of-the-art FPGA hardware. Another difference that has to be considered is that because of the pattern approach the fault injection rate is not constant. Therefore the total number of activated faults has been calculated, based on the number of activated faults per turn. For the evaluation the worst case is assumed, that only one fault is injected per pattern. If the simultaneous injected faults are increased the number of faults per seconds will be increased (one turn consists of all possible bit combinations of the selected fault pattern width). The results of these investigations are compared to our result in Table VII. Because of the different test systems the values have to be normalized. The last column shows the fault injection speed normalized to a clock cycle of 25 ns.

TABLE VII
FAULT INJECTION PERFORMANCE COMPARED TO PREVIOUS RESULTS

Approach	Clock cycle	Inj. Faults	Time	Inj. Speed	Nor. Inj. Speed
	[ns]		[s]	1/[s]	1/[s]
[6]	50	100k	83	1205	603
[12] AE	16.8	129.75M	698	185888	276619
[12] PR	18.97	10k	1014	9.86	13
[14] DPR	10	-	-	12987	32468
[14] PRR	10	-	-	414.94	1037
This work	25	100M	46.76	2.17M	2.17M

The comparison shows that the in-system solution does not suffer from additional performance penalties. Strong spacial containment of possible fault locations leads to very high injection results if combined with large fault injection patterns.

It can also be concluded that autonomous emulation provides injection speed advantages compared to partial reconfiguration techniques. The reconfiguration approach suffers not only from limiting communication interfaces but also from long delays caused by the reconfiguration process. This also does not include the time needed to generate reconfigurable sub-blocks. These problems are targeted by the work presented in [14]. In this publication partial reconfiguration is compared to direct reconfiguration using an FPGA internal port to its configuration memory. While reconfiguration speed is greatly improved, it is still slower than autonomous emulation solutions like the one proposed in this paper.

V. CONCLUSION

This paper presents a highly modularized controller solution for portable fault injection systems. It has been shown that these fault injection campaigns can be executed very efficiently through a generalized interface using a high level abstraction of physical fault sources. The design is scalable to allow both, fully automated campaigns with a larger fault pattern memory or more user controlled campaigns using a small silicon footprint. The performance is comparable to existing platforms using circuit manipulation and significantly faster than ones using partial and complete FPGA reconfiguration.

This is the first step towards a complete fault injection platform for dependability and security evaluations. Its flexible design will allow the test-engineer to shift the focus from complex testing architectures to more complex fault models. This should finally allow to not only model simple SEUs but also complex fault attack scenarios. Through the additional abstraction level a better separation of the test and design engineer's tasks is provided. The gained knowledge of these experiments will be used to get a better understanding of inner-chip fault mechanics. Such full scale investigations need fully automated saboteur injection techniques which are currently under development.

The consideration of such attacks will be necessary to design efficient and secure smart card systems. This is also valid for highly integrated systems or systems under high environmental stress.

ACKNOWLEDGMENT

The authors would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology, which funded the Power-Modes project under the FIT-IT contract FFG 825749. We would also like to thank our project partners Infineon Technologies Austria AG and Austria Card GmbH.

REFERENCES

[1] M. Nicolaidis, "Time redundancy based soft-error tolerance to rescue nanometer technologies," in *Proc. IEEE VLSI Test Symposium (1999)*, 1999, pp. 86 –94.

[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*, vol. 94, no. 2, pp. 370–382, feb 2006.

[3] D. D. Andres, J. C. Ruiz, D. Gil, and P. Gil, "Fades: a fault emulation tool for fast dependability assessment," in *Proc. IEEE Int. Conf. Field Programmable Technology (FPT 2006)*, 2006, pp. 221–228.

[4] Xilinx, "Virtex-5 fpga family," Online, 07 2010. [Online]. Available: http://www.xilinx.com/products/virtex5/index.htm

[5] J. Boue, P. Petillon, and Y. Crouzet, "Mefisto-l: a vhdl-based fault injection tool for the experimental assessment of fault tolerance," in *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, 1998. Digest of Papers.*, 23-25 1998, pp. 168 –173.

[6] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, and M. Violante, "An fpga-based approach for speeding-up fault injection campaigns on safety-critical circuits," *Journal of Electronic Testing*, vol. 18, no. 3, pp. 261–271, 2002, 10.1023/A:1015079004512. [Online]. Available: http://dx.doi.org/10.1023/A:1015079004512

[7] A. Benso and B. Prinetto, *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*. Kluwer Academic Publishers, 2003.

[8] P. Ellervee, J. Raik, K. Tammemae, and R.-J. Ubar, "Fpga-based fault emulation of synchronous sequential circuits," *IET Computers & Digital Techniques*, vol. 1, no. 2, pp. 70–76, 2007.

[9] Gaisler, "Leon3 processor," Online, 07 2010. [Online]. Available: http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53

[10] A. Pellegrini, K. Constantinides, D. Zhang, S. Sudhakar, V. Bertacco, and T. Austin, "Crashtest: A fast high-fidelity fpga-based resiliency analysis framework," in *Proc. IEEE Int. Conf. Computer Design (ICCD 2008)*, oct 2008, pp. 363–370.

[11] L. Antoni, R. Leveugle, and M. Feher, "Using run-time reconfiguration for fault injection in hardware prototypes," in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT 2002)*, 2002, pp. 245–253.

[12] C. Lopez-Ongil, L. Entrena, M. Garcia-Valderas, M. Portela, M. A. Aguirre, J. Tombs, V. Baena, and F. Munoz, "A unified environment for fault injection at any design level based on emulation," *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 946–950, 2007.

[13] M. Jeitler, M. Delvai, and S. Reichor, "Fuse - a hardware accelerated hdl fault injection tool," in *Proc. SPL Programmable Logic 5th Southern Conf*, 2009, pp. 89–94.

[14] L. Kafka, "Analysis of applicability of partial runtime reconfiguration in fault emulator in xilinx fpgas," in *DDECS '08: Proceedings of the 2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 1–4.

[15] B. F. Dutton, M. Ali, C. E. Stroud, and J. Sunwoo, "Embedded processor based fault injection and seu emulation for fpgas," in *Proc. International Conf. on Embedded Systems and Applications 2009*, 2009, pp. 183–189.

[16] L. Kafka, M. Danek, and O. Novak, "A novel emulation technique that preserves circuit structure and timing," in *International Symposium on System-on-Chip, 2007*, 20-21 2007, pp. 1 –4.

[17] R. Leveugle, "Early analysis of fault-based attack effects in secure circuits," *IEEE Transactions on Computers*, vol. 56, no. 10, pp. 1431–1434, 2007.

[18] R. Leveugle and K. Hadjiat, "Multi-level fault injections in vhdl descriptions: Alternative approaches and experiments," *Journal of Electronic Testing*, vol. 19, no. 5, pp. 559–575, 2003.

[19] J. Baraza, J. Gracia, D. Gil, and P. Gil, "Improvement of fault injection techniques based on vhdl code modification," in *Tenth IEEE International High-Level Design Validation and Test Workshop 2005*, 30 2005, pp. 19 – 26.

[20] J.-C. Baraza, J. Gracia, S. Blanc, D. Gil, and P.-J. Gil, "Enhancement of fault injection techniques based on the modification of vhdl code," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 6, pp. 693 –706, jun. 2008.

[21] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, and J. Haid, "Automatic saboteur placement for emulation-based multi-bit fault injection," In Press.

[22] C. Pohl, R. Fuest, and M. Porrmann, "vmagic – automatic code generation for vhdl," *newsletter edacentrum*, vol. 2, pp. 7–10, Jul. 2010.

[23] C. Bachmann, A. Genser, C. Steger, R. Weiss, and J. Haid, "Automated power characterization for run-time power emulation of soc designs," in *Proc. 13th Euromicro Conf. Digital System Design: Architectures, Methods and Tools (DSD)*, 2010, pp. 587–594.

[24] A. Pellegrini, V. Bertacco, and T. Austin, "Fault-based attack of rsa authentication," in *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2010, pp. 855–860.

Automatic Saboteur Placement for Emulation-Based Multi-Bit Fault Injection

Johannes Grinschgl, Armin Krieg,
Christian Steger, Reinhold Weiss
Institute for Technical Informatics
Graz University of Technology, Austria
Email: {johannes.grinschgl, armin.krieg,
steger, rweiss}@tugraz.at

Holger Bock, Josef Haid
Design Center Graz
Infineon Technologies Austria AG, Austria
Email: {holger.bock, josef.haid}@infineon.com

***Abstract*—During recent years the dependability and security requirements of system-on-chip (SoC) designs have increased tremendously. Both, dependability and security, domains are concerned with operational faults of a random or intentional nature. In former case random faults e.g. caused by radiation or degradation effects could lead to execution errors with possible dramatic results. The security domain is more concerned with intentional faults injected by an adversary during a physical attack to drive the system into an unintended state. The resistance of such a design against faults can be emulated during early design phases using fault injection methods. For these methods the design-under-test is augmented with additional circuitry to emulate faults at predestined locations. One method uses saboteurs, elements that are transparent during normal operation and faulty if activated, are placed into the target system. If this placement process includes a high number of saboteurs, the hardware description manipulation could be a challenge for the design engineer.**

Therefore this paper presents an automatic placement methodology for fault injection evaluations using saboteur techniques. The automatized process allows for the efficient placement of large amounts of saboteurs. This enables the designer to evaluate a high number of different dependability and fault attack scenarios during early design phases using FPGA-based functional emulation. Selected case studies show how this approach can be applied to a common general purpose architecture in an efficient way.

I. Introduction

Dependability and security requirements of system-on-chip (SoC) have increased over the last years. Currently most of these requirements are either ensured by tests after the first samples of a new product are available or by lengthy and computational intensive simulation runs. To improve a designer's ability to detect possible flaws and weaknesses in a target design a wide variety of hardware-accelerated emulation techniques has been introduced. These techniques allow to functionally test a given hardware description on a FPGA-based platform. Detailed evaluations of dependability and security features rely on a large amount of tested fault scenarios. Designer of applications in security critical environments have to consider the possibility of fault attacks, the deliberate introduction of operational faults by an adversary. Similar to the security domain, also in the dependability domain systems have to be tested for their fault resistance to ensure correct

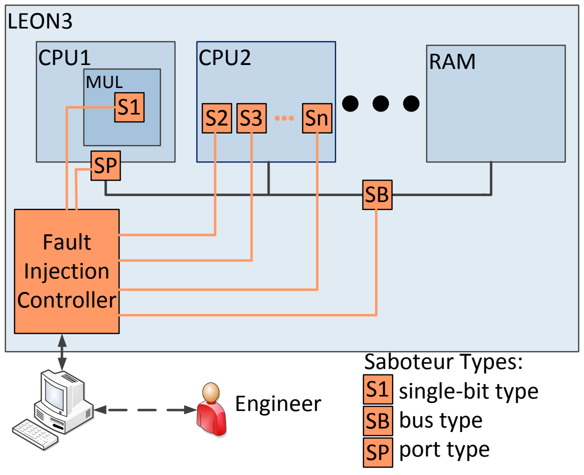


Fig. 1. Fault injection method based on the novel automatic saboteur placement method

behavior in case of faults. Depending on the application field fault could lead to life threating errors or loss of expensive equipment. Contrary to faults caused by an attack scenario in this case these are of a random nature.

A good overview of possible attack scenarios in the security domain is given by [1] and [2]. Not only the type of an attack is important, also which area of a system is manipulated. Another critical characteristic of such an attack is of importance, the attack timing. This results out of the fact that with increasing complexity of SoCs the number of possible parallel fault attacks is increasing.

An overview of the most common faults tested for in dependability systems are described in [3].

In this paper an automatized fault injection emulation method is presented which can be used for dependability and security evaluations. This approach is based on an automatic saboteur placement method (see Figure 1). Through this approach the following disadvantages of hardware accelerated fault emulation approaches are targeted:

- Complex fault scenarios result in high VHDL code adaption times
- In case of manual adaption the process can prove to be

error prone
- In case of manual adaption only a small number of saboteurs can be added

The advantage of high emulation speeds is not affected. The speed-up of the saboteur placement allows for the addition of saboteurs at an early design phase, because the effort of the saboteur placement is significantly reduced. The saboteur fault injection approach allows for evaluations at very early design stages. Correction of design weaknesses at an early phase results in significantly reduced redesihn costs [4].

An overview of the related work is given in Section II. In Section III an overview of the used fault injection method and the main ideas behind this approach are shown. Detailed information about the structure of the saboteur placement method is given in Section IV. Section V shows all required steps and hints to reproduce this flow. And finally, Section VI shows the results of an application of this saboteur placement flow to a common general-purpose architecture.

II. Related Work

Fault injection methods can be split into three main groups: simulation, physical and emulation groups.

A. Fault Injection Simulation

The most common method used for fault injection is the simulation approach. Compared to the emulation approach design simulation requires an extensive amount of time as shown [5]. The advantage of this method is its flexibility and the lacking need of hardware description adaption. An early example of this approach is shown in [6]. This paper presents a flow using simulation commands to run a full fault injection simulation.

Also for simulation approaches hardware-description adaption can yield its advantages. One of the first fault injection methods using such an approach was the MEFISTO tool [7], [8]. This tool is used to place saboteurs which can be controlled by the fault injection process computed by the simulating computer. As similar approach has been introduced in [9]. Rothbart et al. presented fault injection methods for system level description in [10]. Their approach adds fault injection modules between the described components to simulate fault attacks. In [11] cdifferent fault injection simulation approaches are described.

B. Fault Injection Physical Method

The second method group includes techniques to physically inject faults into a given system. This requires exact knowledge of the device and depending on the performed tests specialized equipment may be required [12]. Another big disadvantage of this method is that is requires a physical implementation of the target device. Therefore if the test results show that the device does not fulfill the given security or dependability standards, a redesign may be required. At such a late design stage the correction of design weaknesses result in high redesign costs.

C. Fault Injection Emulation

To cope with the problems of long simulation time and expensive physical testing hardware accelerated emulation techniques are preferred. One method of FPGA-based fault emulation is to use partial reconfiguration abilities of modern SRAM FPGAs. Currently this technique is slow, but very flexible. It allows the test engineer to run specific tests without high time overhead. Examples for this method can be found in many publications like [13]–[16]. A good overview over the partial reconfiguration concept itself is given in [15].

In [17] an approach is described where a so called Fault Injection Hardware Unit (FIHU) is placed into a FPGA design. This unit is used for controlling the partial reconfiguration process of the Virtex II FPGA. The unit requires only very little FPGA resources and its only disadvantage is that this approach is slower than a fault injection process using saboteurs [15].

Another approach is based on the adaption of the design hardware description. This is realized using so called saboteur or mutant blocks [18]. Saboteurs are modules which are placed into signal lines to disturb these signals in a defined way at predetermined times. The injectable faults can be divided into transient and permanent faults. These fault types can again be split into subtypes like stuck-at errors, time delayed, bit flips and many others.

The fault injection process using saboteurs represents a very fast emulation method. The main disadvantage is its reduced flexibility compared to simulation methods because of the required augmentation of the description and re-synthesis of the SoC design.

In [18] also the possibility of placing saboteurs automatically is discussed in theory without a practical design application. Leveugle et al. presented a method how to automatically place mutants into a VHDL design [19]. The difference between saboteur and mutant approaches is the different level of flexibility. Mutants are pre-manipulated modules that work normally if not activated but behave like a defective version of the module used for manipulation if activated. In [20] a VHDL analyzer tool called HSECT is used to find attack relevant regions for fault injection simulation. The authors use this information only for a simulation based fault injection method. In [21] a method is shown how to route automatically signals out of a VHDL design for power emulation purposes, but this approach is not reversible.

III. Fault Injection Method

First a motivation for the automatic fault injection method is given to show the need for a automatic fault injection process. In several publications different methods are shown how a system can be specifically influenced using fault injection. The following two examples show how an automatic fault injection method can alleviate fault injection evaluations.

Examples of use on AES Fault Attack: In [22] an attack on the AES cryptographical algorithm is shown. For this attack a ultra violet light source is used to clear parts of the SBOX of the microcontroller-based AES implementation. After the attack a readout of the SBOX results into a byte of

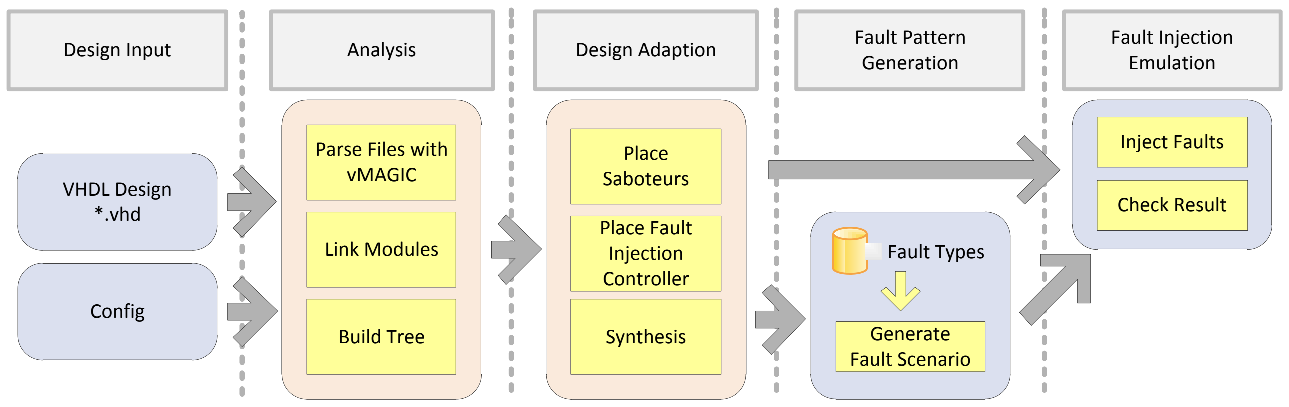


Fig. 2. Overview of the fault injection flow

which all bits are set to one. To check if an implementation is resistant to such an attack the designer has to evaluate how the system would react to such a manipulation. Using the approach presented in this publication a high amount of saboteurs can be efficiently placed into the targeted system during design phase. These saboteurs allow for fast emulation of possible signal manipulations to evaluate how the design reacts to different fault attacks. In this case only the data interface to the EEPROM has to be disturbed for fault emulation.

Examples of use on Network-on-Chip Switches: Network-on-chips (NOC) have to be tested for their resistance to single-event-upsets (SEU) and crosstalk effects. In [23] a method is shown how these dependability tests can be performed by showing that a SEU emulating fault injection can be easily implemented by adding only a controllable fault injection register. This functionality can also be introduced by our chosen saboteur approach. The main advantage of our approach compared to this register-based one is that we can place saboteurs automatically into the design keeping a high level of test flexibility.

After this motivation a short overview of the whole fault injection method is given to facilitate the understanding of the automatic saboteur placement flow. Figure 1 presents the structure of the fault injection methodology. This fault injection technique is based on the automatic adaption of the VHDL hardware description of a design-under-test. The required adaptations of the description consist of the routing of signals to the top-level module, the placement of saboteurs and the placement of the fault injection controller.

Figure 2 depicts the fault emulation method. This methodology can be divided into five blocks:

- **design input:** The input of this fault injection emulation method is a VHDL description of a system-on-chip.
- **analysis:** The analysis step parses the VHDL design and generates a tree representation containing the structure of the design.
- **VHDL design adaption:** The adaption of the VHDL design augments the VHDL code with saboteurs and a fault injection controller. The last step of the code augmentation contains the synthesis of the fault injection emulator.
- **fault pattern generation:** The fault pattern generator uses the information generated during the previous steps to generate attack scenarios.
- **fault injection emulation:** During the fault injection emulation the design is evaluated using pre-defined fault patterns.

A. *Saboteurs*

Saboteurs are modules to manipulate signal states in a defined manner. According to [24] several different fault effects implemented by such saboteurs can be described:

- single event upset (SEU)
- stuck-at 0
- stuck-at 1
- open-line
- delay
- indetermination
- stuck-open

A simple saboteur structure is preferred to keep circuit overhead low and to not unnecessarily slow down synthesis. More complex fault scenarios can be achieved by the combination of different simple fault models. Figure 3 shows a more detailed view the saboteur functionality.

B. *Fault Injection Controller*

For the efficient control of the placed saboteurs a fault injection controller is required. This controller uses fault patterns to define time and location of the selected fault scenario. See [25] for more details.A fault pattern has at least to contain the following information:

- Fault type (stuck-at, bit-flip, . . .)
- Saboteur selection
- Fault timing

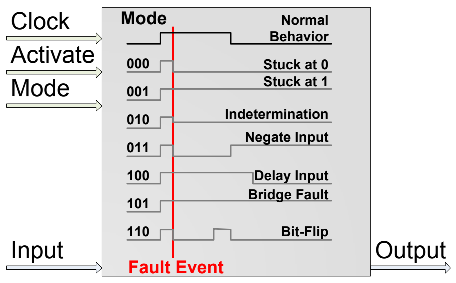


Fig. 3. Saboteur functionality (adapted from [25])

C. Fault Patterns

A fault pattern is a high-level representation of a specific fault scenario. While some scenarios only affect a limited local area like a fault using a fine laser, other affect large parts of the whole system. In latter case a large amount of saboteurs would have to be spread into the design. It is only necessary to place saboteurs on security-relevant or dependability-relevant signals. As seen in Equation 1 the number of different test scenarios depends on the number of saboteurs. But a blind increase of the number of saboteurs does not always directly increase the quality of the security or dependability evaluation.

$$N_{patterns} = (2^{N_{saboteurs}} - 1) \tag{1}$$

To maximize the test coverage against faults it is required to manipulate all security- and dependability-relevant signals. To route these signals manually is a very time-consuming and error-prone process. The automatic saboteur routing process allows to add a large number of saboteurs in a efficient manner.

IV. Novel Automatic Saboteur Placement Method

The automatic fault injection flow can be split into three main parts (see Figure 4). The first part consists of a VHDL parsing step. This step is required to get an overview of the design structure. The second (signal selection) step requires user interaction because the user has to define which signals are relevant. Finally the tool flow is used to automatically adapt the VHDL hardware description. During this adaptation process saboteurs and a fault injection controller is added to the system. This automatized description adaption process results into a fully synthesizable hardware description.

A. Parsing/Analysis of the VHDL Code

Figure 5 gives an overview over the parsing and analysis methodology. This method is used to generate a tree which represents the structure of the hardware description. The generated representation of the SoC design is required because a VHDL design can be split into multiple modules partially divided into different files. The advantage of this general representation is that all files have to be parsed only once to detect inter-module dependencies. The whole methodology was implemented for VHDL designs but it should be easily portable to Verilog or any other hardware description language. This process is split into three steps:

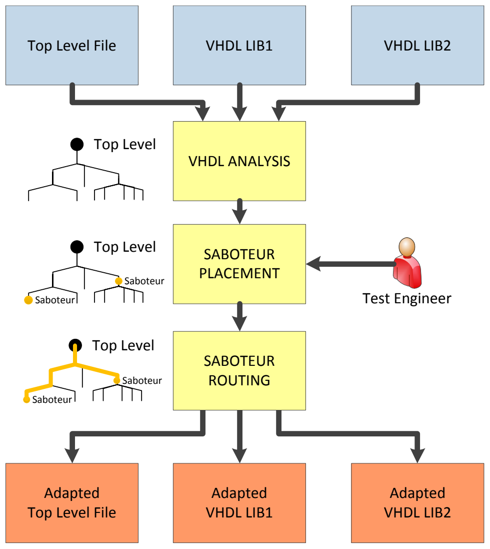


Fig. 4. Saboteur placement flow

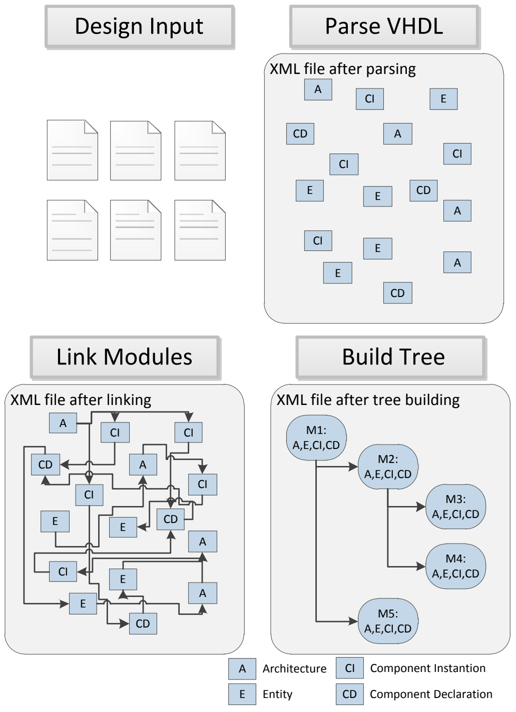


Fig. 5. Method for parsing, analysis and tree generation of the VHDL code

- parsing the VHDL files
- linking of detected modules
- building of a tree representation

B. Saboteur Placement

The saboteur placement process requires interaction by the design engineer. At the moment no automatic process for localizing security relevant regions has been implemented. Another alternative would be to place saboteur modules into

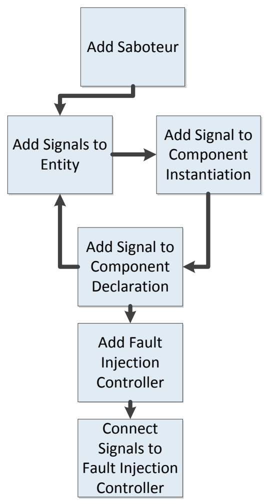

Add Saboteur
Add Signals to Entity
Add Signal to Component Instantiation
Add Signal to Component Declaration
Add Fault Injection Controller
Connect Signals to Fault Injection Controller

Fig. 6. Saboteur placement method

every available signal. Caused by the tremendously increased management effort this approach would be inefficient because of the enormous number of signals in even small designs. But it would be possible to place saboteurs to every signal of a critical module in the design (e.g. AES block). To place the saboteurs the test engineer uses a GUI, which represents the design as an expandable tree where the modules, its ports and all signals are listed. This allows the test engineer to locate the required signals quickly. Signal selection criteria could include the following:

- Dependability relevance of the signal
- Security relevance of the signal
- Influence of the signal on the power profile (allow differential power analysis (DPA) [26])
- Clock signals
- Critical signals of fault detection blocks
- Critical signals of fault recovery blocks
- Memory interfaces
- Register ports
- Data and address buses

C. Saboteur Routing

The saboteur routing process adapts the VHDL code using the saboteurs and the connection to the fault injection controller. Figure 6 shows the flow of the routing method. This process requires to modify the whole design, therefore the control signals of the saboteurs have to be routed to the top level module. This method adds the control signals of all included saboteurs to the interface of all parent modules. These signals are connected directly to the fault injection controller which is automatically placed at the top level module. The only adaption to the code which can not implemented automatically is the connection of the controller to the PC. The reason why the interface to the PC cannot be implemented automatically is because this flow should be independent of the FPGA and the environment. When the interface to the PC is implemented once it can be added to the flow. After adding the interface to the tool the fault injection controller can be connected automatically.

V. Implementation

This section describes the implementation of the saboteur placement concept. The tool is implemented using Java to ensure platform independence.

A. Parsing/ Analysis of the VHDL Code

The parsing and analysis task is one of the most important steps of the whole saboteur placement process, because every one of the following steps relies on the generated tree. The process itself is split into three tasks. First the parsing of the VHDL files. The second task consists of linking the generated information, followed by the tree building task.

1) Parsing of the VHDL Files: As described earlier during the parsing step the VHDL design is analyzed to extract inter-module dependencies. This step is based on the vMAGIC VHDL parser generator [27]. The parser generates a Java class-tree of the VHDL file. To get all required information about the hardware description only this representation has to be analyzed. The vMAGIC tool flow only supports the parsing of one file. Therefore some intermediate steps are required to retrieve a representation of the whole VHDL project. At first all information has to be extracted from the VHDL hardware description.

- library declarations
- ”use” clauses
- entities
- component declarations
- architectures
- component instantiations
- signals
- ports
- generic maps
- ”generate” statements
- constants

The information of all VHDL files is stored into an XML file. This file contains all key facts of the VHDL files plus additional information such as file name, library name, path and so on.

This intermediate step is required to ensure system independency of the saboteur placement process. The parsing of the VHDL files is the most time-consuming process of the whole saboteur placement flow. Only using this intermediate step large VHDL designs are handleable. The holding of all VHDL files opened would result in a significantly increased memory consumption of the program. This approach reduces the number of simultaneously opened files to one. Also runtime is reduced by this approach because the parsing of the required information from the XML file is faster than the parsing of the VHDL file. An additional advantage is that for the following steps only this XML file is required and only at the very last step, the VHDL files are required again.

2) Linking: In this context linking means to connect every architecture with its entity and every entity with its declaration. This is required because this information maybe split up into multiple files. This problem can be solved through the analysis of use clauses.

The linking concept is based on a recursive search algorithm. During the parsing process every architecture is linked to its entity. The entity is also linked to every component declaration. The component declaration is linked to the component instantiation.

3) Tree Building: The last step of the parsing/analysis flow is the building of a hierarchical representation of the complete SoC design. Starting at the top module all previously defined links are used to build a tree. VHDL supports "generate" statements which are used to make a design easy to configure. The configure parameters are often stored into a configuration file or can also be derived from generic maps. To generate the correct tree these generic statements have to be resolved. Therefore all generic maps, all constants and signal attributes have to be resolved. This requires a recursive solving function of constants. This implies that this step has to be done for every configuration of the design.

B. Saboteur Placement

The saboteur placement process is based on the interaction of a test engineer. It requires too much space on the FPGA to place a saboteur everywhere. For an easy placement of the saboteurs a graphical user interface (GUI) is implemented. This GUI presents the whole design using a tree structure. The test engineer only has to select the signal or port which have to be disturbed and specifies a name of this saboteur. All information is stored in the XML file. This allows for the repetition of this step as often as required to place all saboteurs. After all saboteurs are placed they have to be routed to the fault injection controller.

C. Saboteur Routing

During the saboteur routing step saboteurs are placed into the VHDL code structure and the control signals are routed to the top module. The interface of a saboteur mainly consists of the input port to which the target signal is connected. The second port is the output of the saboteur. This port represents the disturbed signal. The remaining ports of the saboteur are control signals. These signals are used to control the function of the saboteur. To route the signals to the top level module the vMAGIC tool is used.

To add this saboteur to the specified ports the following code blocks are required. This code sections are defined using a template file which can be easily adapted to other saboteurs. If a saboteur has been changed only the template file has to be modified.

```
library tugraz;
use tugraz.fi_leon3_package.all;
```

Listing 1. Required library and use clauses definition

An overview of the required adaptation to the VHDL files is given in the following text. The first one is the library and package definition (see Listing 1).

The control signals of the saboteurs must be routed to the top level module. Therefore the easiest solution is to add ports to all parent modules. An example is given in Listing 2.

```
FI_CLK : in std_ulogic := '0';
FI_SIGNAL_SELECT_MODE :
    in std_logic_vector(2 downto 0) := (others => '0');
saboteur_activate_FI_SIGNAL : in std_ulogic := '0';
```

Listing 2. Control signals which have to be added to the modules

The next step is to add a signal to the architecture. This signal must have the same type and size like the signal which should be disturbed. If the saboteur is added into a signal line the original signal is used as input and the new generated signal is used as output of the saboteur. An example is given in Listing 3.

```
signal attack_target_FI_SIGNAL : std_ulogic;
```

Listing 3. Signal which has to be added to the architecture

The last step is to add the saboteur itself to the architecture. An example is given in Listing 4.

```
sab003 : fi_saboteur_1_synwait
  port map (
    FI_CLK,
    attack_target,
    attack_target_FI_SIGNAL,
    FI_SIGNAL_SELECT_MODE,
    saboteur_activate_FI_SIGNAL
  );
```

Listing 4. Saboteur placement code

VI. Experimental Results

A. Test Environment

For the evaluation a ML507 board containing a Virtex5 XC5VFX70 FPGA is used. This FPGA already includes a PowerPC that is used for the communication to the host PC.

As a fault injection target we took the open source Leon3 SPARC V8 implementation from Aeroflex Gaisler [28]. Therefore results can be easily reproduced. The second reason was that the Leon3 can be configured easily by using a configuration interface. This allows us to test the saboteur placement concept using many different configurations. For our test we use a Leon3 in a single-core configuration. The test system for synthesis was an AMD X6 based PC with 3.2 GHz and 8GB RAM. The implementation of the saboteur placement tool is designed single threaded. This implies that only one core is used for the test.

B. Saboteur Placement

The test scenario was to add saboteurs using the automatic fault injection tool. The saboteurs are placed into the multiplier. The multiplier was used because in [29] a method is shown that assume that only one bit of the multiplication output of an RSA authentication must be attacked to extract the private key. Five files have to be adapted to place the saboteurs. For the placement process using the automatic

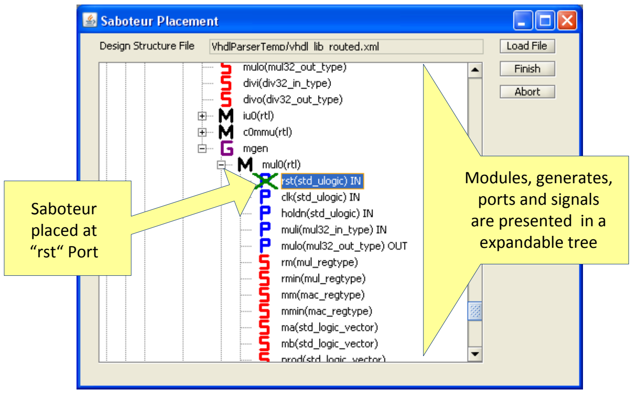


Fig. 7. The saboteur placement GUI allows the test engineer to specify the location of the saboteurs.

TABLE I
VHDL DESIGN ANALYSIS AND SABOTEUR PLACEMENT TIME

VHDL design analysis	
Parsing	48565ms
Linking	5373ms
Routing	16072ms
Sum	**70010ms**
VHDL design adaption time per saboteur	
Saboteur placement	20ms
Signal routing	180ms
Sum	**200ms**

saboteur placement tool the most time consuming step was to specify where the saboteurs shall be placed. But the very error-prone process of placing the saboteur should be eliminated. Figure 7 shows the graphic user interface (GUI) used to add the saboteurs to the VHDL design.

The normalized results are shown in Table I. It can be seen that for small numbers of saboteurs the analysis time is much higher than the pure placement time for the saboteur. But this analysis step must only be done once because all parsing information are stored in an XML file. This step must only be repeated when the architecture of the SoC design have been changed. The analysis processes can be run in parallel, because the VHDL files are independent to each other. This parsing step can be accelerated at least about 50% by optimization of the saboteur placement tool (multi threading). The selection of the signal which should be disturbed by the test engineer is the largest proportion of time required for the saboteur placement approach. This time is not filled into the table because it is extremely depending on the design and the knowledge of the test engineer. But if the designer know already where to place the saboteurs he can easily find the signal in the expendable tree.

C. Synthesis Time and Hardware Overhead

This section shows the differences in synthesis time of a system augmented with saboteurs compared to an unchanged

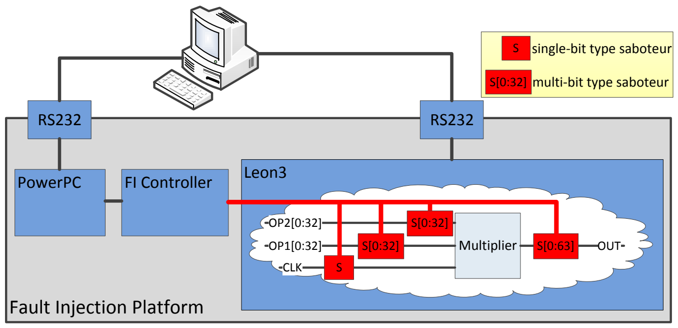


Fig. 8. Evaluation of the fault injection flow test setup (adapted from [25])

system. For an exemplary VHDL design a LEON3 processor in single-core configuration has been chosen. As seen in Table II the synthesis without saboteurs takes about 25min using our chosen evaluation system. Augmented with 50 saboteurs synthesis time is prolonged by only 30sec. The number of slices required for the FPGA implementation is increased only by three percent using 50 saboteurs.

TABLE II
SYNTHESIS TIME AND HARDWARE REQUIREMENTS DEPENDING ON THE NUMBER OF SABOTEURS.

Nr. Saboteurs	**0**	**10**	**10**	**50**
Routing method	n.a.	manual	auto.	auto.
Synthesis Time[s]^{a}	1536	1543	1543	1559
Synthesis Time OH^{a,b}**[%]**	-	0.46	0.46	1.5
Slices	14780	14950	14948	15172
Slices OH^{a}**[%]**	-	1.15	1.14	2.65

^{a}Executed on a 3.2 GHz AMD X6 system
^{b}Overhead

D. Evaluation of the Fault Injection Flow

The fault injection flow is used to show that the automatically routed saboteurs are working effectively. For this test we added saboteurs to a LEON3 processor in single-core configuration. The saboteurs are placed at the multiplier(see Figure 8). The multiplier is used because in [29] an attack on a RSA authentication process is shown by attacking the multiplication. To control the whole process we connected the fault injection controller to the PowerPC of the Virtex5-FXT FPGA-device. This PowerPC processor is used to run a Linux operating system that automatically connects to a host computer using an Ethernet interface. This host computer is used to control the fault injection flow. The LEON3 processor is used to run a program calculating multiplications of two random numbers every 100 us. The operands and result are then sent to the host computer using a RS232 interface. After activation of the saboteurs the multiplication was disturbed.

VII. CONCLUSION

In this paper an automatic fault injection emulation method based on automatic saboteur placement is presented. The automatized approach allows for the efficient augmentation of complex hardware designs in reasonable amounts of time. The shorter investigation cycles enable a system designer to implement test scenarios for more complex dependability and security evaluations at early design stages. The augmented hardware description can then used for efficient FPGA-based hardware accelerated emulation of the target design.

Its feasibility and efficiency is shown using case studies from the dependability and security domains. We showed how our selected approach can be applied to a common general-purpose architecture and that only a slight time overhead is introduced to the synthesis and evaluation flow.

Our future work will include the automation of the signal extraction process to support the design engineer with the selection of critical signals.

ACKNOWLEDGEMENT

The authors would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology, which funded the Power-Modes project under the FIT-IT contract FFG 825749. We would also like to thank our project partners Infineon Technologies Austria AG and Austria Card GmbH.

REFERENCES

[1] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, “The sorcerer’s apprentice guide to fault attacks,” *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, 2006.

[2] B. M. Gammel and J. Ruping, “Smart cards inside,” in *Proc. 31st European Solid-State Circuits Conf. ESSCIRC 2005*, 2005, pp. 69–74.

[3] H. R. Zarandi, S. G. Miremadi, and A. Ejlali, “Dependability analysis using a fault injection tool based on synthesizability of hdl models,” in *Proc. 18th IEEE Int Defect and Fault Tolerance in VLSI Systems Symp*, 2003, pp. 485–492.

[4] W. W. Peng and D. R. Wallace, *Software Error Analysis*. Summit, NJ, USA: Silicon Press, 1994.

[5] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, and M. Violante, “Exploiting fpga-based techniques for fault injection campaigns on vlsi circuits,” in *Defect and Fault Tolerance in VLSI Systems, 2001. Proceedings. 2001 IEEE International Symposium on*, 2001, pp. 250 –258.

[6] J. Guthoff and V. Sieh, “Combining software-implemented and simulation-based fault injection into a single fault injection method,” in *Proc. Twenty-Fifth Int Fault-Tolerant Computing FTCS-25. Digest of Papers. Symp*, 1995, pp. 196–206.

[7] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, “Fault injection into vhdl models: the mefisto tool,” in *Proc. Twenty-Fourth Int Fault-Tolerant Computing FTCS-24. Digest of Papers. Symp*, 1994, pp. 66–75.

[8] J. Boue, P. Petillon, and Y. Crouzet, “Mefisto-l: a vhdl-based fault injection tool for the experimental assessment of fault tolerance,” in *Proc. Digest of Papers Fault-Tolerant Computing Twenty-Eighth Annual Int. Symp*, 1998, pp. 168–173.

[9] R. Velazco, R. Leveugle, and O. Calvo, “Upset-like fault injection in vhdl descriptions: A method and preliminary results,” in *Proc. IEEE Int Defect and Fault Tolerance in VLSI Systems Symp*, 2001, pp. 259–267.

[10] K. Rothbart, U. Neffe, C. Steger, R. Weiss, E. Rieger, and A. Muehlberger, “High level fault injection for attack simulation in smart cards,” in *Proc. 13th Asian Test Symp*, 2004, pp. 118–121.

[11] Y. Torroja, T. Riesgo, E. de la Torre, and J. Uceda, “A comparative analysis of different fault simulation techniques for vlsi circuits testing,” in *Proc. IECON ’91. Conf. Int Industrial Electronics, Control and Instrumentation*, 1991, pp. 1226–1231.

[12] J. Karlsson and P. Folkesson, “Application of three physical fault injection techniques to the experimental assessment of the mars architecture.” IEEE Computer Society Press, 1995, pp. 267–287.

[13] S. Bayar and A. Yurdakul, “Self-reconfiguration on spartan-iii fpgas with compressed partial bitstreams via a parallel configuration access port (cpcap) core,” in *Proc. Ph.D. Research in Microelectronics and Electronics PRIME 2008*, 2008, pp. 137–140.

[14] P. Kenterlis, N. Kranitis, A. Paschalis, D. Gizopoulos, and M. Psarakis, “A low-cost seu fault emulation platform for sram-based fpgas,” in *Proc. 12th IEEE Int. On-Line Testing Symp. IOLTS 2006*, 2006.

[15] L. Kafka, “Analysis of applicability of partial runtime reconfiguration in fault emulator in xilinx fpgas,” in *DDECS ’08: Proceedings of the 2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 1–4.

[16] L. Sterpone and M. Violante, “A new partial reconfiguration-based fault-injection system to evaluate seu effects in sram-based fpgas,” *Nuclear Science, IEEE Transactions on*, vol. 54, no. 4, pp. 965 –970, 2007.

[17] M. Sonza Reorda, L. Sterpone, M. Violante, M. Portela-Garcia, C. Lopez-Ongil, and L. Entrena, “Fault injection-based reliability evaluation of sopcs,” in *Proc. Eleventh IEEE European Test Symp. ETS ’06*, 2006, pp. 75–82.

[18] J. C. Baraza, J. Gracia, D. Gil, and P. J. Gil, “Improvement of fault injection techniques based on vhdl code modification,” in *Proc. Tenth IEEE Int. High-Level Design Validation and Test Workshop*, 2005, pp. 19–26.

[19] R. Leveugle, “Fault injection in vhdl descriptions and emulation,” in *Proc. IEEE Int Defect and Fault Tolerance in VLSI Systems Symp*, 2000, pp. 414–419.

[20] W. Sheng, L. Xiao, and Z. Mao, “An automated fault injection technique based on vhdl syntax analysis and stratified sampling,” in *Proc. 4th IEEE Int. Symp. Electronic Design, Test and Applications DELTA 2008*, 2008, pp. 587–591.

[21] C. Bachmann, A. Genser, C. Steger, R. Weiss, and J. Haid, “Automated power characterization for run-time power emulation of soc designs,” in *Proc. 13th Euromicro Conf. Digital System Design: Architectures, Methods and Tools (DSD)*, 2010, pp. 587–594.

[22] J.-M. Schmidt, M. Hutter, and T. Plos, “Optical fault attacks on aes: A threat in violet,” in *Proc. Workshop Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2009, pp. 13–22.

[23] A. Eghbal, P. M. Yaghini, H. Pedram, and H. R. Zarandi, “Fault injection-based evaluation of a synchronous noc router,” in *Proc. 15th IEEE Int. On-Line Testing Symp. IOLTS 2009*, 2009, pp. 212–214.

[24] J. Gracia, J. Baraza, D. Gil, and P. Gil, “Comparison and application of different vhdl-based fault injection techniques,” in *Defect and Fault Tolerance in VLSI Systems, 2001. Proceedings. 2001 IEEE International Symposium on*, 2001, pp. 233 –241.

[25] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, and J. Haid, “Modular fault injector for multiple fault dependability and security evaluations,” in *DSD 2011*, In Press.

[26] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *Advances in Cryptology CRYPTO 99*, ser. Lecture Notes in Computer Science, vol. 1666. Springer Berlin / Heidelberg, 1999. [Online]. Available: http://www.springerlink.com/content/kx35ub53vtrkh2nx/

[27] C. Pohl, R. Fuest, and M. Porrmann, “vmagic – automatic code generation for vhdl,” *newsletter edacentrum*, vol. 2, pp. 7–10, Jul. 2010.

[28] Aeroflex Gaisler, “Leon3 processor,” 22. December 2010, www.gaisler.com/leonmain.html.

[29] A. Pellegrini, V. Bertacco, and T. Austin, “Fault-based attack of rsa authentication,” in *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2010, pp. 855–860.

Microprocessors and Microsystems xxx (2012) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Case study on multiple fault dependability and security evaluations

Johannes Grinschgl [a,*], Armin Krieg [a], Christian Steger [a], Reinhold Weiss [a], Holger Bock [b], Josef Haid [b], Thomas Aichinger [c], Christiane Ulbricht [c]

[a] *Institute for Technical Informatics, Graz University of Technology, Inffeldgasse 16/I, Graz, Austria*
[b] *Infineon Technologies Austria AG, Design Center Graz, Babenbergerstrasse 10, Graz, Austria*
[c] *Austria Card Plastikkarten und Ausweissysteme GmbH, Lamezanstrasse 4-8, Wien, Austria*

A R T I C L E I N F O

Article history:
Available online xxxx

Keywords:
Fault injection
Fault emulation
Multi-bit faults
Saboteurs

A B S T R A C T

The increasing level of integration and decreasing size of circuit elements leads to higher probabilities of operational faults. More vulnerable electronic devices are also more prone to external influence from energizing radiation. Additionally, the concerns of chip designers include not only the natural causes of faults but also the misbehavior of chips due to “planned” attacks, as, for example, in critical security applications. In particular, smart cards are exposed to complex attacks through which an adversary attempts to extract knowledge from secured systems by provoking undefined states. These problems increase the need to test new designs for their fault robustness.

This paper presents a case study on fault injection strategies. An in-system fault injection strategy for automatic test pattern injection by enabling the emulation of fault effects on the circuit level is introduced. Second, an approach is presented that provides an abstraction of the internal fault injection structures to a more generic high-level view. Through this abstraction, it is possible to help the operating system designer test a product against different fault effects without knowing how to produce this effect by a fault attack. Therefore, we implemented a modular fault injection controller that is located along with the system under test on the emulator platform.

1. Introduction

The continuing success of the semiconductor industry with regard to downscaling structures has led to highly integrated but also highly sensitive devices. External radiation effects and thermal and electric degradation have become common problems for the dependability of a system [1].

Through these effects, transient or even permanent faults are introduced, which leads to a change in the system behavior. These faults occur randomly, unlike faults resulting from so-called fault attack scenarios. In this case, an attacker deliberately injects faults into a system to change the system behavior. Without dedicated precautions, such attacks are easy to implement [2].

In recent years, intensive research has introduced several different tools to simulate or emulate possible fault scenarios during the design phase. In particular, fault emulation proved to be a very effective way of testing systems under the influence of fault sources. The platform usually used to emulate such a faulty system is a field programmable gate array FPGA because of its flexibility. An example of such an FPGA fault injection platform is shown in Fig. 1. There are different ways to inject faults into circuits. One is the use of partial reconfiguration features of the FPGA [3], but this design approach heavily limits the platform choice because there are only a few candidates, such as the Virtex families from Xilinx [4]. Another way is to instrument the given circuit either with manipulated logic elements or with integrated controllable fault elements. The latter case can distinguish between saboteurs and mutants [5]. Saboteurs are small circuit elements that do not affect the system behavior under normal conditions. If activated, they directly inject faults into the targeted submodule by disturbing the internal signals. To disturb signals, saboteurs must be placed between their source and their sink. Mutants are modified submodules that also do not affect system behavior under normal conditions but, if activated, behave like a faulty version of the original. To simulate or emulate fault attacks with mutants, the submodule must be replaced by a mutated submodule.

To accomplish such a test setup, it is necessary to have access to the hardware description or a standardized test interface. Such a test interface could consist of test chains [6]. Another important step in creating an effective fault injection platform is the selection of a proper fault model. For dependability evaluations, a single event upset (SEU) fault model is often sufficient. If the faults are caused by radiation or degradation, it can be safely assumed that only a single random fault will occur at a time [7]. In contrast, security evaluations consider intentional faults. Therefore, it is possible

* Corresponding author.
E-mail address: Johannes.Grinschgl@tugraz.at (J. Grinschgl).

0141-9331/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.micpro.2012.05.016

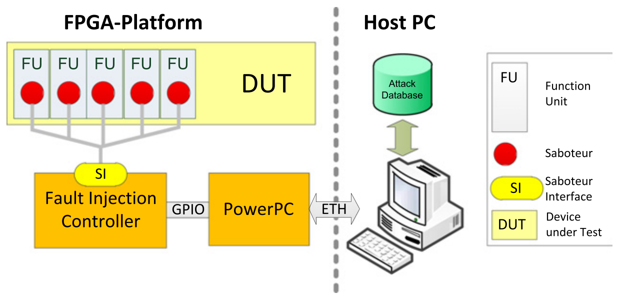


Fig. 1. Schematic view of the proposed fault injection system (obtained from [27]).

that an attacker introduces several faults at once to achieve the desired effect of secret exposure.

This type of multi-bit fault model results in much more complex fault relationships than SEU models because the time and space of such faults cannot be assumed to be random. This model must reflect the interdependency of several SEUs. Thus, in dependability analyses, similar types of sensitive sub-circuits will fail at the same time. In security evaluations, this dependency is defined by the attack scheme of the adversary.

Finally, the controlling element itself must be designed very carefully. The scope ranges from a very simple implementation completely controlled by an external source such a personal computer to very sophisticated integrated designs. Simple implementations have the advantage of being very adaptable to system changes, but they are also highly dependent on the communication interface used. Therefore, they can be considered slow. An implementation based on a PCI interface is shown in [6]. Sophisticated solutions allow for very high fault injection rates but are difficult to adapt to system changes [8]. In recent years, implementations of fault injection emulation controllers have been either simple and limited by slow interface performance or highly complex and therefore not portable from one design to another.

In this paper, we propose a modular fault injector (MFI) that combines the advantages of a simple portable design and fast, highly complex implementations. It also helps to provide a common platform for fault injection campaigns on different target architectures. Another point that has often been neglected in previous work in this field is the consideration of fault attacks.

Finally, the consideration of multi-bit fault patterns instead of simple SEUs can be seen as a main contribution of this work. This consideration is especially important for security evaluations. This fully synthesizable controller can also be used as an online-testing implementation if desired. The main goals of this work can be summarized as follows:

- Fault effect modeling for software development.
- Fully modular fault injector design.
- Multi-bit fault injection to support fault attack emulation.

This paper is structured as follows. Section 2 briefly describes the state of the art of work related to emulated fault injection using integrated controllers and different reconfiguration techniques. In Section 3, the design of the fault injection controller is described. Section 4 presents some of the experimental results of this MFI by means of the LEON3 processor [9]. Finally, conclusions are drawn in Section 5.

2. Related work

The introduction of fault emulation on FPGA platforms has led to several publications concerned particularly with the emulation of SEUs in space applications. Approaches using direct circuit manipulation such as the solution proposed in this paper must be distinguished from techniques using the reconfiguration features of certain FPGAs.

The former possibility can again be divided into two subcategories. One uses specialized hardware units to instrument saboteurs or mutants as shown in [8]. The second variant uses available processor cores for fault injection automation similar to the solution provided by [10]. This solution promises a high emulation performance because of fast on-chip communication channels, but it is more difficult to port to other platforms.

In recent years, the partial and complete runtime reconfiguration of FPGAs became a common technique to implement fault injection in a flexible way. Reconfiguration is possible on special FPGA series using an external communication interface to directly feed different fault configurations into the FPGA, as shown in [11–13]. If this external interface is too slow, it is also possible to use existing processor resources for the reconfiguration task, as presented in [14,15]. While the reconfiguration approach is tempting (and has been used by several research groups), it also limits the maximum reachable fault injection performance and the designer's platform selection. An approach similar to the reconfiguration approaches is presented in [16], in which the synthesized netlist is augmented to preserve the original structure of the system.

In the field of security evaluations, the authors of [17] used a proven fault injection platform presented in [18]. This investigation confirms the strong need for multiple fault injection campaigns in the design process. However, in their experiments, only a small fault multiplicity (six) was used to prove this point. In [19], a system demonstrating the use of multiple emulation platforms to run simultaneous fault injection campaigns is described. This parallelism increases the emulation speed.

3. Modular fault injector

For this case study, a fault injection method is required. Therefore, a modular fault injection controller concept and its implementation are presented in this section. The following requirements summarize the main drivers behind the development of this modular fault injector (MFI):

- Effect modeling for software development.
- Support for fault patterns to support multi-bit fault injection.
- Multi-mode saboteur support.
- Scalable interfaces to support automatic saboteur placement.
- Standardized communication interface.
- Internal memory for automatic fault injection.

To support the easy application of the fault injection system to a new design under test, a standardized communication interface is needed. The General Purpose Input/Output (GPIO) communication interface enables the developer to use the proposed controller in a wide selection of different architectures. The GPIO interface consists of pins that can be configured via software commands. These pins allow for controlling the MFI without disturbing the device-under-test. Extensive fault injection campaigns are only possible using a large number of active saboteurs. It is not possible to route such a large number manually; therefore, internal interfaces must be scalable to enable automated injection processes. An effective separation of the design and evaluator tasks can only be guaranteed through a generalized view of the fault injection system, which is accomplished using so-called fault patterns, high level representations of the underlying saboteur distribution. These

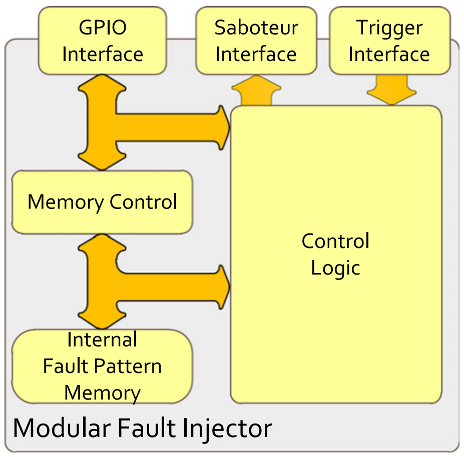


Fig. 2. Schematic view of the modular fault injector (adapted from [27]).

patterns are also necessary for efficient physical fault attack emulation.

High-speed automated operation is enabled through a flexible internal memory system. In the design phase of the proposed fault injector, future advances of the evaluation platform must be considered, which include support of large fault patterns and burst loading from the bus-system. Different fault and attack scenarios call for different saboteur types. To simplify the reconfiguration process, different saboteur modes are provided by a single flexible saboteur design.

A basic block diagram of the proposed fault injection system is shown in Fig. 1. The flow consists of the fault injection controller, which specifies the attack scenario; the saboteurs, which disturb signals; and a PowerPC, which is used as the interface to the PC. The PowerPC is an optional component because the interface of the MFI can be easily configured. As described in the following sections, the MFI is controlled over a GPIO interface. This interface can also be controlled from different devices, e.g., for Smart card fault emulation, and during the development process of the operation system, the MFI can be controlled with APDU commands.

3.1. Fault injection controller

The fault injection controller is the main part of the MFI. It controls the mode and the activation time of the saboteurs. As shown in Fig. 2, the fault injection controller consists of two interfaces. The first is the saboteur interface. This interface is a bus where, for each saboteur, an activating signal is included. It also contains mode signals to control the mode of the saboteurs. The second port is the General Purpose Input/Output (GPIO) port. This interface is used to control the fault injection controller. The internal fault pattern memory is filled via the GPIO port. This memory is used to specify when each attack pattern is activated.

Due to the GPIO interface of the MFI, it is possible to automate fault injection campaigns. The easiest way, when using a Virtex5 FXT FPGA, is to control the MFI via GPIO commands through the FPGA-internal PowerPC. Using this solution, a simple, generic software interface to the generic hardware interface is provided, which has the advantage that the test-engineer does not need specific knowledge about the fault injection system itself. Especially in the case of security evaluations, the evaluation engineer will be mostly concerned with the proper mapping of a real attack scenario to the saboteur matrix inside the system of interest. The GPIO interface of the MFI can be easily changed to every other interface, which allows the writing of the internal memory of the MFI. This flexibility makes the whole flow independent of the test environment.

3.1.1. Implementation

The fault injection controller is split into five main parts.

- **Saboteur interface:** The saboteur interface was implemented to allow for easy expandability to support several hundreds of saboteurs. The size of the saboteur interface bus is equal to the number of saboteurs plus the mode signals to control the mode of the saboteurs. Another method to control the saboteurs is the scan-chain approach. The disadvantage of the scan-chain approach is that multiple clock cycles are required for the configuration of the saboteurs.
- **GPIO interface:** The GPIO interface is the interface of the MFI to the controlled processor. All attack scenarios can be loaded into the MFI via this port.
- **Memory control:** The memory control is used to place the fault patterns into the internal fault pattern memory. It also generates signals for the control logic if a special command is sent via the GPIO port.
- **Internal fault pattern memory:** This memory stores the fault patterns, which define the attack scenario.
- **Control logic:** The control logic activates the saboteurs depending on the information stored in the internal fault pattern memory.
- **Trigger:** The MFI also has a trigger source, which can be used to synchronize the fault injection with the current state of the executed software.

3.2. Saboteurs

To model a wide selection of possible faults, we introduce a configurable saboteur. According to [20,21], saboteurs can be categorized into several different types. Depending on their location, they can be differentiated between serial and parallel saboteurs. Depending on the directionality, a division can be made between uni- and bidirectional ones. Finally, depending on their complexity, simple and complex saboteurs can be distinguished. The saboteur used in this work can be classified as a unidirectional, serial, simple saboteur. Thus, it only works in one direction, it is located directly in the connection signal, and it affects only one port at the input and output side. The supported fault models of such an injection element are shown in Table 1.

The proposed saboteur can be used to inject faults into single bit lines or even complete buses, which allows the emulation of bus attacks with, e.g., unfocused lasers or influence by strong external energy sources. The first four saboteur configuration modes are equivalent to direct circuit modifications. Bit-flips could also be forced by short, intensive, external pulses. Delay faults emulate circuit behavior in the case of operating voltage changes. A schematic block diagram and fault effect visualization of the proposed single-bit saboteur is shown in Fig. 3.

Table 1
Saboteur operation types and their description.

Saboteur mode	Fault type	Description
Stuck-at-zero	Permanent	Signal value of '0' until reload
Stuck-at-one	Permanent	Signal value of '1' until reload
Indetermination	Permanent	Undefined signal state until reload
Bridging fault	Permanent	No output propagation until reload
Negation of input	Permanent	Undefined signal state until reload
Bit-flip	Transient	Output inverts input for one cycle
Artificial delay	Transient	Input to output propagation delay

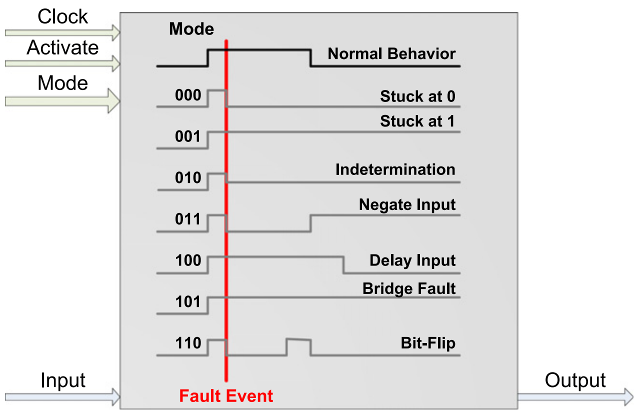


Fig. 3. Block diagram of the proposed saboteur element (obtained from [27]).

3.2.1. Implementation

The complexity of the implementation of the saboteurs depends on the features a saboteur supports. Therefore, the required number of slices for one saboteur mainly depends on their complexity. If the interface of the saboteurs is not changed, only the architecture of the saboteur must be modified to support more features. Thus, once the saboteurs are placed, the evaluation engineer can adapt the saboteurs by changing only one file, which makes the saboteur placement concept very flexible.

3.3. Fault pattern support

To evaluate a device-under-test for its fault attack sensitivity, it is important to know the exact location of security relevant silicon regions. Because the system is designed for fault effect modeling to test security features during software development, the required locations to place saboteurs are limited. Important locations can be memory interfaces, inputs and outputs of the system or inputs and outputs of calculation modules. At these locations, saboteurs must be placed by an evaluation engineer. A fault pattern approach is used for an effective mapping of saboteurs to their corresponding fault locations. Of course, not every possible pattern combination must be transmitted to the MFI. Every pattern used represents a very likely fault effect. For example, memory data at a specific address have been manipulated. Where exactly the attack was injected does not matter; only the effect that can be observed by the software is important for the operating system designer. For hardware evaluation, other methods must be used. This approach allows the user to check that the software security features developed work as intended, independent of how the fault effect is produced. The basic concept of mapping the fault pattern with the physical implementation is shown in Fig. 4.

The encoding of the fault pattern is performed by an array with *x* columns and *y* rows. Each element represents one saboteur or an area where no saboteur is placed. To define a special attack pattern, only an array where the active saboteurs are marked must be transmitted to the MFI.

3.3.1. Implementation

In this example, two methods of pattern generation are used to show the function of the fault MFI. The first one is a random number generator to show the functionally of the MFI. The second method is a pattern defined by a test engineer, which can be used for fault effect production. The complexity of such a pattern generation does not increase the required slices of the MFI because the patterns are not generated directly in the MFI.

3.4. Automatic placement of saboteurs

For a large number of saboteurs, an automatic saboteur placement approach is required because manual saboteur placement is a very time consuming and error prone process. In [20], a theoretical method is discussed for placing saboteurs automatically into a VHDL design. Our approach works very similar to that approach. In Fig. 5, a schematic example of automatic saboteur placement is shown. The saboteur placement tool is based on the vMAGIC VHDL parser library [23]. VMagic is a Java API that can parse and adapt VHDL code automatically. The flow can automatically add hundreds of saboteurs [22]. Then, the VHDL code is automatically adapted with saboteurs and the fault injection controller. A method to modify the VHDL code automatically is shown in [24].

3.4.1. Implementation

For this paper, the automatic saboteur placement tool was used to place the saboteurs into a critical signal in the design, which improves the very error prone and time consuming process of manual placement of the saboteurs.

3.5. Attack scenario

As shown in Fig. 6, the attack flow is based on a golden model run. The golden model runs from one reset to the next reset of the DUT. All of the golden model information is stored (e.g., the output, none volatile memory (NVM), and timing).

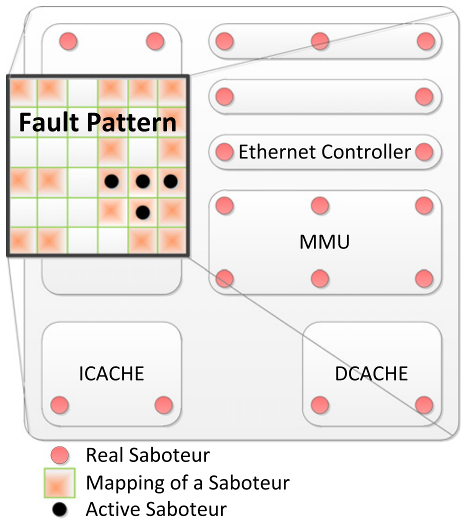


Fig. 4. Schematic view of an extracted fault pattern (obtained from [27]).

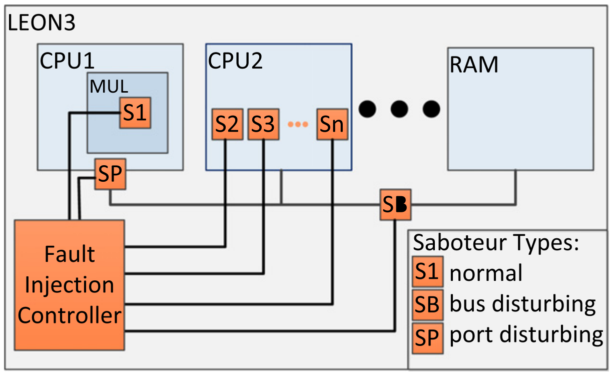


Fig. 5. Automatic routed saboteurs (adapted from [22]).

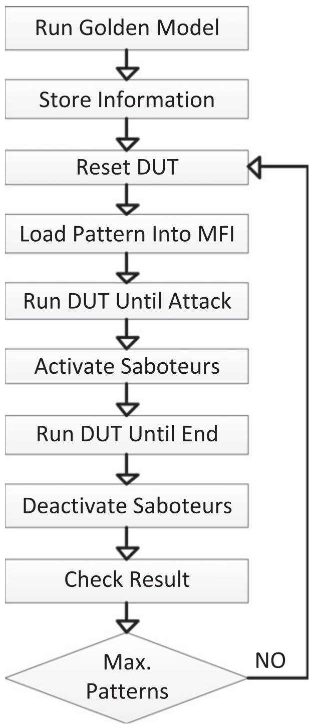


Fig. 6. Flow diagram shows a typical fault injection run (adapted from [27]).

The next step is to reset the whole system. Then, all of the attack patterns are stored in the internal fault pattern memory via the GPIO interface. In addition, the attack mode must be transmitted via GPIO. The mode is used to define the attack type of the saboteurs (e.g., bit-flip). After a pre-determined time, the specified fault attack will be injected, and then the saboteurs are activated depending on the fault pattern. Now, the faulty DUT will run until it is reset or a pre-defined timeout is reached. This timeout is required because the device could run into an infinite loop after the fault injection.

After the fault injection run is finished, the saboteurs are deactivated, and the result is compared to the results from the golden model run. If the results are not equal, a fault analysis process must be launched. This procedure is continued until all attack patterns are tested.

The fault pattern generator creates attack patterns for the fault injection. In this example, only a random number generator is used as the pattern generator to show the function of the fault MFI. The complexity of such pattern generation does not increase the required slices of the MFI because the patterns are not generated directly in the MFI.

4. Experimental results

To prove the effectiveness of our approach, we chose to implement the proposed controller on a widely used platform, the LEON3 SPARC V8 conformant processor from Gaisler Research [9].

The test setup is implemented on a Virtex5 FPGA using the Xilinx ML507 evaluation board. Simulation results are obtained using Modelsim software, which is part of the ISE software package provided by Xilinx. The fault injection setup is configured as shown in Table 2.

4.1. LEON3 platform setup

The LEON3 is configured for a single-core configuration using the default platform settings for this particular evaluation board (ML507). The MFI is configured via GPIO and can be accessed using the PowerPC.

4.2. Saboteur configuration

The random approach of fault injection proposed in this publication can find global dependencies between local fault occurrences. Of course, with an increasing number of saboteurs, the number of possible combinations increases until an analysis of the results is not possible in a reasonable time frame. Therefore, a small number of saboteur locations were chosen for these fault injection experiments.

4.3. Simulation results

To ensure the correct behavior of the fault injector and its saboteur, the whole LEON3 system are thoroughly simulated using the Modelsim software package. The simulation includes not only the fault injection process but also start and calculation instructions. The results of these injection campaigns are shown in Table 3.

The pattern injection rate is constant for patterns smaller than 32 because of the working principle of the MFI. For larger patterns, additional GPIO transfers are required. However, with increasing pattern size, the amount of concurrently injected faults increases accordingly.

4.4. Proof of fault models

To show that all possible fault models of the saboteur work as intended, a simple program is attacked. The program attacked is a simple program that sends five bytes via the UART to a host PC. On the UART interface, saboteurs are placed and activated during the send process. Table 4 shows the effects that can be reproduced with the saboteurs. As shown, all of the effects can be reproduced in software as well on the FPGA.

4.5. VHDL synthesis results

To estimate the necessary FPGA area requirements, the synthesis of a typical platform configuration has been performed. The results of several synthesis runs using different saboteur configurations are shown in Table 5.

The MFI and its saboteurs have a negligible effect on the size of the resulting synthesized design. Basing on the routing results, it should be possible to implement a large number of saboteurs on this FPGA platform. These results show that the saboteurs and the MFI produce an overhead to the system. The PowerPC is an op-

Table 2
Fault injection setup.

Saboteur type	Fault mode	Fault target type	Fault injection targets
Single-bit	Bit-flip	Control logic	Integer pipeline Cache controller Register file Multiplier unit Divider unit

Table 3
Simulated fault injection performance.

Saboteurs	Inj. patterns	Time (s)	Patterns (s)
8	256	274	0.93
16	256	282	0.91
24	256	286	0.90
32	256	299	0.86

Table 4
Saboteur fault model simulation and emulation results.

Saboteur mode	Simulation	Emulation
Stuck-at-zero	ok	ok
Stuck-at-one	ok	ok
Indetermination	ok	ok
Bridging fault	ok	ok
Negation of input	ok	ok
Bit-flip	ok	ok
Artificial delay	ok	ok

Table 5
VHDL synthesis results.

Saboteurs	Look-up-tables	Overhead (%)	Slices	Overhead (%)
0	14897	–	10423	–
8	14950	0.36	10513	0.86
32	15107	1.41	10642	2.10
96	15194	1.99	10716	2.81
170	15244	2.33	10774	3.37
326	15478	3.9	11088	6.34

Table 6
Emulated fault injection performance.

Saboteurs	Inj. patterns	Time (s)	Patterns (s)	Pattern blocks (s)
8	100 M	46.76	2.17 M	2.17 M
32	100 M	46.76	2.17 M	2.17 M
96	100 M	114.48	873.5 k	2.17 M
170	100 M	156,4	639.4 k	2.17 M
326	100 M	282.16	354.4 k	2.17 M

tional block to control the MFI. This controller is only used to control the GPIO of the MFI. The GPIO can be controlled directly by a host PC or any other tool.

4.6. Fault injection performance

In this subsection, the results of several fault injection campaigns are presented to show that higher structural flexibility does not come with disadvantages concerning emulation speed. It also must be mentioned that such a multi-bit injection campaign leads to complex results in system behavior, and therefore, the analysis step will most likely be the slowest link in the emulation chain. The results are summarized in Table 6. The number of saboteurs has an influence on the number of patterns that can be injected per second. The reason is that the GPIO interface only support the transmission of 32 bits of the pattern simultaneously. If the size of the pattern is increased, the time required for the pattern transmission also increases. The last column shows that the transmission of one pattern is interdependent of the number of saboteurs. It is not necessary to send the whole pattern for each attack. If only one pattern block must be changed, only one GPIO transmission is required. Only the maximum number of faults is important because not all of the saboteurs are activated. The more important numbers are the patterns injected per second. The speed of the pattern injections per second is not independent of the number of saboteurs. However, the time required for the transmission of one fault pattern is constant.

The results presented in Fig. 7 shows the maximum speed that the fault pattern can be changed per second. This number is important if an attack is emulated by a laser moving over the chip. Thus, it is possible to change the fault effect in the range of 300 k–2 M times per second. If one fault per run should be injected, the whole test requires the run time of the program that should be tested in addition to the configuration time of the MFI.

4.7. Case study: Attack on RSA authentication

The MFI is used to show a case study of a fault attack on RSA authentication [25]. The RSA authentication process signs a message with a private key. To generate the signature, the RSA algorithm uses multiplications. In [25], it is concluded that fault attacks on the multiplication unit can be used to attack the RSA algorithm. For this fault attack, the authors assume that only one bit of the multiplier output switches. In this case study, we show that our approach can emulate such fault attacks exactly.

For this test, we chose the LEON3 processor in a single core configuration. For the attack target, the output signals and one of the operand registers of the multiplier have been chosen. Because only the fault effect of this one-bit result attack should be reproduced, no hardware error detections methods are implemented in the LEON3 processor.

An overview of the test environment is given in Fig. 8. Listing 2 shows the pseudo code describing the fault injection flow. The saboteur placement methodology is described in [22]. The FPGA-internal PowerPC runs a Linux kernel to allow programming of the fault injection flow in a comfortable way. The fault injection control software runs on the PowerPC. The host PC evaluates the results of the fault injection. During this attack, the saboteurs are configured in bit-flip mode.

As shown in Listing 1, the code that is executed on the LEON3 is a simple program that only calculates one multiplication and send the result back to the host PC. This program should only be used as a proof of concept.

The results are shown in Table 7 and Fig. 9. The MFI can emulate a fault attack on the multiplier interface. Because the MFI supports a large number of saboteurs, it is possible to place the saboteurs at not only the output: it is also possible to test also the effect of a fault injection to the input of the multiplier. If the operant 1

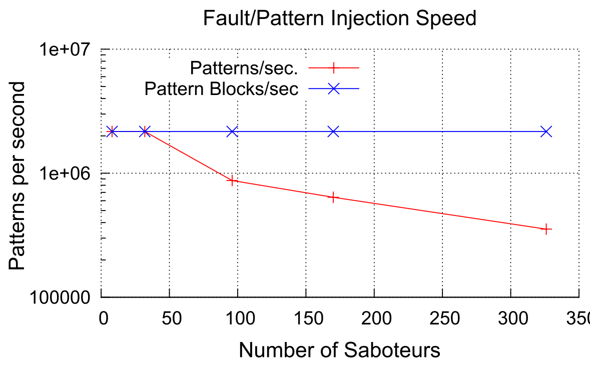


Fig. 7. Speed of the fault injection depending on the number of saboteurs.

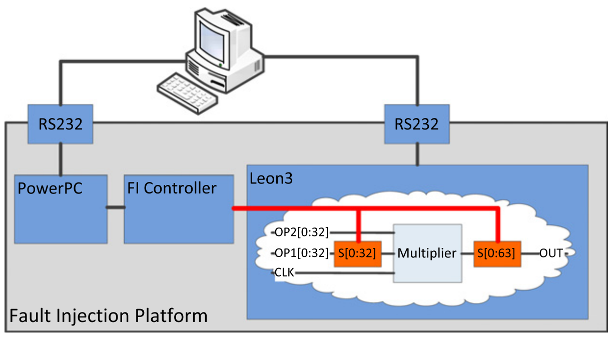


Fig. 8. Overview of the test environment (adapted from [27]).

```
main(){
  init_system();
  output = a*b;  //attacked operation
  print(output);
}
```

Listing 1. Pseudo code of the LEON3 software.

```
run_golden_model();
store_all_information();
for(i=0;i<n_patterns;i++){
        reset_dut();
        load_pattern(pattern[i]);
        run_DUT(pattern_time[i]);
        activate_saboteurs();
        run_DUT(max_time);
        deactivate_saboteurs();
        check_result();
}
```

Listing 2. Pseudo code of the fault injection flow.

Table 7
Number of faults detected depending on the attack pattern.

Attack target Number of saboteurs	OP1 1	OP1 2	OP1 3	OP1 4	Out 1
No. of bit-flips					
0	37	37	37	37	0
1	208	0	0	0	5000
2	275	365	129	59	0
3	477	428	532	224	0
4	676	624	478	646	0
5	899	836	862	704	0
6	942	862	762	852	0
7	714	686	787	817	0
8	369	529	580	604	0
9	226	337	432	487	0
10	105	170	221	319	0
11	40	82	99	167	0
12	20	22	56	58	0
13	9	13	13	24	0

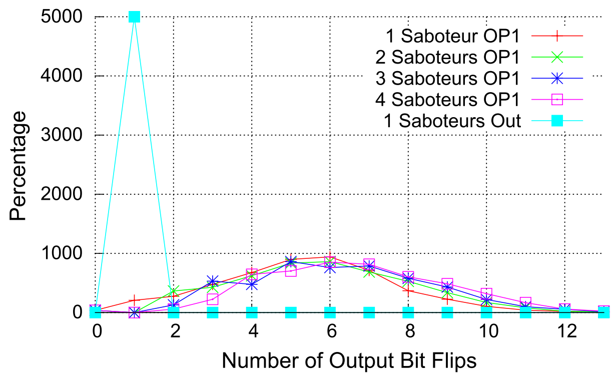


Fig. 9. Influence of saboteurs on the multiplication unit. Bit-flips of the output depending on the number of active saboteurs at the input/output.

(OP1) is attacked, the number of output bits switches depends on which bit of the input is attacked. A similar effect can be reproduced by attacking multiple bits at OP1. If exactly one bit at the output should be disturbed, the simplest method is to attack the output (OUT) of the multiplier with exactly one saboteur.

This emulated fault reproduces a fault attack on RSA authentication where exactly one output bit of the multiplication must be disturbed [25]. To test whether the operating system can detect such an attack, it makes sense to place saboteurs at the output of the multiplier and manipulate exactly one bit of the result.

4.8. Case study: Long-time test for light attack emulation on the memory of a smart card

The MFI can also be used for tests with long durations. This test shows how the MFI can be used in systems development to improve security. As an example, the estimated effect of a light attack on a smart card memory is emulated [26]. In particular, the effect of permanent faults injected into the memory should be evaluated. Therefore, security relevant memory regions must be attacked every clock cycle.

Fig. 10 shows that only some memory regions at special times are security relevant. The relevance of the different memory regions must be defined by the evaluation engineer. Some criteria for the engineer by which code regions should be tested can include regions that.

- Contain final or intermediate results of an security relevant function.
- Contain keys of an encryption process.
- Contain, for example, an AES S-Box.

This reduction is required because attacking every memory cell is too time-consuming. In Eq. (1), the number of tests required is calculated. If the critical memory address section is 100 bytes, the critical regions consist of 1 k clock cycles, and the number of tested patterns per memory address is 10, we obtain an overall number of 1 M tests. Assuming that the time required for one run is 500 ms, then the whole test would require approximately 6 days.

$$N_{tests} = N_{memoryaddresses} * N_{clkcycles} * N_{attackpatterns} \quad (1)$$

Fig. 11 shows the structure of the fault emulator for this fault injection campaign. For this test, the MFI is placed directly into the Smart Card system. The MFI is controlled via a serial interface, which can also be used to control the smart card and allows us to control the MFI without adding an additional port to the smart card emulation system.

For this test, we used software that requires 150 k clock cycles for a full execution. The security-relevant region of this code is approximately 1000 clock cycles long. During these cycles, an authentication process is performed. To test whether the operating system can cope with light attacks on the memory, we choose 100 attack points within security relevant regions. We also reduced the tested memory section to 1536 bytes because only these bytes are

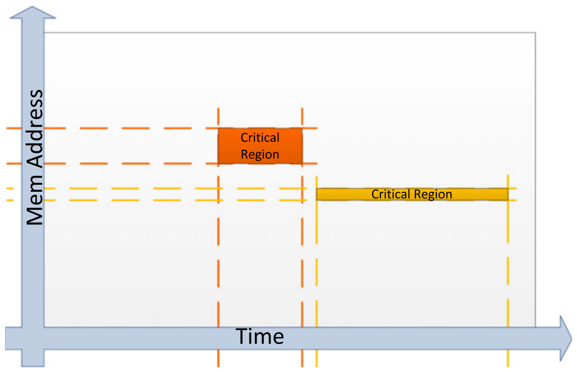


Fig. 10. Critical memory regions of a smart card system.

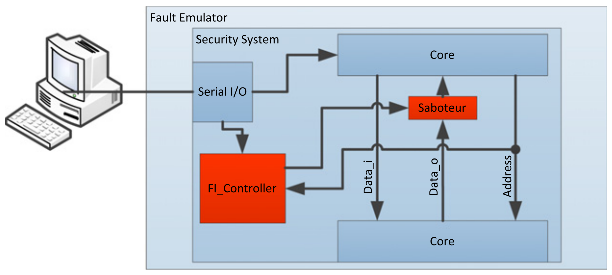


Fig. 11. System modifications for the RAM fault emulations.

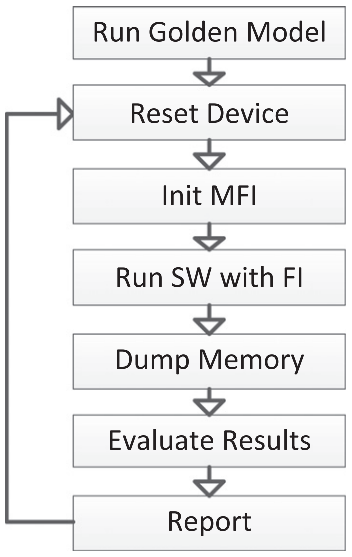


Fig. 12. RAM fault emulation flow.

Table 8
Results of the light attack emulation on the memory.

	N	%
Total number of injections	156672	100
Memory cell never accessed	121570	77.6
Fault injected	35102	22.4
No effect	30373	19.4
Fault detected by smart card	4729	3.0

security-relevant. To emulate a permanent fault, we set the mode of the saboteurs to stuck-at-one. This leads to a total of 157 k tests (see Eq. (2)).

$$N_{tests} = 1536 * 102 * 1 = 156672 \quad (2)$$

The flow of this test is shown in Fig. 12. The test flow can be split into seven main parts:

Table 9
Fault injection performance compared to previous results.

Approach	Clock cycle (ns)	Inj. faults	Time (s)	Inj. speed 1/(s)	No. inj. speed 1/(s)
[6]	50	100 k	83	1205	603
[12] AE	16.8	129.75 M	698	185888	276619
[12] PR	18.97	10 k	1014	9.86	13
[14] DPR	10	–	–	12987	32468
[14] PRR	10	–	–	414.94	1037
This work	25	100 M	46.76	2.17 M	2.17 M

- **Run a Golden Model:** The golden model run is required as reference to evaluate whether an attack was successful or not. During the golden model runs, all output values and the content of the memory are stored.
- **Reset Device:** The reset is required to start the execution at the same point for each run. This reset is the reason that the execution time of a fault injection campaign is approximately as long as the execution of the software tested.
- **Initialize MFI:** The initialization of the MFI requires only a few microseconds (5–30 s). The values that must be transmitted to the MFI include the following:
  - What attack should be performed (in this case, stuck-at-one).
  - When the attacked should occur.
  - What memory address should be attacked.
- **Run the Software with Fault Injection:** After the MFI is initialized, the execution of the software is started. When the predefined time is reached, the MFI starts the fault injection. All of the software outputs are stored. If the execution of the software requires more than twice the execution time of the golden model, the system is stopped.
- **Dump the Memory:** To determine whether the attack was successful, the memory of the whole system is dumped out of the device to the PC.
- **Evaluate Results:** The evaluation step checks what effect the attack had on the system. The effects can include the following:
  - Fault injected into memory.
  - Output changed.
  - Output and memory content changed.
- **Report:** This step writes the information collected during this test to a file.

Finally, the results of the RAM fault emulation are shown. Table 8 shows that we used a total of 157 k fault injection runs. The execution of the whole test requires 22 h. In 122 k attack scenarios, the fault injection had no effect because the memory cell we attacked was never used after the attack. 35 thousand attacks changed the content of the RAM, but they had no effect on the non-volatile memory or the output of the system. In the rest of the attack cases, the fault detection algorithms of the smart card operating system detects the fault injection. There was no attack case where the system was successfully attacked. The system never sent incorrect outputs or changed non-volatile memory content.

4.9. Comparison to existent fault injector solutions

In recent years, many high-performance fault injection emulation platforms have been published. While fault injection speed is only one parameter to quantify the performance of a fault injector solution, it is important to measure how many faults can be injected in a reasonable time. It is especially important for complex system-on-chip designs with a high number of interesting injection points. In [6], fault injection campaigns using different benchmark configurations have been compared to typical simulation solutions. To cover all design approaches considered, reconfiguration techniques are also presented in [12]. This publication shows two implementations, one using autonomous emulation (AE) and one using partial reconfiguration (PR). The test object in this case is the freely available CORDIC16 core.

Our fault injector solution profits from a higher clock frequency and state-of-the-art FPGA hardware. Another difference that must be considered is that, because of the pattern approach, the fault injection rate is not constant. Therefore, the total number of activated faults is calculated based on the number of activated faults per turn. For the evaluation, the worst case is assumed, where only one fault is injected per pattern. If the simultaneously injected faults are increased, the number of faults per seconds will be increased (one turn consists of all possible bit combinations of the fault pattern width selected). The results of these investigations are compared to our result in Table 9. Because of the different test systems the values have to be normalized. The last column shows the fault injection speed normalized to a clock cycle of 25 ns.

The results of this work presented in Table 9 shows the maximum speed that the fault pattern can be changed per second. This number is important if an attack is emulated by a laser moving over the chip. Thus, it is possible to change the fault effect in the range of 300 k–2 M times per second. If one fault per run should be injected, the whole test requires the run time of the program that should be tested in addition to the configuration time of the MFI.

The comparison shows that the in-system solution does not suffer from additional performance penalties. Strong spatial containment of the possible fault locations leads to very high injection results if combined with large fault injection patterns. It can also be concluded that autonomous emulation provides injection speed advantages compared to partial reconfiguration techniques. The reconfiguration approach suffers not only from limiting communication interfaces but also from long delays caused by the reconfiguration process. This reconfiguration also does not include the time needed to generate reconfigurable sub-blocks. These problems are targeted by the work presented in [14]. In this publication, partial reconfiguration is compared to direct reconfiguration using an FPGA internal port to its configuration memory. While the reconfiguration speed is greatly improved, it is still slower than autonomous emulation solutions such as the one proposed in this paper.

5. Conclusion

This paper presents a case study on multiple fault dependability and security evaluation. A highly modularized controller solution for portable fault injection systems was used. It has been shown that fault injection campaigns can be executed very efficiently through a generalized interface using a high level abstraction of physical fault sources. The design is scalable to allow both fully automated campaigns with a larger fault pattern memory and more user controlled campaigns using a small silicon footprint. The performance is comparable to existing platforms using circuit manipulation and significantly faster than ones using partial and complete FPGA reconfiguration. This study is the first step towards a complete fault injection platform for dependability and security evaluations. Its flexible design will allow the evaluation engineer to shift focus from complex testing architectures to more complex fault models, which should finally allow the modeling of not only simple SEUs but also complex fault attack scenarios. Through the additional abstraction level, a better separation of the test and design engineering tasks is provided. The knowledge gained from these experiments will be used to obtain a better understanding of inner-chip fault mechanics. Such full scale investigations require fully automated saboteur injection techniques, which are currently under development. Consideration of such attacks will be necessary to design efficient and secure smart card systems and is also valid for highly integrated systems or systems under high environmental stress.

Acknowledgments

The authors thank the Austrian Federal Ministry for Transport, Innovation, and Technology, which funded the Power-Modes Project under the FIT-IT contract FFG 825749. We would also like to thank our project partners Infineon Technologies Austria AG and Austria Card GmbH.

References

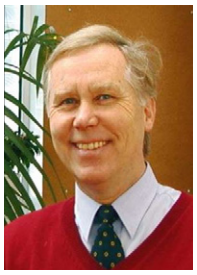
[1] M. Nicolaidis, Time redundancy based soft-error tolerance to rescue nanometer technologies, in: Proc. IEEE VLSI Test Symposium (1999), 1999, pp. 86–94. doi: 10.1109/VTEST.1999.766651.
[2] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, C. Whelan, The sorcerer's apprentice guide to fault attacks, Proceedings of the IEEE 94 (2) (2006) 370–382, http://dx.doi.org/10.1109/JPROC.2005.862424.
[3] D.D. Andres, J.C. Ruiz, D. Gil, P. Gil, Fades: a fault emulation tool for fast dependability assessment, in: Proc. IEEE Int. Conf. Field Programmable Technology (FPT 2006), 2006, pp. 221–228. doi: 10.1109/FPT.2006. 270315.
[4] Xilinx, fpga family. <http://www.xilinx.com/products/virtex5/index.htm> (07.10).
[5] J. Boue, P. Petillon, Y. Crouzet, Mefisto-l: a vhdl-based fault injection tool for the experimental assessment of fault tolerance, in: 28th Annual International Symposium on Fault-Tolerant Computing, 1998. Digest of Papers, 1998, pp. 168–173. doi: 10.1109/FTCS.1998.689467.
[6] P. Civera, L. Macchiarulo, M. Rebaudengo, M.S. Reorda, M. Violante, An fpga-based approach for speeding-up fault injection campaigns on safety-critical circuits, Journal of Electronic Testing 18 (3) (2002) 261–271, http://dx.doi.org/10.1023/A:1015079004512.
[7] A. Benso, B. Prinetto, Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation, Kluwer Academic Publishers, 2003.
[8] P. Ellervee, J. Raik, K. Tammemae, R.-J. Ubar, Fpga-based fault emulation of synchronous sequential circuits, IET Computers & Digital Techniques 1 (2) (2007) 70–76, http://dx.doi.org/10.1049/iet-cdt:20050065.
[9] Gaisler, Leon3 processor. <http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53> (07.10).
[10] A. Pellegrini, K. Constantinides, D. Zhang, S. Sudhakar, V. Bertacco, T. Austin, Crashtest: a fast high-fidelity fpga-based resiliency analysis framework, in: Proc. IEEE Int. Conf. Computer Design (ICCD 2008), 2008, pp. 363–370. doi: 10.1109/ICCD.2008.4751886.
[11] L. Antoni, R. Leveugle, M. Feher, Using run-time reconfiguration for fault injection in hardware prototypes, in: Proc IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT 2002), 2002, pp. 245–253. doi: 10.1109/DFTVS.2002.1173521.
[12] C. Lopez-Ongil, L. Entrena, M. Garcia-Valderas, M. Portela, M.A. Aguirre, J. Tombs, V. Baena, F. Munoz, A unified environment for fault injection at any design level based on emulation, IEEE Transactions on Nuclear Science 54 (4) (2007) 946–950, http://dx.doi.org/10.1109/TNS.2007.904078.
[13] M. Jeitler, M. Delvai, S. Reichor, Fuse – a hardware accelerated hdl fault injection tool, in: Proc. SPL Programmable Logic 5th Southern Conf., 2009, pp. 89–94. doi: 10.1109/SPL.2009.4914906.
[14] L. Kafka, Analysis of applicability of partial runtime reconfiguration in fault emulator in xilinx fpgas, in: DDECS '08: Proceedings of the 2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, IEEE Computer Society, Washington, DC, USA, 2008, pp. 1–4. doi: 10.1109/DDECS.2008.4538781.
[15] B.F. Dutton, M. Ali, C.E. Stroud, J. Sunwoo, Embedded processor based fault injection and seu emulation for fpgas, in: Proc International Conf. on Embedded Systems and Applications 2009, 2009, pp. 183–189.
[16] L. Kafka, M. Danek, O. Novak, A novel emulation technique that preserves circuit structure and timing, in: International Symposium on System-on-Chip 2007, 2007, pp. 1–4. doi: 10.1109/ISSOC.2007.4427437.
[17] R. Leveugle, Early analysis of fault-based attack effects in secure circuits, IEEE Transactions on Computers 56 (10) (2007) 1431–1434, http://dx.doi.org/10.1109/TC.2007.1078.
[18] R. Leveugle, K. Hadjiat, Multi-level fault injections in vhdl descriptions: alternative approaches and experiments, Journal of Electronic Testing 19 (5) (2003) 559–575, http://dx.doi.org/10.1023/A:1025178014797.
[19] J.-M. Daveau, A. Blampey, G. Gasiot, J. Bulone, P. Roche, An industrial fault injection platform for soft-error dependability analysis and hardening of complex system-on-a-chip, in: Reliability Physics Symposium, 2009 IEEE International, 2009, pp. 212–220. doi: 10.1109/IRPS.2009.5173 253.
[20] J. Baraza, J. Gracia, D. Gil, P. Gil, Improvement of fault injection techniques based on vhdl code modification, in: Tenth IEEE International High-Level Design Validation and Test Workshop 2005, 2005, pp. 19–26. doi: 10.1109/HLDVT.2005.1568808.
[21] J.-C. Baraza, J. Gracia, S. Blanc, D. Gil, P.-J. Gil, Enhancement of fault injection techniques based on the modification of vhdl code, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 16 (6) (2008) 693–706, http://dx.doi.org/10.1109/TVLSI.2008.2000254.
[22] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, J. Haid, Automatic saboteur placement for emulation-based multi-bit fault injection, in: Proc. 6th Int Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC) Workshop, 2011, pp. 1–8. doi: 10.1109/ReCoSoC.2011.5981521.
[23] C. Pohl, R. Fuest, M. Porrmann, vMAGIC – automatic code generation for VHDL, Newsletter Edacentrum 2 (2010) 7–10.
[24] C. Bachmann, A. Genser, C. Steger, R. Weiss, J. Haid, Automated power characterization for run-time power emulation of soc designs, in: Proc. 13th Euromicro Conf. Digital System Design: Architectures, Methods and Tools (DSD), 2010, pp. 587–594. doi: 10.1109/DSD.2010.38.

[25] A. Pellegrini, V. Bertacco, T. Austin, Fault-based attack of RSA authentication, in: Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE), 2010, pp. 855–860.
[26] M. Neve, E. Peeters, D. Samyde, J.-J. Quisquater, Memories: a survey of their secure uses in smart cards, in: Proc. Second IEEE Int. Security in Storage Workshop SISW '03, 2003. http://dx.doi.org/10.1109/SISW.2003.10004.
[27] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, J. Haid, Modular fault injector for multiple fault dependability and security evaluations, Digital System Design (DSD), 2011 14th Euromicro Conference, Aug. 31 2011–Sept. 2 2011, pp. 550–557, http://dx.doi.org/10.1109/DSD.2011.76.

Johannes Grinschgl received his Master's degree in Telematics from Graz University of Technology in 2008, focusing on microelectronics and telecommunications. Since 2010 he is doing his Ph.D. in Electrical Engineering at the Institute for Technical informatics at Graz University of Technology in collaboration with Infineon Technologies Austria AG and Austria Card GmbH. His research interests incorporate fault emulation as well as fault modeling.

Armin Krieg received his Master's degree in Telematics from Graz University of Technology in 2008, focusing on microelectronics and system-on-chip design. Since 2010 he is doing his Ph.D. in Electrical Engineering at the Institute for Technical Informatics at Graz University of Technology in collaboration with Infineon Technologies Austria AG and Austria Card GmbH. His research interests incorporate fault emulation as well as fault detection and recovery.

Christian Steger received the Dipl.-Ing. degree (equivalent to the American Master of Science) 1990, and the Dr. techn. degree in electrical engineering from Graz University of Technology, Austria, in 1995, respectively. He graduated from Export, International Management and Marketing course in June 1993 at Karl-Franzens-University of Graz. In January 2010 he completed the Entrepreneurship Development Program at MIT SLOAN SCHOOL OF MANAGEMENT in Boston, USA. He is key researcher at the Virtual Vehicle Competence Center (ViF, COMET K2) in Graz, Austria. From 1989 to 1991 he was software trainer and consultant at SPC Computer Training Ges.m.b.H., Vienna. From 1990 to 1991 he was research engineer at the Institute for Technical Informatics, Graz University of Technology, Austria. Since 1992 he is Assistent Professor at the Institut for Technical Informatics, Graz University of Technology. He has joined the Reactive Systems Group at Saarland University as interim professor (Lehrstuhlvertreter) in the winter semester 2010/2011. He heads the HW/SW codesign group (8 Ph.D. students) at the Institute for Technical Informatics. His research interests include embedded systems, HW/SW codesign, HW/SW coverfication, SOC, power awareness, smart cards, UHF RFID systems, multi-DSPs. He is member of the IEEE and member of the OVE (Austrian Electrotechnical Association).



Reinhold Weiss is a Professor of Electrical Engineering (Technical Informatics) and head of the Institute for Technical Informatics at Graz University of Technology, Austria. He received the Dipl.-Ing. degree, the Dr.-Ing. degree (both in Electrical Engineering) and the Dr.-Ing.habil. degree (in Realtime Systems) from the Technical University of Munich in 1968, 1972 and 1979, respectively. In 1981, he was as a Visiting Scientist with IBM Research Laboratories in San Jose, California. From 1982 to 1986 he was Professor of Computer Engineering at the University of Paderborn (Germany). He is author and co-author of about 170 scientific and technical publications in Computer Engineering. His research interests focus on Embedded Distributed Real Time Architectures (parallel systems, distributed fault tolerant systems, wearable and pervasive computing). He is a member of the International Editorial Board of the Computers and Applications (ISCAs) journal. Further, he is a member of IEEE, ACM, GI (Gesellschaft fuer Informatik, Germany), and OVE (Austrian Electrotechnical Association).

Holger Bock received his Diplom Ingenieur (corresponding to Master) degree in electrical engineering at the Graz University of Technology in 1994. From 1991 to 1998 he has been working on concepts, software and hardware development, especially on VLSI-Design for cryptographic coprocessors for smart cards (DES, ECC) at the Institute for Applied Information Processing and Communications Technologies (IAIK). In December 1998 he joined the team at Infineon's development center in Graz as a core competence for security. Since beginning of 2001 he had been a member of the technology and innovations methodology team at Infineon's business group Chipcard and Security ICs, focussing on secure, especially DPA resistant, design methodologies for cryptographic hardware. In October 2006 he has become responsible for worldwide funding management for Infineon's business group Chipcard.

Josef Haid received a Master's degree in Telematics and a doctoral degree in electrical engineering both from the Technical University of Graz in Austria in the years 2001 and 2003 respectively. Presently he is a senior staff engineer at Infineon Technologies in Graz/Austria and is responsible for specification of low-power contactless smart cards. His interests include advanced digital design and low power design of hardware and software.

Thomas Aichinger received his Diplom Ingenieur (corresponding to Master) degree in computer science at the Vienna University of Technology in 2003. From 2002 to 2007 he has been working on assessments of smartcards and software used for qualified electronic signatures, as well as an assessor for electronic payment systems and for data protection at the A-SIT (Secure Information Technology Center – Austria). In June 2007 he joined the R&D department at Austria Card Plastikkarten und Ausweissysteme GmbH responsible for security certification (Common Criteria, CAST, VISA Risk), definition of security requirements and concepts for their implementation in the ACOS Smart Card Operating System. Since 2011 he is also Product Manager for ID Products.

Christiane Ulbricht received her Diploma (M.Sc.) in October 2002. After that she worked as a research assistant at the Institute of Computer Graphics and Algorithms at the Vienna University of Technology. Her Ph.D. thesis focused on photo realistic computer graphics and especially on verifying the correctness of physically based computer graphic systems. In December 2006, she joined Austria Card as project manager at the research and development department. She was promoted to head of project management office in June 2008. Since June 2011, she has become head of the Research and Development department at Austria Card.

Efficient Fault Emulation based on Post-Injection Fault Effect Analysis (PIFEA)

Johannes Grinschgl, Armin Krieg,
Christian Steger, Reinhold Weiss
Institute for Technical Informatics
Graz University of Technology, Graz, Austria
{johannes.grinschgl, armin.krieg,
steger, rweiss}@tugraz.at

Holger Bock, Josef Haid
Design Center Graz
Infineon Technologies Austria AG, Graz, Austria
{holger.bock, josef.haid}@infineon.com

***Abstract*—Over the last years the complexity of SoC's has increased enormously. This increase leads to a high test effort against faults. Therefore, methods have been developed to speed-up fault testing to cope with the increasing number of possible faults. One method is to emulate fault attacks. To cope with the large amount of test data a method to check automatically if a fault injection was successful is required.**

In this paper a novel method is presented how automatically, on a fault emulation platform, can be proven if a fault forces the system to unintended behavior. The PIFEA hardware block is designed to check if the system execution flow is manipulated or if the system detects the fault and switches in a secure mode.

***Index Terms*—fault emulation, fault injection controller, saboteurs, multi-bit faults, automatic test pattern injection**

I. Introduction

Over the last years the complexity of systems-on-chips (SoC) has been increased. Some publications already speak about more than moore [1], [2]. This increasing system-on-chip size also leads to an increasing test effort. Especially the effort for testing the system against natural faults and fault attacks has increased over the last years. Therefore, several methods exist on how to prove if a system on chip can fulfill safety and security levels (e.g., for smart card applications the common criteria test is one of the most widely used methods to classify the chips' security level [3]). Such certification processes are very time consuming and costly. Critical faults, detected during the evaluation process, may imply a costly redesign. To increase the probability that the SoC would pass the certification process at first time, fault testing methods can be used. The testing methods, which can be used during the design phase, can be split into fault simulation and fault emulation. Simulations are on of the oldest method and can be used early in the design phase and are flexible in testing faults because every signal can be manipulated and it is easy to check the fault's effect on the system. The main disadvantage is that simulations are slow compared to other methods. To solve this speed problem fault emulation was developed. This method maps the functionality of the SoC to a configurable hardware platform (e.g., FPGA). The advantage is that the emulation runs extremely fast. The disadvantage is that the control-ability is reduced because not every signal can be manipulated at every point of time and for debugging not every internal signal can be observed. These, disadvantages are often acceptable for the speed-up win. For example in simulation you can not test millions of attacks against the whole memory of a system in a reasonable amount of time, but an emulator system can handle such an amount of attack scenarios. Several methods

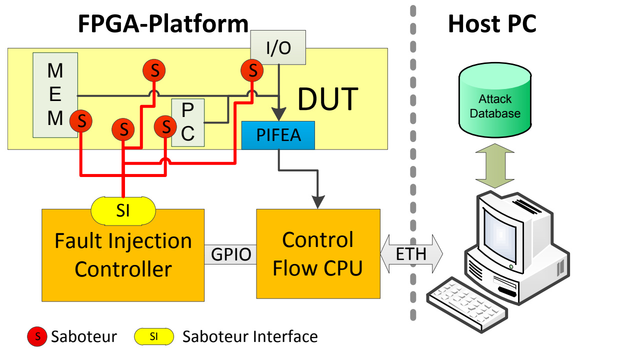


Fig. 1. Schematic view of the proposed fault injection system with post-injection fault effect analysis (adapted from [9])

exist to generate fault injection campaigns in such a way that the number of required test campaigns is reduced. There are methods, which use statistical attacks [4]–[6]. Another method monitors the program flow during a golden model run. This information is then used to detect security relevant regions [7], [8].

For such a high number of fault injection campaigns some methods are required to detect if a fault drives the system in an unintended behavior. To cope with this problem a novel post-injection fault effect analysis (PIFEA) method is presented in this paper. The main contributions of this paper are:

- A novel post-injection fault effect analysis method for detecting security relevant attacks directly on an emulation platform.
- Increasing analysis speed for checking if a fault brings the system in an unexpected state.
- Reducing communication overhead between emulation platform and host PC.

This paper is structured as follows. Section II briefly shows the state of the art on fault effect monitoring. In Section III the design of the fault injection controller is described. Section IV describes the function of the post-injection fault effect analysis concept. Section V shows some experimental results of this concept based on a Leon3 implementation [10]. Finally, conclusions are drawn in Section VI.

II. Related Work

In the field of post-injection fault analysis several works were published in the last years. These works can be split into two main groups. The first group is to observe the output of the system and to check if the output of the system is correct or corrupted. The second method is to observe also internal signals. To check if a fault occurs often golden model runs are used to generate data of a correct program execution. For the observation of the output several publications exists in literature [11], [12]. Output is not only the communication of the device with the environment also power consumption or EMV are viable information for an attacker. This method is often used in emulation and physical fault injection approaches because it is a error-prone and time consuming process to observe internal signals.

The other method is the white box testing method where internal signals are used to test if a system fulfills a certain security level. This method is described very often in the literature but mainly for simulation approaches [13], [14]. The main advantage of this method is that it is a flexible method because also hidden faults can be detected and the analysis what happened to the device after a fault is injected is much easier than with a normal black box testing method. But to evaluate such white box tests huge knowledge of the system is required to find the proper signals which indicate the type of the injected fault.

To the best of our knowledge there exists no post-injection fault effect analysis based on fault emulation which using internal signals for automatic checking if the system is in a secure state or not in literature.

III. Modular Fault Injector

The modular fault injection approach is a flexible fault injection method, which is used to test if the presented approach works correctly [9]. The advantages of the modular fault injection concept is that it can be easily modified for every possible system-under-verification. The modification of the RTL description of the system is done automatically [15], which reduces the time and error-prone process of manually modifying the system. This automatic source adaption is also used to place the modules of the new approach presented in this paper into the device-under-test (DUT). For providing the required flexibility the system consists of following blocks (seen in Figure 1).

1) **Attack Database:** This database is used to store attack and the attack scenarios.
2) **Host PC:** The host PC is the interface between the modular fault injection module and the test engineer. Tools allow the engineer to configure the injection campaigns and the evaluation of the test results.
3) **Modular Fault Injection Controller:** The modular fault injector is used to communicate with the host PC and to manage the whole fault injection campaign. Therefore, the controller offers several features. First, there are trigger modules, which can be used to synchronize the injection flow with the SoC. Two examples of such trigger signals are the program counter or the memory address. Second, there is an interface to the saboteurs, which can be used to inject the faults into the system.
4) **Saboteurs:** Saboteurs are blocks which inject a fault between source and sink of a signal line when they are activated. The injected fault can be configured by the modular fault injection controller. Possible faults are stuck-at-one, stuck-at-zero, delay and invert.

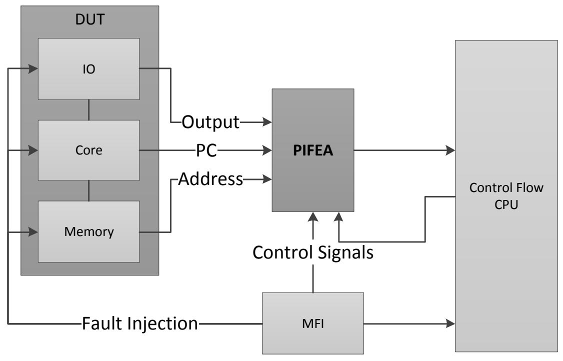


Fig. 2. Connection of the PIFEA module to the DUT and the environment.

5) **Control Flow CPU:** The control flow CPU can be connected between the modular fault injector (MFI) and the host PC. This module is optional but often on emulation platforms a fix controller is mounted, which can be used to reduce the communication overhead to the host PC by doing some pre-processing.

IV. Post-Injection Fault Effect Analysis System

The post-injection fault effect analysis (PIFEA) is a block which checks the status of the system-under-verification after a fault injection. As seen in Figure 2, the PIFEA is directly connected to the system-under-verification. Especially the inputs and outputs, the memory accesses and the program counter values are used to check if the system is in a secure state after a fault injection. The connection to the MFI is required because the PIFEA must know when a fault is injected. The interface to the control flow CPU is used to configure the PIFEA before the injection and to read out the result of the post-injection fault effect analysis. This output indicates if the system is in a secure or insecure state. Also the difference between the logged pc, memory accesses and input/outputs of the golden model run as well as the faulty run can be read out and analyzed.

A. PIFEA Module

The structure of the PIFEA module consists of five main components (see Figure 3).

- **Memory:** The memory stores the input values every clock cycle. This information is stored to allow the analysis of what went wrong if an unindented state is reached. Because the size of the memory is limited, the recording of input data is started when the fault is injected. If the size of the memory is not big enough it is possible to halt the emulation and read out the memory with the control flow CPU.
- **Golden Model Memory:** To compare the stored flow of the system it is important to have a reference flow. This flow is generated during a golden model run. During this run all logged input data is stored to a memory.
- **Comparator:** The comparator checks if the golden model run and the injection run flow match. If there are no differences in the execution there was no detect able system manipulation. If a difference is detected the behavioral of the design flow changes has to be evaluated.
- **Secure State Definition Memory:** This memory is written before the test is started and defines which system

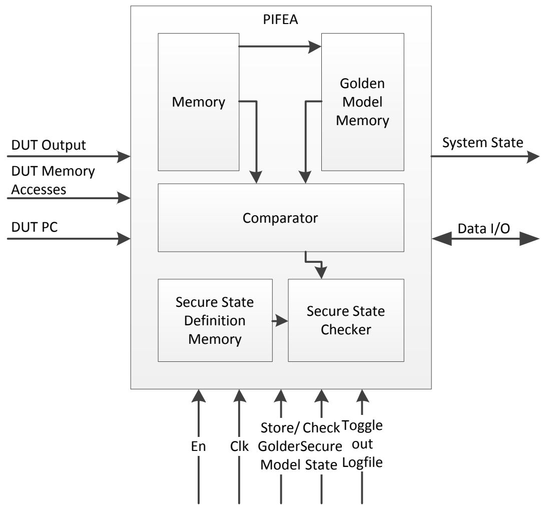


Fig. 3. Structure of the PIFEA Module.

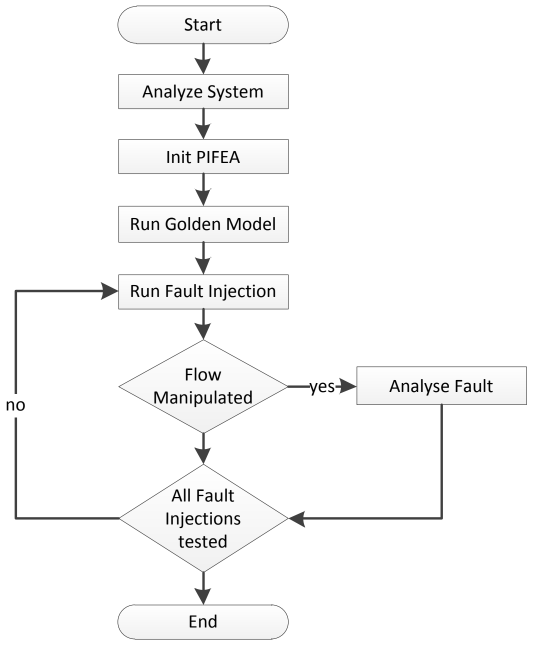


Fig. 4. Flow of the post-injection fault effect analysis.

states are safed for the system. E.g., if the system goes into a reset state, the fault has been detected and the whole system is restarted. An other safe state is that the system executes some fault recovery mechanisms. This state can be detected if the system reaches a specific program counter. Also information of not allowed program flows can be defined here. For example it is not allowed to read from or write to a specific memory address if a fault occurs because there are security relevant variables stored.

- **Secure State Checker:** This checker uses the information stored in the secure state definition memory to analyze if all defined security patterns are fulfilled. All generated information is combined and can be read out by the controlling CPU.

B. *Fault Analysis Flow*

The main concept of the fault analysis flow is to allow for easy and fast analyzing if the system is in a secure or insecure state after a fault injection. Therefore, the flow has to be split into several parts (see Figure 4). An analysis step where the test engineer defines security critical regions (e.g., security critical memory addresses, security critical code region). Also safe states have to be defined by the test engineer (e.g., reset of the system, program counter of fault recovery functions). After this step the system software initialize the PIFEA and runs a golden model run. During this run for every clock cycle a program counter value, the memory accesses, and the output is stored. This step is required because if the system do not reach a secure or insecure state the PIFEA must check if the system is influenced by the fault injection or not. After the golden model run the system-under-verification has to be reset and run until the fault injection is activated. Starting at this point the memory stores the program counter, memory access and the output of the system. Meanwhile the comparator block starts to check if the execution differs from the original execution. If a mismatch is detected the comparator stops to read out the memory and the input information is stored until the memory is full. Also the time is saved when the first differences between the golden model and the faulty model occurred. The security state checker starts also when the fault injection is activated the first time to check if any of the security features are activated or if the system reaches an unexpected state. When the execution of the fault injection is finished the control flow CPU reads out the PIFEA and write a log file for the test engineer. This flow is executed until all test patterns are tested.

V. Experimental Results

The experimental results are based on a ML507 board, which includes a Virtex 5 FPGA [16], [17]. For this FPGA board we synthesized a LEON3 processor in a single-core configuration. For the fault injection we used the MFI, which can be configured by the controlling CPU (for this test the PowerPC on the Virtex 5 FPGA). Also the PIFEA is connected to the controlling CPU.

A. *Saboteur configuration*

To prove if the PIFEA tool can detect critical faults we use saboteurs placed at the memory interface of the LEON3 processor. With an attack to the memory easily every required fault effect to prove the concept of the PIFEA can be produced. These effects the execution flow disturbance (e.g., manipulate variables of an if clause), memory access locations (e.g., manipulate a variable which is used as relative address) and output manipulations (e.g., manipulate a variable which should be sent to the environment).

B. *Hardware overhead*

The overhead of the system modifications compared to the system is shown in Table I. As seen in this table the MFI produces an overhead to the system. The required slices of the MFI do not increase linearly with the size of the system. The only parameter which increases the size of the MFI is the number of the saboteurs and the number of the triggers. The PIFEA module mainly increases the required memory. For the experimental results the BRAM of the FPGA is used as

TABLE I
REQUIRED NUMBER OF FPGA SLICES

	Test System	MFI	PIFEA
Number Slices	7811	2291	238
Number LUT	14081	2043	358
Number BRAM's	30	0	90
Total Memory(KB)	1080	0	3132

TABLE II
RUN TIME RESULTS FOR THE PROVE OF CONCEPT EXAMPLE.

	Simulation	Emulation	Speed-up factor
per run	610s	0.226s	2699
25 Test Cases	15105s	5.7s	2650
10000 Test Cases	est. 6042000s	2278s	2652

storage for the flow logging but every other external memory can be used instead.

C. Prove of Concept

To prove the concept of the PIFEA a simple test program is implemented as shown in Listing 1. This program is simulated and executed on an FPGA. As seen in Table II the emulation is much faster than an RTL simulation of a whole system. The prove of concept test is split into flowing 2 parts:

- Test if the program counter, memory accesses and IO flow detection works properly. Therefore, the line 04, 06 and 10 of the test program are attacked.
- Test if the PIFEA can detect if the system accesses a wrong memory region or if the system executes not allowed code. Therefore, one trigger of the PIFEA is set to the program counter of line 08 and a trigger is set to the memory address of the array[11]. Then the variables flow_control and i are attacked at line 05 or 06.

The results of the PIFEA prove of concept tests can be seen in Table III. As seen in this table attacks to different code lines have different effects and are differently reported by the PIFEA.

```
01:function test begin
02:   variable i=0,result=0,flow_contol=0;
03:      variable array[10]=0;
04:   for  i=1 to 10 do           //manipulate i
05:      if flow_contol==0 then   //manipulate flow_contol
06:         result+=array[i];     //manipulate i
07:      else
08:         ...                   //unallowed code
09:   end for
10:   output(result);             //manipulate result
11:end function
```

Listing 1. Test Program for Prove of Concept

VI. CONCLUSION

In this paper we presented a method how to analyze if a fault injection brings a system into an unintended state directly on an emulation platform. For this analysis all memory accesses, program counter values, and output information is stored and analyzed during the execution of the system-under-verification. Based on the result of this analysis the test engineer can easily find critical attacks out of the millions of fault attacks emulated. Another feature of this method is to allow for analyzing if the system executes unintended code after a fault injection or if a not allowed memory block is accessed. The prove of concept shows that this method works as indented.

TABLE III
MEASURED RESULTS FOR THE PROVE OF CONCEPT EXAMPLE.

Attack target line	Manipulated Value	Detected Effect
04	i	flow manipulation
05	flow_contol	flow manipulation not allowed PC
06	i	memory manipulation not allowed memory address
10	result	io manipulation

ACKNOWLEDGMENT

The authors would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology, which funded the Power-Modes project under the FIT-IT contract FFG 825749. We would also like to thank our project partners Infineon Technologies Austria AG and Austria Card GmbH.

REFERENCES

[1] B. Vigna, "More than moore: micro-machined products enable new applications and open new markets," in *Proc. IEDM Technical Digest Electron Devices Meeting IEEE Int*, 2005.

[2] G. Q. Zhang, F. van Roosmalen, and M. Graef, "The paradigm of "more than moore"," in *Proc. 6th Int Electronic Packaging Technology Conf*, 2005, pp. 17–24.

[3] *Common Criteria Evaluation and Validation Scheme Validation Report*, National Information Assurance Partnership Common Criteria Evaluation and Validation Scheme Std., 2005. [Online]. Available: onlineavailableonhttp://www.commoncriteriaportal.org/files/epfiles/st_vid10023-vr.pdf

[4] P. Ramachandran, P. Kudva, J. Kellington, J. Schumann, and P. Sanda, "Statistical fault injection," in *Proc. IEEE Int. Conf. Dependable Systems and Networks With FTCS and DCC DSN 2008*, 2008, pp. 122–127.

[5] R. Leveugle, A. Calvez, P. Vanhauwaert, and P. Maistri, "Precisely controlling the duration of fault injection campaigns: a statistical view," in *Proc. 4th Int. Conf. Design & Technology of Integrated Systems in Nanoscal Era DTIS '09*, 2009, pp. 149–154.

[6] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Proc. DATE '09. Design, Automation & Test in Europe Conf. & Exhibition*, 2009, pp. 502–506.

[7] R. Barbosa, "Fault injection optimization through assembly-level pre-injection analysis," Master's thesis, CHALMERS UNIVERSITY OF TECHNOLOGY Department of Computer Engineering Gteborg 2004, 2004.

[8] R. Barbosa, J. Vinter, P. Folkesson, and J. Karlsson, "Assembly-level preinjection analysis for improving fault injection efficiency," in *in Proceedings of the Fifth European Dependable Computing Conference (EDCC-5*, 2005.

[9] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, and J. Haid, "Modular fault injector for multiple fault dependability and security evaluations," in *Proc. 14th Euromicro Conf. Digital System Design (DSD)*, 2011, pp. 550–557.

[10] Gaisler, "Leon3 processor," Online, 07 2010. [Online]. Available: http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53

[11] A. Benso, M. Rebaudengo, M. S. Reorda, and P. L. Civera, "An integrated hw and sw fault injection environment for real-time systems," in *Proc. Symp. IEEE Int Defect and Fault Tolerance in VLSI Systems*, 1998, pp. 117–122.

[12] J. Carreira, H. Madeira, and J. G. Silva, "Xception: a technique for the experimental evaluation of dependability in modern computers," vol. 24, no. 2, pp. 125–136, 1998.

[13] J. H. Barton, E. W. Czeck, Z. Z. Segall, and D. P. Siewiorek, "Fault injection experiments using fiat," vol. 39, no. 4, pp. 575–582, 1990.

[14] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, 1997.

[15] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, and J. Haid, "Automatic saboteur placement for emulation-based multi-bit fault injection," in *Proc. 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip*, 2011.

[16] Xilinx, "Virtex-5 fpga family," Online, 07 2010. [Online]. Available: http://www.xilinx.com/products/virtex5/index.htm

[17] ——, "Ml505/ml506/ml507 reference design," Online, 2012. [Online]. Available: http://www.xilinx.com/support/documentation/boards_and_kits/ug349.pdf

Efficient Fault Emulation using Automatic Pre-Injection Memory Access Analysis

Johannes Grinschgl, Armin Krieg,
Christian Steger and Reinhold Weiss
Institute for Technical Informatics
Graz University of Technology
Graz, Austria
{johannes.grinschgl, armin.krieg,
steger, rweiss}@tugraz.at

Holger Bock, Josef Haid
Design Center Graz
Infineon Technologies Austria AG
Graz, Austria
{holger.bock, josef.haid}@infineon.com

***Abstract*—The complexity of SoCs has been increasing enormously over the last years. This increases the effort for testing the SoCs against natural external influences and fault attacks. These tests require a huge amount of time because of the large fault scenario space.**

In this paper a novel method is presented on reduction of system test duration. This speed-up is reached by observing memory accesses during a golden model run to find security relevant regions in memories. Therefore, a novel monitor module has been designed and tested which stores the used memory addresses together with the access time stamps.

***Index Terms*—fault emulation, fault injection controller, saboteurs, memory fault attacks, automatic memory fault analysis, pre-injection memory analysis**

I. Introduction

The complexity increase of semiconductors is based on the decreasing size of transistor structures. This decrease of size leads to a higher vulnerability against natural effects e.g., external radiation effects, thermal and electric degradation [1].

These effects can provoke permanent or transient faults which can lead to a change of system behavior. Because the location and the time of a fault cannot be foreseen, it is important to evaluate safety and security level of the system by fault detection methods. Especially if an attacker tries to provoke a fault, it is required that security relevant information is not exposed. Without security features it is no effort for an attacker to bring the system in an unintended state [2]. Therefore, security features have to be implemented. These features have to be tested to test if the selected methods are sufficient to reach targeted security levels.

Many different fault testing methods have been researched in the last years. These methods can be split into simulation, emulation, and physical tests. For early design phase testing of fault effects on a security critical system, simulation, and emulation methods are used. Especially emulation methods have shown that they can be used in a very effective way. The most commonly used platforms for fault emulation are FPGAs. An overview of such a fault emulation platform which is based on an FPGA is shown in Figure 1.

On FPGA platforms different methods for fault injection into emulated systems are known. The first one is partial

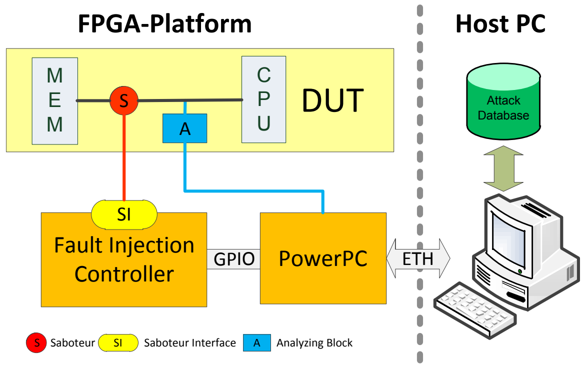


Fig. 1. Schematic view of the proposed fault injection system for Pre-Injection Memory Access Analysis (adapted from [5])

reconfiguration method which is based on the manipulation of the FPGA configuration during system operation. This method only works for some special FPGA devices (e.g., the Virtex families from Xilinx [3]).

The second method is to manipulate the circuit of the system-under-test by adding logic elements or controllable fault elements. These elements are also called saboteurs and mutants [4]. Saboteurs are modules which are placed between source and sink of signal lines. Not activated, these elements have no effect on the behavior of the system, but when activated they can disturb the selected signal. Some of the possible effects can be stuck-at-one, stuck-at-zero, delaying the signal flow and inverting of the input signal. Another system manipulation method are the mutants. A mutant is a modified system block which works as indented until it is activated. Then the block acts like a corrupted block.

For the usage of saboteurs or mutants it is required to have access to the hardware description of the system or the system must already support this features via a standardized test interface. An example of such a test interface could be scan chains [6].

The best fault injection platform is not of interest without having proper fault models. These fault models are different depending on the domain. In the dependability domain often random occurrence of natural faults is emulated. Therefore, single event upsets (SEU) fault models are a proper method to reproduce the effect of e.g., cosmic rays [7], [8]. For security evaluations such simple faults are not the only way for testing the effect of an attack because attackers who trying to bring the system in a mode where secret information can be extracted can easily disturb multiple bits. This leads to fault models which allow for specifying multi-bit faults. Testing the whole system regarding multi-bit faults within reasonable amount of time is unfeasible. Therefore, test strategies have to be implemented. Test strategies are also depending on the field of use. While in dependability systems the whole system has to be tested, in security systems only security relevant regions needed to be tested.

One security relevant target is the memory because critical data might be stored there (e.g., secret keys and access rights). The increasing size of memories on SoCs leads to an enormous effort to test if the system fulfills the security level. It is impossible to test every memory address at every time with all possible attack patterns. Therefore, some methods have to be implemented to reduce this effort. Random methods are used to inject faults into memory. This is especially very inefficient for large memories.

In this paper we propose a method to detect security critical memory regions and reduce the required fault injection emulation time. Especially for operating systems with dynamic memory management this is an important task because the location in the memory of security relevant variables is not known at compile time.

The main goals of this work can be summarized as follows:

- Increasing the fault emulation speed for testing the resistance of a system against memory fault attacks.
- Pre-injection analysis of memory access to find security relevant regions.
- Testing memory attacks independently of the used hardware (controller, memory) and software (operating system, programs).

This paper is structured as follows. Section II briefly shows the state of the art on fault injection campaigns acceleration. In Section III the design of the fault injection controller is described. Section IV describes the function of the pre-injection memory analysis concept. Section V shows some experimental results of this concept based on a LEON3 implementation [9]. Finally, conclusions are drawn in Section VI.

II. Related Work

The increasing complexity of system-on-chips leads to an time intensive testing process. Especially in high security applications the effort to test the system against fault attacks is a major part of the whole development cost [10]. Before the system can be sold as a high security product, it has to be certificated by a test laboratory. These tests are a very time consuming and costly step during the development process. To prepare for the certification process, the fault-resistance of a system is tested already in the concept phase. Therefore, many methods exist to cope with these issues. These methods can be split into two main groups, increase of the speed of one fault test and a pre-injection analysis.

The speed increasing of one fault injection test is done by testing the system on an FPGA based emulator platform [11]. The fault emulation approaches can be split into three main groups.

- **Partial reconfiguration** is a method, where a feature of SRAM-FPGA is used, to modify the netlist of a system during run-time. [12]–[17].
- **Mutants** are corrupted models which act like attacked block. [4], [17], [18]
- **Saboteurs** are blocks which are placed between source and sink of a signal. When activated they can manipulate the signal. [4], [18]

Compared to fault simulation these methods are very efficiently for fault testing. But the time required for a large number of tests is high. Therefore, the combination of the speed of emulation methods with pre-injection analysis can be used.

The pre-injection analysis observe the system during a golden model run and uses the generated information to reduce the required test scenarios. Berrojo et al. presented a method on the usage of read and write information for the reduction of fault injection sets. They analyzed in a golden model run the behavior of registers and their bit-flips. This information is used to generate structured fault emulation processes. For this method the system has to be analyzed during a simulation. The main disadvantage is that an RTL-level simulation for an operating system is a time consuming process. Barbosa et al. [19], [20] show a method on the usage of assembler code and op-code information to detect important attack regions in the design. They read out the op-code and analyze it to know what register is used and attack the registers directly before it is used. This reduces the number of fault scenarios. The disadvantage of this method is that it must be adapted for every core because if the structure of the op-code changes the injection system has to be modified. To the best of our knowledge there exists no pre-injection memory analysis method for accelerating fault emulations in literature.

III. Fault Injection Concept

This fault injection method is similar to modular fault injector (MFI) published in [5]. The aim of a modular fault injection concept is that the MFI can be adjusted exactly to the needs of the project where the MFI should be used. The MFI is based on four main parts seen in Figure 1.

1) **Attack Database:** Stores the attack and the attack scenarios
2) **Host PC:** The Host PC controls and defines the fault injection flow. With tools running on this PC a software developer can easily define which fault effect should be produced. Also the parameters for a long time test can be configured.
3) **Modular Fault Injection Controller:** This modular fault injection controller manages the injection process

on the FPGA. The controller gets different trigger signals as input and activates saboteurs depending on the attack which should be performed. The main important parts of the MFI are an internal memory to store the attack scenario, the control logic which activates and deactivates the saboteurs, and the trigger interface which can be configured for different trigger events.

4) **The Saboteurs:** Saboteurs inject the fault into the signal lines.

With this MFI it is possible to produce different fault effects in a reproducible way. In this paper a method similar to the MFI is used for the fault injection. The method is adapted with the pre-injection memory access analysis described in the following section.

IV. Pre-Injection Memory Access Analysis

The main problem of fault injection methods is to find important targets for the manipulation of the systems behavior. Especially the always increasing size of memory leads to an enormous effort to test every possible fault introduced into the system. As seen in Equation 1 the time(T) required for a whole test depends on two main parameters. The time per scenario and the number of scenarios. For a calculation example following values of a normal smart-card example are assumed:

Memory size	= 100KB
Run-time of the OS	= 100ms
Frequency	= 33MHz

If every byte of the memory should be attacked every 1000th clock cycle we get an overall time for the test of 33 million seconds. This is nearly a whole year which is not practicable. The target is to reduce this effort. The time per scenario is fixed because the test software always requires the same execution time. So the only way to decrease the overall time is by reducing the number of scenarios which must be tested. Therefore, a method has to be developed to allow for the analysis of the memory accesses to reduce the number of tests. To show the functionality of the novel proposed method we must analyze how source code functions access memory. As seen in Figure 2 security programs consist of areas where critical functions are executed. These functions normally only require a subset of the whole memory region. Analyzing these memory accesses manually is a very error prone and time consuming process. Therefore, a novel automatic analysis step is presented in this paper how memory accesses can be automatically analyzed and how this information can be used to reduce the number of required tests. As seen in Figure 3 not only the memory access is important for this analysis but also the information if it is a read or a write access must be considered. To generate the optimal test scenarios firstly all memory accesses have to be logged in a golden model run. This run does not significantly influence the overall test time (If there are thousands of injection runs it increases the test

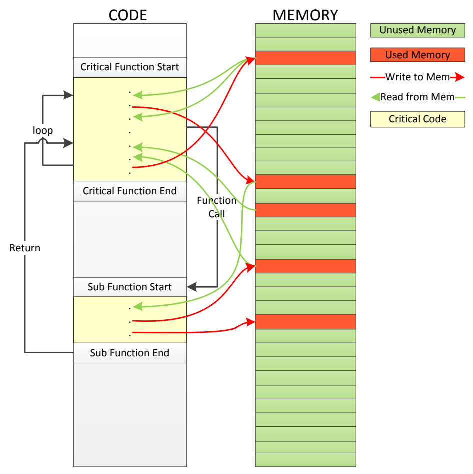


Fig. 2. Mapping between critical functions and used memory cells.

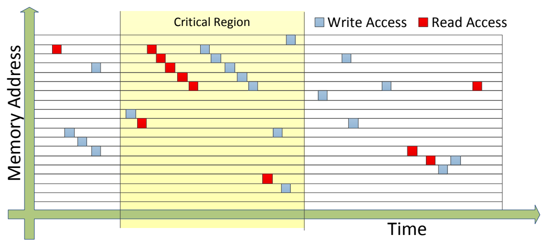


Fig. 3. Memory accesses split into memory read and write accesses during critical code execution.

time only slightly). Such a memory analysis works only if the memory accesses do not change from one run to the next.

$$T = n_{scenarios} * t_{scenario} \tag{1}$$

The system to perform such a memory log is build as shown in Figure 4. This system consists of the following blocks:

- **Memory:** The memory of the system-under-test is connected via a bus to the CPU. The address bus defines the memory address and the r/w signal defines if data is written or read out.
- **CPU:** The CPU requires the interface to the memory for reading out program code and data.
- **MFI:** The modular fault injector controls the fault injection process by activating saboteurs in the design. To synchronize the activation of the fault injection process trigger signals are used. In this case the program counter and some signals which indicate memory accesses are used as trigger signals. In the golden model run the MFI also controls the FIFO.
- **Cycle Counter:** The cycle counter counts the clock cycles starting at the last reset of the system. This information is used to map each memory access to a specific time after system reset.
- **FIFO:** The FIFO memory is used to store all memory accesses within the critical function during the golden model run. Therefore, the current memory address, the

read/write signal, the current cycle counter and the program counter are stored. The collected information can be read out by the Flow Controlling CPU.

- **Flow Controlling CPU:** The flow controlling CPU is an controller placed on the FPGA or a host PC outside of the FPGA which is used for the management of the whole flow.

The system allows for logging all memory accesses. Additional information which is stored are the current program counter value, the cycle counter value, and if the access was a write or a read. Only some areas of the code are security relevant so the FIFO can be adjusted that only memory accesses are stored from a start program counter till an end program counter. During this time it is possible to store one entry to the FIFO for every clock cycle, but to reduce the required memory of the FIFO only an entry is stored when a memory access is performed. If the FIFO is full, the emulation stops until some data is read out from the FIFO. After the golden model run the flow controlling CPU reads out all memory accesses and calculates a test set for the fault injection campaigns.

For this paper a test-set generation method is used where transient bit flips in the memory should be emulated. This means that after an attack the attacked byte is corrupted until the next write to this memory cell. The rules for the test-set generation are:

1) **Inject a fault to a memory cell before every read access:** Between two accesses to a specific memory cell only one fault injection needed to be emulated. The effect of the injection is the same if the fault is injected one or ten cycles before the read of the memory cell if there was no other memory access in between.
2) **Inject a fault to a memory cell until next write access:** Every injection is active until the next write to this memory cell, because a transient fault ends when a write access to the memory cell occurs.
3) **Do not inject before write accesses:** Between a read and a write access no fault injection must be started if there are other accesses to this memory cell during the critical code regions.
4) **Inject at last write accesses:** The last access to this memory cell should always be attacked until the next write outside of the critical memory region occurs.

Figure 5 shows an example of such attack scenarios on one memory address. The deactivation of the fault attack could differ from the picture because if the attack influences the system execution, the time for the next write access to the memory could be changed. Permanent fault attack scenarios are also shown in the figure and could be easily implemented by never deactivating the injected fault.

V. Experimental Results

To show the effectiveness of the memory attack scenario we implemented the proposed method on a widely used platform, the LEON3 SPARC V8 conformant processor by Gaisler Research [9].

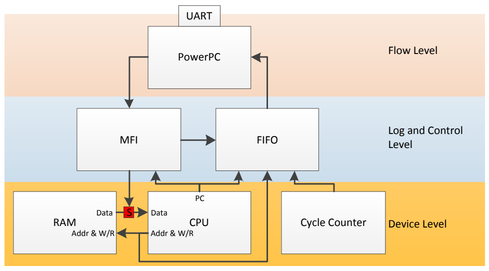


Fig. 4. Hierarchical model of the pre-injection memory access analysis hardware structure.

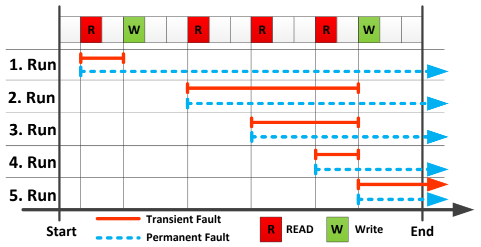


Fig. 5. Memory attack scenario on one address split into transient and permanent fault models.

We used a Xilinx ML507 [3] evaluation board with a Virtex5 FPGA. For the simulation results Modelsim (64bit, version 6.6) which is part of the ISE software package provided by Xilinx is in use. As simulation system a six-core 3.2 GHz AMD Phenom-II machine with 16GB of RAM is used.

A. LEON3 platform setup

The platform for the test is the LEON3 processor in a dual-core configuration (see Table I). For the fault injection the MFI can be controlled via the GPIO interface by the flow controlling CPU (in this case a PowerPC which is part of the Virtex 5 FPGA). Also the FIFO values can be read out by the flow controlling CPU over the GPIO bus.

TABLE I
LEON3 Configuration

Operating Frequency [MHz]	40
Instruction Cache Sets	2
Instruction Cache Set Size [kB]	
Data Cache Sets	1
Data Cache Set Size [kB]	4

B. Saboteur configuration

The saboteurs are placed at the interface between the LEON3 core and the memory. The exact location is between the CPU and the data cache. Therefore, it is possible to test the system at an abstract level because only virtual addresses are attacked. The test engineer does not need knowledge about the memory concept. This allows for reproducing every thinkable memory manipulation with the lowest required knowledge for the software tests. It is also possible to place saboteurs at the

TABLE II
FAULT INJECTION SETUP

	Test System	Test System & Per-Injection Memory Analysis	Overhead [%]
Number Slices	11401	12137	6.5
Number LUT	20832	20954	0.6
Number BRAM's	38	103	171
Total Memory(KB)	1332	3654	174

instruction cache. For these tests we limited the attacks to the data cache because the attack to the instruction cache works in the same way as for the data cache attack.

C. Hardware overhead

The pre-injection memory access analysis results in only minor overhead compared to the whole system (see Table II). Especially the Block RAM size is increasing because of the FIFO which stores the memory accesses, is build using BRAMS. The size of the FIFO can be easily configured based on the size of the FPGA. If the size of the FIFO is too small to capture all memory accesses of the critical regions the process can be split into several parts. Therefore, the emulation stops when the FIFO is full and waits until some data is read out from the FIFO. This increases only the time for the golden model run, but it has no influence on the fault injection campaigns because during the campaigns the FIFO will not be used.

D. Proof of Concept

To prove the concept of the memory attack concept a small calculation program was used. The intention was to test the system with a small test program allowing us to check the results by hand. Therefore, we used the code shown in Listing 1.

```
int test(){
        volatile int i=0,h=0;
        for(i=0;i<10;i++){
                h+=i;
        }
        return h;
}
```

Listing 1. Test Program for Proof of Concept

For the test we set a trigger point at the begin of test function and one trigger point at the end of the function. When the first trigger is activated we start to log all memory accesses. When the second trigger is reached we stop the logging. The emulation runs until the end of the program to estimate run-time of the program. This time is required to detect if the system is running in an infinite loop, during the fault injection campaign.

After compiling, this function consists of 22 assembler commands. To execute the program 47 memory accesses are required. See Table III to get an overview how many memory read and write accesses are required. The estimated number is generated manual by analyzing the assembler code. As seen in this table the number of estimated and measured memory accesses are identical.

TABLE III
MEMORY ACCESSES OF THE PROOF OF CONCEPT EXAMPLE.

	estimated value	measured value
Read	23	23
Write	22	22
Total Memory Accesses	47	47

TABLE IV
RESULTS FOR THE PROOF OF CONCEPT EXAMPLE.

	Simulation [s]	Emulation [s]	Speed-up factor
per run	603	0.224	2692
25 Test Cases	15065	5.6	2690
10000 Test Cases	est. 6030000	2260	2668

After the measurement the system automatically generated a total of 25 test cases. One test case is an fault injection at one memory address which starts at a special start time and ends at the the next write access to this memory cell. To test different manipulations to the memory cell this test case has to be executed several times. For running every test case once the fault injection flow requires 5.6s, which is approximately one injection every 224ms. From this 25 attacks 22 lead to a faulty result(see Table IV). The other 3 tests calculated the right result because not every injection has an influence on the system e.g., if a counter variable is compared to not equal zero and we change the value of this variable from 2 to 1.

The simulation of one test case requires approximately 2600 times longer than one emulated test-case of the system. To compare the attack scenario to a test scenario without the improvement of cache analysis we calculated with 10000 test cases. 10000 test cases are used because we estimate that at least 100 addresses have to be tested. Each of these address should be tested at 100 different points in time. Therefore, the pre-injection memory access analysis results in a speed-up of nearly 400.

E. AES Example

To show that the approach also works on a larger test program we used an AES implementation from [21]. We attacked only the encryption of a 16 Byte plain text. The memory logging is only activated when the encryption function is executed. From the analysis step we get the following results as seen in Table V. There are 361 data memory accesses during the encryption function. The execution of this function requires 2584 clock cycles. To calculate the speed-up compared to an attack scenario without pre-injection memory analysis we used this number of 2584 clock cycles as reference. The second value which is important for the calculation is how many memory addresses have to be attacked. There are 163 addresses used during the encryption process. These memory cells are distributed over the memory. The difference between the highest and the lowest used ROM address is 255 and the difference between the highest and the lowest used RAM address is 712. These values are used to calculate the estimated number of fault attacks which have to be performed to test

TABLE V
DATA MEMORY ACCESSES DURING AES ENCRYPTION

Read	250
Write	111
Total Memory Accesses	361
Rom Addresses	121
Ram Addresses	42
Total Addresses	163
Total Clk Cycles	2584
Test Cases (without our approach)	249873
Test Cases (with our approach)	249
Speed-up factor	1003

TABLE VI
SIMULATION AND EMULATION TIME FOR THE AES EXAMPLE

	Simulation [s]	Emulation [s]	Speed-up factor
per run	480	0.5	960
249 Test Cases	est. 119520	120	996
249873 Test Cases	est. 119939040	119050	1007

the system. If every memory cell within the used ROM and used RAM address range is attacked every ten clock cycles we get an total of 249873 test cases. As seen in Table V our approach only has to perform 249 test cases. This is a speed-up of approximately 1000. Table VI shows the required time for simulating and emulating these test cases.

VI. CONCLUSION

This paper presents a hardware independent method for fault injection acceleration by pre-injection analysis of memory accesses. For this analysis all memory accesses within a critical code segment are stored in a FIFO and are analyzed. Based on the result of this analysis the test campaign is calculated by using simple rules as defined in Section IV.

An advantage of this design is its flexibility, because only the memory interface of the system-under-test has to be connected to the pre-injection memory analysis tool. It is usable for every system which contains memory. The method has a scaleable hardware overhead so it can be used on every FPGA. The results show that this flow can provide a speed-up for a fault injection flow.

Our future work is to further reduce the number of required fault injection campaigns by using hit and miss information of cache accesses.

ACKNOWLEDGMENT

The authors would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology, which funded the Power-Modes project under the FIT-IT contract FFG 825749. We would also like to thank our project partners Infineon Technologies Austria AG and Austria Card GmbH.

REFERENCES

[1] M. Nicolaidis, "Time redundancy based soft-error tolerance to rescue nanometer technologies," in *Proc. IEEE VLSI Test Symposium (1999)*, 1999, pp. 86 –94.

[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*, vol. 94, no. 2, pp. 370–382, feb 2006.

[3] Xilinx, "Virtex-5 fpga family," Online, 07 2010. [Online]. Available: http://www.xilinx.com/products/virtex5/index.htm

[4] J. Boue, P. Petillon, and Y. Crouzet, "Mefisto-l: a vhdl-based fault injection tool for the experimental assessment of fault tolerance," in *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, 1998. Digest of Papers.*, 23-25 1998, pp. 168 –173.

[5] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, and J. Haid, "Modular fault injector for multiple fault dependability and security evaluations," in *Proc. 14th Euromicro Conf. Digital System Design (DSD)*, 2011, pp. 550–557.

[6] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, and M. Violante, "An fpga-based approach for speeding-up fault injection campaigns on safety-critical circuits," *Journal of Electronic Testing*, vol. 18, no. 3, pp. 261–271, 2002, 10.1023/A:1015079004512. [Online]. Available: http://dx.doi.org/10.1023/A:1015079004512

[7] A. Benso and B. Prinetto, *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*. Kluwer Academic Publishers, 2003.

[8] A. V. Karapetian, R. R. Some, and J. J. Beahan, "Radiation fault modeling and fault rate estimation for a cots based space-borne supercomputer," in *Proc. IEEE Aerospace*, vol. 5, 2002, pp. 5–2121.

[9] Gaisler, "Leon3 processor," Online, 07 2010. [Online]. Available: http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53

[10] GAO, "National partnership offers benefits, but faces considerable challenges," *United States Government Accountability Office*, 2006. [Online]. Available: http://www.gao.gov/new.items/d06392.pdf

[11] A. Pellegrini, K. Constantinides, D. Zhang, S. Sudhakar, V. Bertacco, and T. Austin, "Crashtest: A fast high-fidelity fpga-based resiliency analysis framework," in *Proc. IEEE Int. Conf. Computer Design (ICCD 2008)*, oct 2008, pp. 363–370.

[12] L. Antoni, R. Leveugle, and M. Feher, "Using run-time reconfiguration for fault injection in hardware prototypes," in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT 2002)*, 2002, pp. 245–253.

[13] C. Lopez-Ongil, L. Entrena, M. Garcia-Valderas, M. Portela, M. A. Aguirre, J. Tombs, V. Baena, and F. Munoz, "A unified environment for fault injection at any design level based on emulation," *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 946–950, 2007.

[14] M. Jeitler, M. Delvai, and S. Reichor, "Fuse - a hardware accelerated hdl fault injection tool," in *Proc. SPL Programmable Logic 5th Southern Conf*, 2009, pp. 89–94.

[15] L. Kafka, "Analysis of applicability of partial runtime reconfiguration in fault emulator in xilinx fpgas," in *DDECS '08: Proceedings of the 2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 1–4.

[16] B. F. Dutton, M. Ali, C. E. Stroud, and J. Sunwoo, "Embedded processor based fault injection and seu emulation for fpgas," in *Proc. International Conf. on Embedded Systems and Applications 2009*, 2009, pp. 183–189.

[17] R. Leveugle and K. Hadjiat, "Multi-level fault injections in vhdl descriptions: Alternative approaches and experiments," *Journal of Electronic Testing*, vol. 19, no. 5, pp. 559–575, 2003.

[18] P. Ellervee, J. Raik, K. Tammemae, and R.-J. Ubar, "Fpga-based fault emulation of synchronous sequential circuits," *IET Computers & Digital Techniques*, vol. 1, no. 2, pp. 70–76, 2007.

[19] R. Barbosa, J. Vinter, P. Folkesson, and J. Karlsson, "Assembly-level preinjection analysis for improving fault injection efficiency," in *in Proceedings of the Fifth European Dependable Computing Conference (EDCC-5*, 2005.

[20] R. Barbosa, "Fault injection optimization through assembly-level pre-injection analysis," Master's thesis, CHALMERS UNIVERSITY OF TECHNOLOGY Department of Computer Engineering Gteborg 2004, 2004.

[21] B. Gladman, "Implementations of AES (Rjindael) in C/C++ and assembler," December 2010. [Online]. Available: http://gladman.plushost.co.uk/oldsite/cryptography_technology/index.php

POWER-MODES - POWer EmulatoR and MOdel based Dependability and Security evaluations

ARMIN KRIEG, Graz University of Technology
JOHANNES GRINSCHGL, Graz University of Technology
CHRISTIAN STEGER, Graz University of Technology
REINHOLD WEISS, Graz University of Technology
HOLGER BOCK, Infineon Technologies Austria AG
JOSEF HAID, Infineon Technologies Austria AG

Innovation cycles have been shortening significantly during the last years. This process puts tremendous pressure on designers of embedded systems for security or reliability critical applications. Eventual design problems not detected during design time can lead to lost money, confidentiality or even loss of life in extreme cases. Therefore it is of vital importance to evaluate a new system for its robustness against intentionally and random induced operational faults. Currently this is generally done using extensive simulation runs using gate-level models or direct measurements on the finished silicon product. These approaches either need a significant amount of time and computational power for these simulations or rely on existing product samples.

This paper presents a novel system evaluation platform using *power emulation* and *fault injection* techniques to provide an additional tool for developers of embedded systems in security and reliability critical fields. Faults are emulated using state-of-the-art fault injection methods and a flexible pattern representation approach. The resulting effects of these faults on the power consumption profile are estimated using state-of-the-art power emulation hardware. A modular system augmentation approach provides emulation flexibility similar to fault simulation implementations. The platform enables the efficient evaluation of new hardware or software implementations of critical security or reliability solutions at an early development phase.

Categories and Subject Descriptors: B.2.3 [**Reliability, Testing, and Fault-Tolerance**]: ARITHMETIC AND LOGIC STRUCTURES

General Terms: Design, Security, Reliability

Additional Key Words and Phrases: Hardware security, Hardware reliability, Fault attack resistance, Power estimation, Fault emulation

ACM Reference Format:
Krieg, A., Grinschgl, J., Steger, C., Weiss, R., Bock, H., Haid, J. 2011. POWER-MODES - POWer EmulatoR and MOdel based Dependability and Security evaluation. ACM Trans. Reconfig. Technol. Syst. 0, 0, Article 0 (2012), 19 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

This work is supported by the Austrian Federal Ministry for Transport, Innovation, and Technology, which funded the Power-Modes Project under the FIT-IT contract FFG 825749. Thanks are also going to our project partners Infineon Technologies Austria AG and Austria Card GmbH.
Author's addresses: A. Krieg and J. Grinschgl and C. Steger and R. Weiss, Institute for Technical Informatics, Graz University of Technology; H. Bock and J. Haid, Design Center Graz, Infineon Technologies Austria AG.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

The increasing silicon integration density brought tremendous challenges for system engineers with it. First, an ever increasing number of new features have to be implemented in ever decreasing implementation cycles. Second, a growing number of these highly integrated systems are used in critical fields with high demands on system security and reliability. Therefore system designer support is needed during the design phase to test new hardware and software implementations for possible weaknesses [Ravi et al. 2004].

Especially for high safety applications (e.g. space, automotive, aeroplane) a wide variety of fault injection techniques has been developed [Arlat et al. 1990; Jenn et al. 1994; Leveugle and Hadjiat 2003]. In this field faults are mostly of a random nature and therefore these evaluation approaches rely on a random selection and injection of operational errors. For these platforms generally no fault attacks have been considered because of their controlled nature [Leveugle 2007]. Otherwise for modern security evaluations it is of vital importance to know the impact of faults on the power consumption profile because it forms a direct (simple power analysis - SPA) or indirect (differential power analysis - DPA) information source to an adversary (e.g. [Kocher et al. 1999; Roche et al. 2011]). To estimate the power profile during run-time hardware accelerated methods like *power emulation* have been proposed. These techniques can be used to close a gap often remaining in classic fault injection platforms for reliability purposes. Such a gap constitutes a serious problem in security critical systems as even the reset behavior during an attack already provides information to the attacker as depicted in Figure 1. Such information allows to draw conclusions about internal fault reaction mechanisms like emergency memory accesses that would lead to increased consumption or a decreased power profile if a simple device reboot is triggered (such effects have been shown in [Bar-El et al. 2006]).

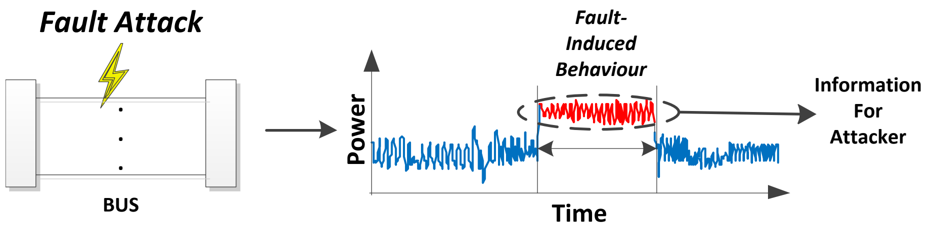


Fig. 1. Influence of security reset on power consumption profile

To evaluate hardware and software implementations for their fault robustness these techniques have been combined to form a fault effect evaluation platform. The main contributions of this work are

— A novel evaluation flow combining *fault injection*, *power emulation*, *power* and *fault analysis* techniques.
— The introduction of a novel FPGA-based platform for run-time fault-attack and dependability evaluations.
— Case studies using a common AES software and hardware implementation and general purpose processor.

This paper is structured as follows. In Section 5 a brief overview over the current state-of-the-art concerning similar evaluation platforms is given. Section 2 contains a short introduction into *fault injection* methods and applications. Followed by Section 3 summarizing the main characteristics of different power analysis methods. Section 4 gives an introduction into power emulation and its application for power analysis purposes. In Section 6 the principle methodology of the fault effect evaluation platform is presented. Followed by Section 7 showing some experimental results to prove the viability of our approach. Finally Section 8 gives a short conclusion about our work and some views into future developments in this field.

2. FAULT INJECTION (FI)

The term fault injection describes the intentional introduction of faults into a system to simulate errors that are caused by external influences changing the system's normal behavior. Faults can be injected on several different levels of abstraction:

— **Hardware-Level**: This level requires the least amount of system-invasion as existing test equipment can be reused. Faults can be injected on pin-level by manipulating data and control flow [Arlat et al. 2002]. Internal manipulation is enabled through radiation sources or by direct injection into the silicon device [Gunneflo et al. 2002]. The disadvantage of fault injection on this level is the need for existing prototype samples.
— **Software-Level**: The basic principle behind software fault injection is the same as in hardware FI [Tsai et al. 2002]. A comprehensive software testing environment is presented in [Segall et al. 2002].
— **Modeling-Level**: As shown in Section 5 direct evaluation of hardware description level models is of utter importance to embedded system designers. As shown in [Baraza et al. 2002] faults can be introduced using simulator commands or by augmentation of the descriptive code itself.

For this fault injection platform we will concentrate on hardware-description based approaches. In this case faults can be introduced using simulator commands in hardware model simulators or by adapting the model description itself for hard-accelerated emulation.

2.1. Simulation-based

The advantage of simulation-based approaches is that no changes to the model description itself are necessary. The complete model simulation flow can be reused for fault injection examinations. Faults are activated through simulator commands to manipulate certain states of the system. This can be done because every element of system design is accessible during simulation. The disadvantage of this approach is that a significantly amount of computational power is necessary for detailed simulations of complex designs. Especially if the abstraction level is lowered to the gate-level lengthy simulation runs are necessary for comprehensive evaluations.

2.2. Adaption-based

Adaption-based fault injection approaches augment the existing hardware description with additional circuitry to influence the behavior of predefined system elements. While specific knowledge about the location of faults manifesting themselves in form of errors is necessary, very high evaluation performances are enabled. Especially in combination with emulation techniques using FPGA prototyping platforms large amounts of different fault patterns can be evaluated in relatively short periods of time [Grinschgl et al. 2011b]. Two different concepts of manipulative elements are generally used:

— **Mutant**: In this case a complete system module is replaced with a manipulated version of this module. During normal operation this module works as designed but after fault activation it behaves like a defective one.
— **Saboteur**: Saboteurs are small elements that are inserted into signal lines. During normal operation these act transparent but after fault activation a predefined fault effect influences the targeted signal.

3. POWER ANALYSIS (PA)

In standard CMOS technology power consumption and emitted electro-magnetical radiation of a given system is dependent on the amount of simultaneously switching transistors. Therefore it not only depends on the control flow of a program but also on the data it processes. This fact is exploited by power analysis approaches using timing analysis or statistical methods to extract secret information like the encryption key through this externally available power information. This powerful tool therefore became a strong companion of adversaries and cryptologists alike.

3.1. Simple Power Analysis (SPA)

In case of SPA only a small number of power consumption radiation traces are recorded for further analysis. These are analyzed in a direct manner, meaning that these traces are evaluated for specific algorithm dependent variations and timing characteristics. SPA is therefore based on a variant of control flow analysis of cryptographic workload. To efficiently execute this evaluation at least basic knowledge about the used algorithm or implementation is needed.

Classic programming language elements that result in strongly visible power profile variations are branches. Depending on the branch condition these also lead to strong timing differences that could be detected by an adversary. Therefore software implementations of cryptographic algorithms must never be based on branches to hinder SPA analysis. Another effective countermeasure against such power analysis methods is execution randomization to disconnect execution behavior from the algorithmic behavior.

3.2. Differential Power Analysis (DPA)

DPA combines the power or radiation profile analysis techniques presented in Subsection 3.1 with statistical methods. Since its publication by [Kocher et al. 1999] it has been proven to be a powerful tool for the extraction of secret information from cryptographic devices. While SPA is only based on visible trace variations caused by differences in the execution of the algorithm, DPA exploits trace information depending on data-dependent transistor switching. While the original presented techniques required high amounts of power or radiation records if simple countermeasures are present, newer methods reduced these significantly. Especially noticeable is correlation DPA as shown in [Brier et al. 2004] and [Mangard et al. 2007].

Countermeasures could be specialized logic styles and shielding to flatten emitted leakage profiles. Randomization and masking techniques help to counteract statistical methods and power consumption assumptions. Generally it is more difficult to efficiently fight statistical power analysis attacks. As shown in [Schaumont and Tiri 2007] some countermeasure combinations may even be counterproductive.

4. POWER EMULATION (PE)

The wish for efficient power estimation methodologies led to the development of the power emulation technique, i.e., the hardware-accelerated power evaluation of a given system-under-test. It became a promising alternative to simulation-based power profiling approaches and its hardware-based nature allows the integration into embedded systems.

4.1. Principle of Power Emulation

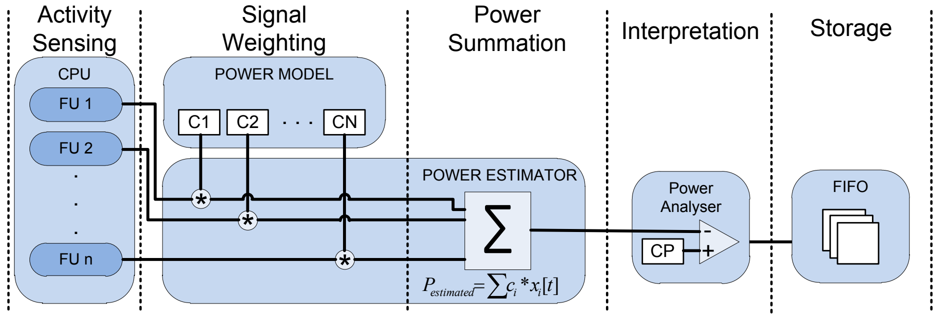


Fig. 2. Power Emulation - Principle Architecture (adapted from [Krieg et al. 2011b])

Coburn et al. introduced the principle of *power emulation* (PE) in [Coburn et al. 2005] as a state-of-the-art functional emulation using hardware-implemented power models. The functional emulation and power estimation process are executed concurrently during the run-time of the system.

As depicted in Figure 2 the generated approximated power values can be immediately used for further processing or saved as traces in case of power consumption evaluations.

4.2. Extended Power Emulation Unit

Our power evaluation platform is partially based on a high-level *power emulation unit* similar to the one introduced in [Genser et al. 2009]. It comprises a direct implementation of the principle architecture depicted in Figure 2.

Its basic operation utilizes additive linear power macro models as expressed in Equation 1 to generate static and dynamic power consumption estimates. These values are generated for different system components by monitoring its power-relevant control signals x_i. For each member of the control signal set of size Nc a weight c_i and for each observed data-line a value d_i in a set of size Nd inside the macro model is defined. The static power consumption is considered by coefficient c_0. Resulting in a combined power model size of $Nc + Nd + 1$ coefficients that have to be implemented inside the emulation architecture.

$$\hat{P}[t] = \hat{P}_{sta} + \hat{P}_{dyn}[t] = c_0 + \sum_{i=1}^{Nc} c_i x_i[t] + \sum_{i=1}^{Nd} d_i x_i[t] \tag{1}$$

To enable the usage of this power estimation principle for DPA evaluation, its power model has to be extended for the usage of not only state-dependent (SD) but also data-dependent (DD) signals. The original design of the PE unit was based on the analysis of the state of various system components by monitoring selected power-relevant control signals. This approach is only viable for pure power consumption estimation evaluations within a reasonable accuracy. A power model designed only under the consideration of a system's control flow is typically not usable for security evaluations because of the missing dependency between power profile and processed data. Therefore these power models have to be extended with basic signal switching information of data-processing architecture elements. Such an extended power emulation hardware module and characterization process has been presented in [Krieg et al. 2011a].

In case of our chosen target architecture we selected both operand registers of its arithmetic logic unit (ALU). The elements of these operands have to be pre-processed to extract only switching information. This approach allows to introduce data-dependency into the power consumption model while resulting in a very similar *power emulation* result. This is achieved by data-line switching activity extraction to simulate the current peak behavior of CMOS inverters. Such data-line weights can be either defined dynamically to investigate worst- and best-case scenarios, or by extraction of signal line capacitances (as provided by the RC extraction phase of common semiconductor back-end tools). Our generated power models have been evaluated and compared to gate-level power simulations and their accuracy has been confirmed to be similar to power models only concerning control signals. The evaluation of software side-channel leakage countermeasures can be enabled by scaling only these data-dependent model coefficients. Meaning, leakage can be intentionally over and under estimated to simulate well and less optimal routed designs resulting in different gate capacitances.

To enable our FPGA-based evaluation platform for simple run-time power tracing the power estimation unit has been additionally extended with an internal and configurable FIFO buffer. This memory allows the storage of the trace of a single encryption run without control interference of the cryptographic algorithm. After such an evaluation run has been finished its contents can be safely written to an external memory target.

4.3. Extended Power Characterization for DPA Evaluations

An automated power characterization methodology has been presented in [Bachmann et al. 2010]. A similar approach has to be extended to enable the generated power models consisting of both state-dependent and data-dependent model coefficients. The power characterization flow as depicted in Figure 3 can be described by the following stages:

(1) **Pre-Processing**: To gain a good coverage of all components of a given system-under-test, a set of microbenchmarking applications is simulated and power-profiled using state-of-the-art gate-level power estimation tools (Synopsys PrimeTimePX). This simulation process results in training-sets containing signal activity and power estimation data.

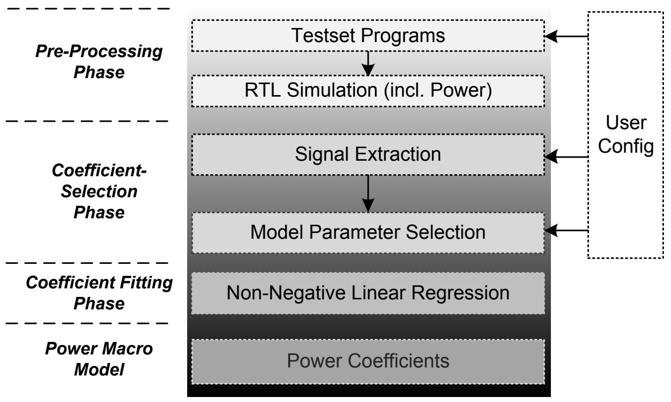

Pre-Processing Phase
Coefficient-Selection Phase
Coefficient Fitting Phase
Power Macro Model
Testset Programs
RTL Simulation (incl. Power)
Signal Extraction
Model Parameter Selection
Non-Negative Linear Regression
Power Coefficients
User Config

Fig. 3. Automated power characterization methodology for power evaluations

(2) **Coefficient selection**: Our characterization approach applies several filters to the training set data retrieved during the previous stage to remove signals that show no (zero-activity filter), irrelevant or redundant activity (cross-correlation filter). Basing on this filtered data-set power-relevant signals, i.e., power model parameters x_i are determined according to their correlation to the gate-level power simulation data. The completely automated approach presented in [Bachmann et al. 2010] only considers control-flow-related signals for power modeling. Therefore we extended this approach by the addition of data signals based on a user-defined configuration. These data signals have to be preprocessed using an in-house VCD-file analyzer solution to extract only its switching activity.
(3) **Coefficient fitting**: The resulting parameters out of the selection process are now used for the determination of coefficients c_i by means of a model coefficients fitting process. This process is using a non-negative linear regression technique. Finally a power model containing both state- and data-dependent coefficients is generated.

5. RELATED WORK

This work includes the application of a variety of technologies to gain a higher level view of complex security problems. These techniques can be divided into three main groups. First, emulation-based power estimation methodologies to retrieve accurate cycle-accurate power estimates at early stages of the design process. Second, security evaluations of cryptographic software and hardware implementations rely on the application of power-analysis techniques as soon as possible during the design flow. Finally, fault injection platforms have been presented to confront hardware models or manufactured devices with eventual occurring faults.

5.1. Emulation-Based Power Estimation

In [Coburn et al. 2005] a brief overview on the basic power emulation principle is given. Run-time improvements by power estimation hardware-acceleration of about 10x to 500x compared to commercial power estimation tools are achieved. The authors also introduced strategies to minimize the hardware overhead caused by the emulation methodology. In later work this approach has been extended in [Ghodrat et al. 2007] into a hybrid power estimation technique for complex SoCs. Power analysis times have been significantly reduced by this framework through the use of combined simulation and emulation techniques. A main advantage of hardware-assisted power estimation is the possibility of direct interpretation during run-time of a process. Such an approach, used for the power-aware process migration between cores has been presented in [Bhattacharjee et al. 2008].

5.2. Early Power Analysis Techniques

The first approach comprises a purely software-based solution [Shumov and Montgomery 2010]. For extracting side channel leakage information from a given program, it is executed and every instruction and processor state is traced. This information is fed into an easily extensible analysis module. Such analysis modules are directly implemented for the Microsoft Debugger API. The authors do not state any information about accuracy and simulation performance.

The second approach uses in-system knowledge provided by power consumption simulators using instruction set simulators. While promising high simulation speeds this approach is mostly limited to thoroughly profiled general purpose architectures. Implementations of this approach were presented in [den Hartog and de Vink 2005] and [Thuillet et al. 2009]. Both solutions promise high simulation speeds, but do not present any hints about the accuracy of the used power profiles. Inaccurate power profiles could lead a software engineer into a sense of false security as the possibility of missed information leakage could be higher than anticipated. The implementation presented in [Thuillet et al. 2009] uses several abstract power models to isolate possible sources of leakage. The presented approach has been designed for the Atmel AVR microcontroller series.

The third approach uses highly accurate gate- or even transistor-level simulations to identify critical parts of the circuit-under-test. This level of accuracy comes with high costs in terms of simulation speed and needed simulation equipment. The designer therefore has to choose between very short simulation times or reduction of the evaluation to the smallest possible leakage contributer. The former choice is mostly infeasible because it is of utter importance to know if an adversary could extract information using a high number of power traces. Works based on highly accurate analog simulations using SPICE simulators [Li et al. 2005; Regazzoni et al. 2007; Regazzoni et al. 2009] and using register transfer level (RTL) simulations [Bucci et al. 2007] for improving simulation speeds have been presented. However, the timely effort of offline power simulations remains a limiting factor for the trace count. The authors of [Bucci et al. 2007] also note that RTL simulations may hide leakage contributors only visible after further design flow steps.

5.3. Fault Injection Platforms

To improve the reliability of high-safety applications several fault injection techniques have been introduced [Arlat et al. 1990; Jenn et al. 1994; Leveugle and Hadjiat 2003]. Faults can be injected into completely manufactured parts e.g. using radiation or during the design phase using hardware description manipulations. The latter can be divided into simulation and emulation based approaches depending if a model of the implementation is simulated or emulated on a FPGA platform.

Because high-level hardware descriptions can be simulated directly during the design phase, first fault injection tools for such hardware-models were based on simulation techniques. One of these first simulation tools was MEFISTO specifically targeted on fault injection into VHDL models [Jenn et al. 2002]. A higher level approach using the system-level description language SystemC has been shown in [Rothbart et al. 2004]. A pure simulation-based technique has been complemented with automatized saboteur and mutant insertion strategies for the VFIT tool in [Baraza et al. 2006]. Further improvements concerning the injection performance of simulation and emulation approaches have been presented in [Valderas et al. 2007].

The need for higher fault injection coverage led to the heavy research activity in the field of emulation techniques. As shown in [Leveugle 2002] emulation promises significantly higher injection rates and therefore enables the possibility to evaluate more possible fault configurations than using simulation. The usage of FPGAs for hardware prototyping also allowed to use novel reconfiguration techniques for fault injection as presented in [Antoni et al. 2002]. Since the introduction of these techniques several improvements have been introduced in [Zheng et al. 2008; Daveau et al. 2009].

All these presented approaches have been developed for dependability evaluations, meaning that in most cases only single faults have been considered. As shown in [Leveugle 2007] security evaluations need more complex fault models as intentional faults could happen at several positions at once.

5.4. Discussion

Existing work concerning the evaluation of SoC designs at early design stages had strong focus on a single problem. Therefore, physical and emulation-based approaches have been introduced for dependability evaluations and simulation-based techniques have been the prime choice for security investigations. This results of the need for performance to satisfy statistical models covering random fault models and high behavioral accuracy for the investigation of circuit effects as gate switching or faults resulting from laser attacks.

As long as all parts of the system are investigated by the same entity, this is a valid procedure as the test engineers can use very information about the software and hardware implementation. In large SoCs and in smart-cards this is usually not the case. The hardware is for example provided by a large semiconductor manufacturer and the software is developed by a specialized company that targets a specific market sector. Especially in the smart-card section a wide range of rules has been defined for security evaluation to ensure the combination of the hardware and software fulfills a certain level of trust. Because the hardware manufacturer will try to hide the details of the secure implementation, it is not possible to provide such detailed information like a comprehensive simulation model to the customer. Our modular fault injection approach allows the hardware manufacturer to provide a debugging tool that abstracts the fault effect visible on software level from the hardware effects causing these effects.

Finally, the power consumption aspect has been completely disregarded in literature. In systems with a limited power budget a compromise has to be found between the most secure hardware architecture and a reasonable consumption behavior. A characterization and test environment to determine an optimized software and hardware implementation that fulfills both, power and security targets is therefore strongly needed. To the best of our knowledge there is no comprehensive approach providing power analysis, fault analysis and a complete power characterization flow, described in literature.

6. EVALUATION PLATFORM

The combination of fault injection and power emulation techniques opens a variety of new design analysis possibilities. Because of their hardware accelerated nature they can be integrated in a combined form into an FPGA-based platform.

6.1. Emulation Hardware Generation Flow

As shown in Figure 4 the hardware accelerated emulation architecture generation flow has been divided into the following phases:

— **Model Creation Phase:** In this phase a high-level fault event is mapped to a circuit-internal fault model. It is not necessarily run through during every fault emulation campaign. Additional power coefficient characterization has to be performed if significant hardware changes invalidated a previous power emulation implementation or if the flow is applied to a new design.

— **VHDL Augmentation Phase:** According to the chosen fault model and location, saboteurs have to be placed into the system to emulate a chosen fault effect. if necessary for further analysis steps, additional analysis modules also have to be placed during this stage.

These tasks have to be completed by the hardware designer to prepare the design and fault patterns for further distribution.

Model creation phase. The fault model itself can be divided into a low-level and a high-level view. The low-level fault model concerns the effects directly influencing the system on circuit-level. Such fault models could be stuck-at, indetermination, delay, open, close or bit-flip faults. A more detailed overview over such low-level fault-models is given in [Baraza et al. 2006]. High-level fault models can include many low-level effects resulting in a specific influence on normal program execution. These models are specifically important for security evaluations which include power- or light attacks that lead to several faults happening at the same time [Leveugle 2007]. Currently we are not able to provide an automated approach to generate such high-level models and therefore good evaluations results rely on the experience of the test designer.

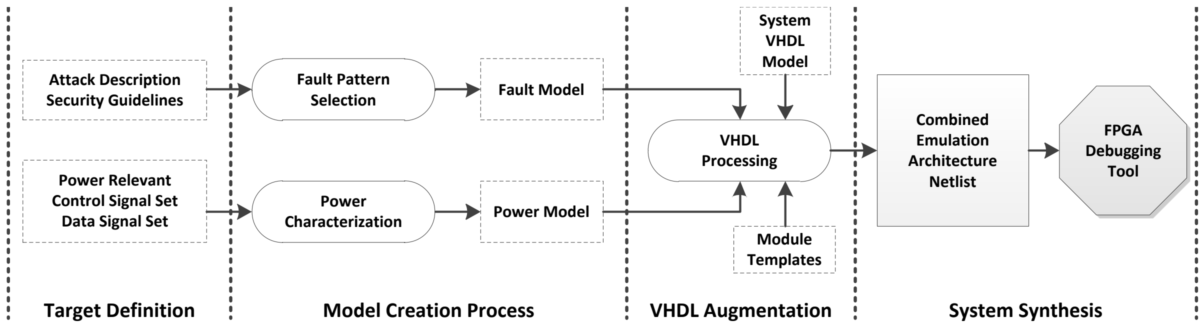


Fig. 4. Emulation hardware generation flow

If the power consumption behavior of the targeted system changed significantly, the system has to be re-characterized to retrieve an updated power macro model. The process has also been automatized using MATLAB for coefficient selection, characterization and verification. The verification process compares the derived power model to cycle-accurate power simulation results for a chosen set of verification programs.

VHDL Augmentation Phase. After the selection of interesting fault models fault locations can be determined. Onto each selected fault location a saboteur has to be placed to emulate the desired effect at the correct position. Selection, activation and configuration signals of these saboteurs then have to be routed to the top level to be connected to the fault injection controller. Because of the complexity of modern processor designs this process has been completely automated using a VHDL parser application based on the VMagic VHDL code generator [Pohl et al. 2009]. The complete placement process is shown in [Grinschgl et al. 2011a].

6.2. Fault Emulation Architecture

All parts of the proposed fault emulation platform are designed in a technology-independent way to allow for the application to FPGA evaluation systems of different manufacturers. Meaning all augmentations are implemented on a RTL-level to rely on existing implementation flows as provided by the evaluation board manufacturer. The basic hardware accelerated fault emulation platform is depicted in Figure 5.

In case of a Virtex5-FXT based FPGA system an integrated PowerPC processor can be used for fault injection control. The generalized GPIO-interface allows to use any hard- or soft-core for control. Also an AMBA-interface is available if an in-system solution is desired. The fault injection controller can be pre-programmed to listen for a specific program counter (PC) value. An internal counter is used to start the injection process when the PC-of-interest is called a pre-determined amount of times.

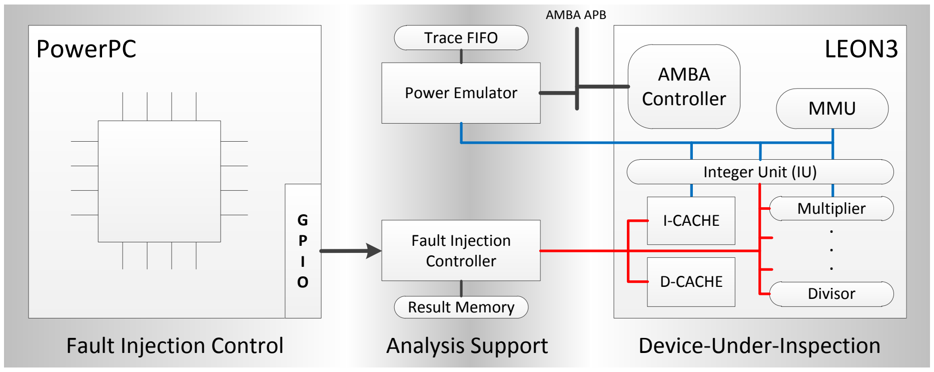


Fig. 5. Hardware accelerated fault emulation architecture

6.3. System evaluation flow

After all characterization and generation tasks have been executed and proper emulation architecture has been derived, the resulting netlist can be loaded onto a FPGA-based evaluation device. As shown in Figure 6 the following phases have to be passed through to gain information about system power consumption, leakage and fault robustness.

— **Profiling phase:** In the profiling phase test programs are executed on the target platform while faults are continuously injected to gain knowledge about the possible execution effects.
— **Analysis phase:** All data gathered during the profiling phase is now thoroughly examined to detect reasons and develop countermeasures against undesired execution errors.
— **Decision phase:** Finally analysis results are used to select software or hardware measures to improve overall robustness of the target architecture.

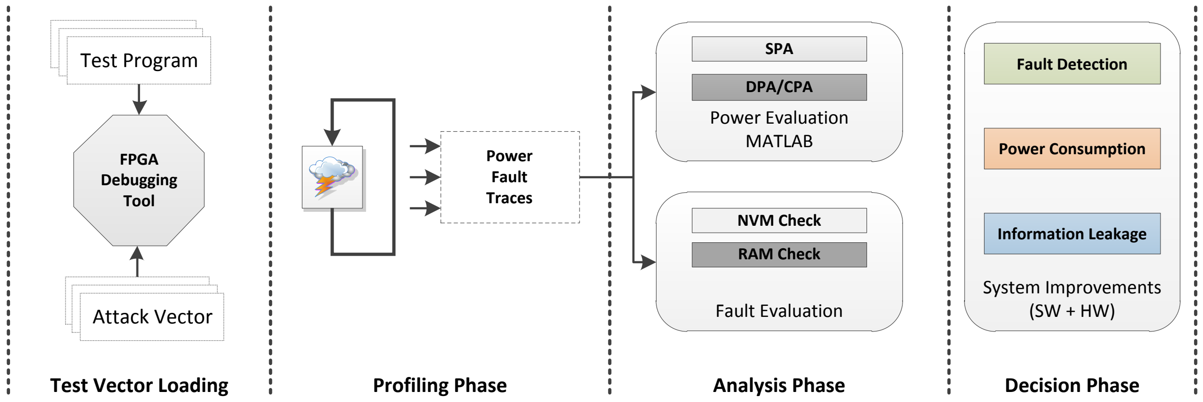


Fig. 6. Hardware accelerated fault and power evaluation flow

Profiling Phase. The profiling phase comprises the main stage of the hardware-accelerated fault emulation process. A test-set containing applications that would be considered a typical load during normal operation is executed on the evaluation platform. During this execution faults are injected using a fault injection controller and saboteurs routed during the placement phase. The fault injection controller can be pre-programmed to start injection at specific values of the processor program counter or directly controlled by an additional external processor.

Power profile tracing can be executed in two ways:

— **Run-time profile tracing using a PE unit with dedicated trace memory** : As described in Section 4.2 dedicated trace memory has to be provided to store intermediate power profile information. This memory has to be regularly read and saved to a persistent storage module for later analysis. This information can be easily converted to text-files for usage with MATLAB scripts.
— **Power profile tracing using RTL simulation** : This would be the technique of choice if examinations have to done during a very early stage of the design process. The simulation model of the PE unit generates text-files of a simple format that can be easily parsed and analyzed using MATLAB scripts. Encryption/decryption run trace separations have to be ensured using software commands, for example by placement of NOP instructions.

Fault analysis tracing can also be done in two ways. First, the fault injection controller contains a checker interface to transfer data from dedicated checker modules to an external processor. These modules allow to evaluate which injected faults resulted into a direct operational fault. Second, additional analysis support modules can be routed into the design-under-inspection if a completely in-system solution is desired. Such support blocks could be checker or power data pre-processing modules to ease data-interpretation or trigger modules to detect specific system operation states.

Analysis Phase. During the analysis phase recorded data is used to gain information about the power, dependability and security characteristics of the targeted system. The derived power profile trace file of the executed test-set programs is parsed using MATLAB scripts. This information then has to be divided into subtraces to reduce analysis effort.

In case of power analysis evaluations all en- or decryption traces have to be separated to enable DPA. The quality of the differential power analysis is highly dependent of the alignment of the analyzed traces. Therefore, eventual trace timing differences have to be corrected by cross correlation techniques. To analyze the power analysis robustness of a given cryptographical software implementation the stored power profile traces can be processed using MATLAB scripts from the OpenSCA toolkit. This process has been completely automatized and results in attack attempts on all key bytes of the used key.

Fault emulation trace information recorded during execution can also be easily parsed using MATLAB to gain data matrices usable in efficiency and statistical analysis.

Decision Phase. After all analysis tasks have been performed the gained data can be used to derive changes of the targeted architecture. Such changes could include stronger error detection or recovery mechanisms or power optimizations to improve the available power budget. If several different fault attack countermeasures are present it is also possible that their combination results in reduced robustness against power analysis attacks. In this case it is possible that other combinations or weaker mechanisms using different countermeasure techniques lead to a better countermeasure performance.

6.4. Automatized Fault Emulation

For real-world applications in the smart-card development and certification sector automatized fault injection campaigns are needed to ensure the coverage of a wide range of possible attack and device aging scenarios. To allow a similar test flexibility compared to slow fault simulations, modular control and fault injection elements as well as a high level fault model representation are needed.

6.4.1. Modular fault injection control blocks. While simple small saboteur elements enable to generate faults at nearly every possible position inside the design, for exhaustive countermeasure testing it is also necessary to define points of time when an attack should be emulated. Such an attack time can be derived in the following ways:

— Program counter (PC): The PC defines the current stage of the running program. By listening to this register an attack at a certain point of the program can be started.
— Clock: As communication programs can endure a significant amount of time, it is more reasonable to use a clock signal counter starting at a pre-defined point like a certain PC value.
— Addresses: If certain memory blocks have to be changed during an attack sequence it is necessary to examine system internal address and data buses. Access to the memory point-of-interest can be manipulated during transmission from or to the processor pipeline or cache.

To enable easy augmentation of a system-under-test so called trigger modules are introduced that can be placed to a certain point of interest. As shown earlier most attack scenarios can be accomplished by listening to the PC register, clock signal and address buses. The needed trigger functionality can be further reduced to two types of trigger modules: single-bit and multi-bit triggers. These trigger modules only output a binary signal to show if a given condition has been reached and therefore they can be combined by logic functions to create different fault activation conditions. All trigger modules are connected to the fault injection controller by shared buses to enable power-on configuration.

Single-bit trigger module. A single-bit trigger module contains a very simple counter structure and a configurable detection circuit. If the input signal changes a pre-defined amount of times a output signal is raised to trigger further processing of other trigger modules or the fault injection controller.

Multi-bit trigger module. Similar to the single-bit trigger module this larger variant contains a counter structure and additional circuits to mask and compare the input value to a given target value. If necessary additional conditions can be compared, for example the bus access status or specific access rights.

6.4.2. High level fault model representation. The fault emulation system will also be provided to another company section or customer to test pre-production software on the given pre-silicon hardware design. As such a customer usually must not have internal information about the RTL-description fault model representations have to be provided. Such representations could be very specific fault patterns to emulate an attack shown in Section 7 or more general representations to simulation intentional memory changes.

Using control elements as presented in Subsection 6.4.1 the complete memory space of a given target application can be manipulated during runtime. This approach enables software developers with no access to the RTL-description to do detailed fault attack campaigns as depicted in Figure 7. These are necessary during certification phases or early evaluations of new secure operating systems.

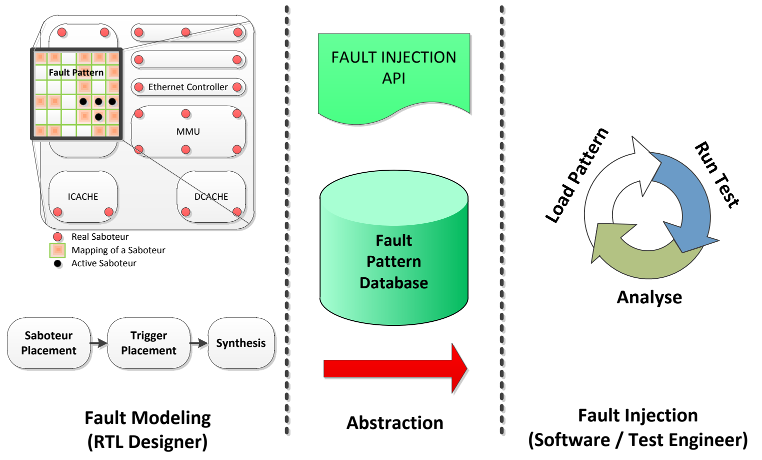


Fig. 7. High-Level Fault Pattern Approach

7. EXPERIMENTAL RESULTS

To prove the usability of our approach we chose a widely used hardware platform and software implementation of the AES cryptography algorithm. For our hardware platform we selected an open source implementation of the SPARC v8 architecture developed by Aeroflex Gaisler [Aeroflex Gaisler 2010]. This processor has been synthesized using Xilinx ISE software and tested on the ML507 evaluation board also from Xilinx.

Our operating system tests have been executed using the SnapGear Linux distribution provided by Aeroflex Gaisler. Our analysis platform consists of MathWorks Matlab 2009b on an six-core 3.2 GHz AMD Phenom-II machine using eight giga-bytes of RAM.

The characterization process has been done using gate-level and power simulations using Synopsys PrimeTime provided by our industrial partner. This process resulted into a data-aware power macro model containing 90 coefficients. For our selected 90nm production process library we achieved accuracies of lower than four percent average and lower than 15 percent RMSE cycle accurate estimation errors. In Figure 8 a comparison of gate-level measurements with the results of power emulating the dhrystone benchmark is shown.

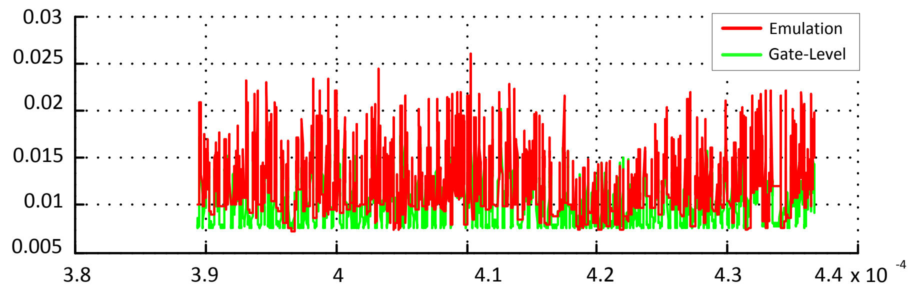


Fig. 8. Dhrystone Benchmark - Comparison Gate-Level to Power Emulation

In Table I a performance overview is shown comparing the achievable injection speed of our approach to previously published systems. This evaluation concerns the pure injection of fault patterns into a given processor as shown in [Grinschgl et al. 2011b].

Table I. Fault injection performance compared to previous approaches

Approach	Clock cycle	Inj. Faults	Time	Inj. Speed	Nor. Inj. Speed
	[ns]		[s]	1/[s]	1/[s]
[Civera et al. 2002]	50	100k	83	1205	603
[Lopez-Ongil et al. 2007] AE	16.8	129.75M	698	185888	276619
[Lopez-Ongil et al. 2007] PR	18.97	10k	1014	9.86	13
[Kafka 2008] DPR	10	-	-	12987	32468
[Kafka 2008] PRR	10	-	-	414.94	1037
This work	25	100M	46.76	2.17M	2.17M

The comparison is valid as these approaches only concern the injection of faults without any further processing of the resulting operating errors. Other solutions use autonomous emulation (AE), partial reconfiguration (PR), partial runtime reconfiguration (PRR) and direct runtime reconfiguration (DPR).

7.1. Dependability Evaluation

Fault injection is already widely used for the emulation of faults to test the efficiency of fault detection and recovery approaches. Existing solutions often do not consider the impact of such dependability improving hardware extensions on the system's power consumption profile. In power constraint systems high power consumption peaks could lead to dangerous supply voltage drops that endanger operating reliability. To present the advantages of a multi-disciplinary approach our LEON3-target system has been augmented with three different parity-based error-detection schemes. These error-detection mechanisms (2, 4 and 5-bit parity codes) have been added to the memory management unit (MMU) to protect the instruction- and data-cache databus. These parity codes are checked inside the integer-unit to ensure integrity of the read data. To implement our fault injection experiments we augmented this buses with a bus-saboteur transparent under normal conditions. This saboteur allows the injection of multiple faults to every data line routed through it. Fault patterns have been selected randomly and injected into a looping calculation. Data errors are detected if the parity-code generated inside the MMU is different to the parity-code generated inside the IU.

The results of such a fault injection campaign are shown in Table II and match our expectations concerning their operation principles. The 2-bit parity implementation performs significantly worse than stronger implementations. For the next evaluation step we could therefore select the 5-bit implementation for our power consumption explorations.

Table II. Fault injection evaluation of parity-based error detection scheme - Data-Cache-Bus

		Detected Errors (DE)	DE [%]
Injected Faults	100000	-	-
2-bit Parity Scheme	-	74783	74.78
4-bit Parity Scheme	-	93705	93.7
5-bit Parity Scheme	-	96873	96.87

Another application for automated fault emulation campaigns would be timing investigations to test given designs for their robustness against device degradation effects. Signal delays caused by negative-bias temperature instability (NBTI) can be emulated by high-clocked delay elements inside the signal path [Krieg et al. 2011c].

Conclusion. Common error-detection schemes using parity-based hamming codes have been thoroughly examined for their error-detection efficiency. The proposed platform can also be used for the concurrent examination of fault resistance, power consumption and delay behavior of a given RTL-description.

7.2. Security Evaluation A (AES Software Implementation)

The nature of faults occurring during an attack scenario is very different to one of those resulting of natural causes. The reason is that these faults happen randomly while an adversary intentionally injects such faults in order to drive the attacked system into an unintended state. To prove the feasibility of our approach we chose a published attack on the AES symmetric cryptographic algorithm shown in [Schmidt et al. 2009]. The authors presented a light-attack on the AES-SBOX residing inside an programmable flash memory device. This attack is emulated by the placement of saboteurs on the memory data-bus of our target architecture. These saboteurs are always activated when the memory region containing the AES-SBOX is read by our test-program. This way all SBOX accesses return 0xFF as a result and are therefore equivalent to accesses to an device erased using ultra-violet light. The resulting wrong intermediate results can then be used to calculate the complete key used for this AES-encryption. In Table III the input key of the AES encryption (a), the corrupted AES output (b) and the calculated key after the fault injection process (c) are shown.

Table III. AES-SBOX fault emulation results

1F	1B	17	13
1E	1A	16	12
1D	19	15	11
1C	18	14	10

(a) Original Key

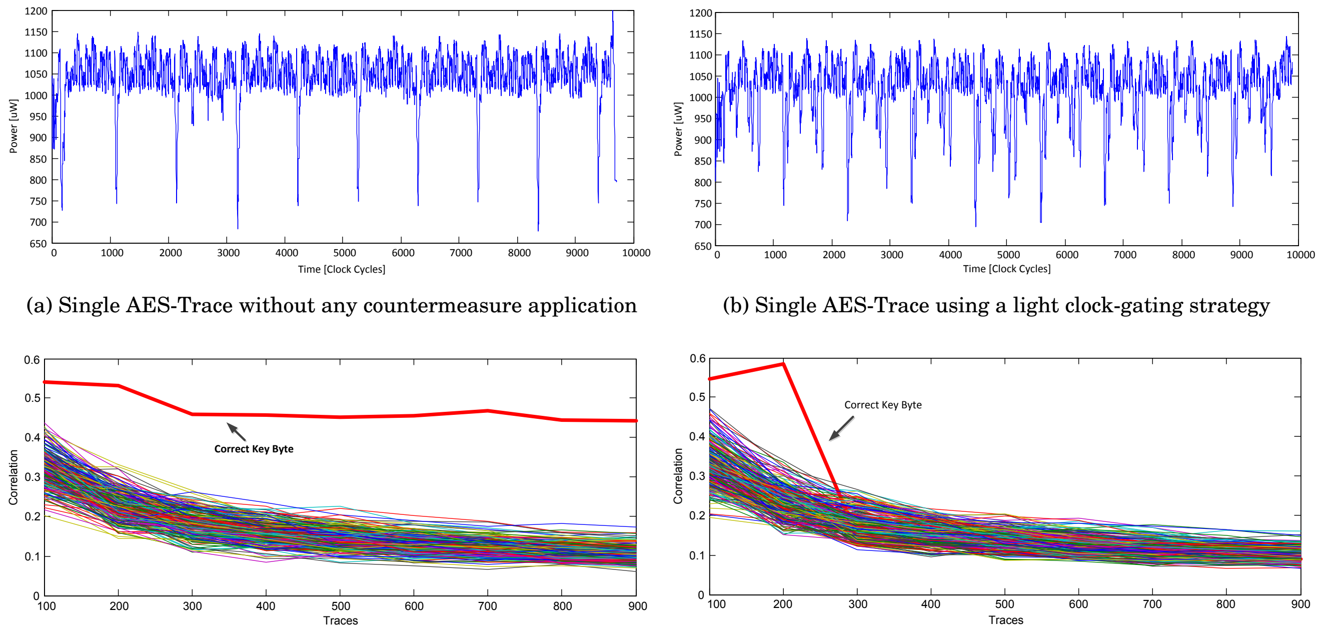
e0	1b	f7	08
e1	1a	f7	08
e2	19	f7	08
31	84	e9	1c

(b) Corrupted Output

1F	1B	17	13
1E	1A	16	12
1D	19	15	11
1C	18	14	10

(c) Calculated Key

By using a data-aware power estimation macro model different power analysis types are also enabled. Figures 9a and 9b show cropped power traces containing one AES encryption run with and without influence of a simple clock gating based countermeasure as described in [Krieg et al. 2011b] To reduce power profile artifacts caused by task switching done by the operating system kernel, this figures have been recorded during RTL simulation. For emulation a 32000 sample FIFO has been implemented to allow the storage of a single trace in hardware for further analysis. The evaluation software has to read out this data after each tracing run to store large amounts of investigation data. A performance comparison of all applied techniques, as determined in [Krieg et al. 2011a] , is shown in Table IV. Figures 9c and 9d give an overview how successful an DPA attack on the unmodified and augmented architecture would be. Clock gating has been described in literature as a measure to increase profile noise and to decrease inter-profile correlation [Benini et al. 2003]. In this case the chosen countermeasure would not have a significant impact on power consumption but its effectiveness against power analysis would have to be improved.



(a) Single AES-Trace without any countermeasure application (b) Single AES-Trace using a light clock-gating strategy

(c) DPA-Attack on unmodified architecture (d) DPA-Attack on architecture containing light countermeasure

Fig. 9. Tracing and Differential Power Analysis (DPA)

Table IV. Tracing performance: RTL Simulation vs. FPGA Emulation (taken from [Krieg et al. 2011a])

	Traces	Time [sec]	Speedup
Direct RTL Simulation	100	3850.05	-
FPGA Emulation (w/o OS-overhead)	100	0.0258	149226
FPGA Emulation (w/ OS-overhead)	100	43.13	89.27

Conclusion. Therefore additional hardware used for improved circuit reliability can also be used to improve fault attack robustness. Simple manipulations of the clock signal provide only weak protection against power analysis attacks.

7.3. Security Evaluation B (AES Hardware Implementation)

In [Takahashi et al. 2007] Takahashi et al. describe a differential fault attack on the key schedule of an AES implementation. The authors claim that using this method it is possible to extract a complete 128-bit secret key using only seven pairs of correct and faulty cipher-texts without using brute-force search.

A reduced variant proposed in [Takahashi et al. 2007] only needs two pairs of correct and faulty cipher-texts to extract 48-bit of the AES key. For this work we will show how to evaluate a given AES hardware block for its robustness against such a reduced fault attack.

Target of this approach is the key schedule mechanism, which can be manipulated at three different points as shown in 10. The first point to be attacked is the 3rd 32-bit column of the 9th round key. Another possibility would be to attack 2nd 32-bit column and the 1st 32-bit column of the 9th round key. Both scenarios result in corrupted cipher-texts that can be used for the calculation of the final AES key. The following process is defined by seven rules as proposed in [Takahashi et al. 2007] and does not require further attacks on the hardware.

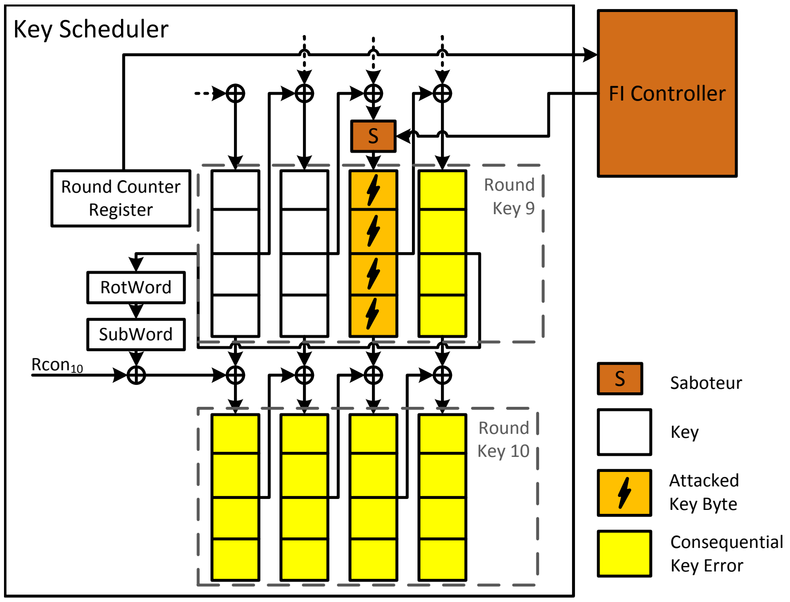


Fig. 10. Structure of the AES Core fault attack mechanism

For this case study we chose to attack the 3rd column of the AES calculation. For our target we chose an open implementation of the AES algorithm [OpenCores 2011]. As seen in Figure 10 placed a saboteur into the 3rd column key generation mechanism. This saboteur is controlled by the fault injection controller(FIC). The FIC has also been connected to the internal AES round counter register. This way the fault injection process can be automatically activated when this register reaches the 9th round. During the fault attack random bits of the targeted round key column are flipped.

Table V. Two plain-texts and its corresponding cipher-text pairs

(a) Plain-text 1

00	44	88	cc
11	55	99	dd
22	66	aa	ee
33	77	bb	ff

(b) Cipher-text 1

69	6a	d8	70
c4	7b	cd	b4
e0	04	b7	c5
d8	30	80	5a

(c) Faulty cipher-text 1

29	2a	71	fc
6b	c0	63	1b
27	d1	1a	c2
06	ec	fc	92

(d) Plain-text 2

00	04	08	0c
01	05	09	0d
02	06	0a	0e
03	07	0b	0f

(e) Cipher-text 2

0a	41	f1	c6
94	6e	c3	53
0b	f0	94	ea
b5	45	58	5a

(f) Faulty cipher-text 2

4a	01	67	92
3b	7c	d4	fc
e6	4f	39	ed
44	99	24	92

Table V shows plain-texts and their corresponding cipher-text pairs with and without manipulation. After the fault attack using methods described in [Takahashi et al. 2007] the following key bytes ($K_{0,3}$, $K_{2,0}$, $K_{2,1}$, $K_{2,3}$, $K_{3,0}$, $K_{3,3}$) have been calculated as shown in Table VI.

Table VI. Fault attack and key calculation results

00	04	08	0c
01	05	09	0d
02	06	0a	0e
03	07	0b	0f

(a) Key

54	f0	b0[a]	1e[b]
99	85	91[a]	2e[b]
32	57	47[a]	3d[b]
d1	68	3c[a]	ee[b]

(b) Corrupted 9th Round Key

54	f0	10	be[c]
99	85	93	2c
32[c]	57[c]	ed	97[c]
d1[c]	68	9c	4e[c]

(c) Correct 9th Round Key

[a]Attacked Corrupted Key Bytes
[b]Consequentially Corrupted Key Bytes
[c]Calculated key bytes after fault attack

Conclusion. In this chapter the application of a direct manipulation of a cryptographic hardware module has been shown. The saboteur-based approach allows the testing of general crypto-implementations as long as the RTL-level hardware description is available to the test engineer. The general applicability our automatized fault injection approach has been proven using an open available AES-module and a non-implementation specific hardware attack described in literature.

8. CONCLUSION

This paper presents a novel FPGA-based platform and methodology for reliability and security evaluations of smart-card and general purpose processor implementations. Through the combination of fault injection and power emulation techniques efficient hardware/software co-simulation of cryptographic software and hardware is enabled. The instant evaluation of power consumption and fault resistance behavior allows for the design of novel architectures under the consideration of power and security constraints. The effectiveness of our proposed hardware accelerated fault emulation platform has been shown using software and hardware implementations of the symmetrical AES algorithm. Novel modular trigger and fault injection mechanisms allow for the efficient execution of long running injection campaigns.

The efficient modeling and implementation of fault attacks will be play a vital role during the design of future secure architectures and certification for high security standards.

Our future work includes the usage of the proposed platform to examine new smart-card architectures for possible weaknesses. It also will be used to identify effective modifications for general purpose architectures to improve their reliability and fault attack robustness. The combined approach will also allow for a detailed trade-off analysis of different error detection and recovery mechanisms considering both power and security constraints.

ACKNOWLEDGMENTS

The authors would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology, which funded the Power-Modes Project under the FIT-IT contract FFG 825749. We would also like to thank our project partners Infineon Technologies Austria AG and Austria Card GmbH.

REFERENCES

AEROFLEX GAISLER. 2010. LEON3 Processor.

ANTONI, L., LEVEUGLE, R., AND FEHER, M. 2002. Using run-time reconfiguration for fault injection in hardware prototypes. In *Defect and Fault Tolerance in VLSI Systems, 2002. DFT 2002. Proceedings. 17th IEEE International Symposium on*. IEEE, 245–253.

ARLAT, J., AGUERA, M., AMAT, L., CROUZET, Y., FABRE, J., LAPRIE, J., MARTINS, E., AND POWELL, D. 2002. Fault injection for dependability validation: A methodology and some applications. *Software Engineering, IEEE Transactions on 16,* 2, 166–182.

ARLAT, J., AGUERA, M., AMAT, L., CROUZET, Y., FABRE, J.-C., LAPRIE, J.-C., MARTINS, E., AND POWELL, D. 1990. Fault injection for dependability validation: a methodology and some applications. *Software Engineering, IEEE Transactions on 16,* 2, 166 –182.

BACHMANN, C., GENSER, A., STEGER, C., WEISS, R., AND HAID, J. 2010. Automated Power Characterization for Run-Time Power Emulation of SoC Designs. In *DSD 2010*. 587–594.

BAR-EL, H., CHOUKRI, H., NACCACHE, D., TUNSTALL, M., AND WHELAN, C. 2006. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE 94,* 2, 370–382.

BARAZA, J., GRACIA, J., GIL, D., AND GIL, P. 2006. Improvement of fault injection techniques based on VHDL code modification. In *High-Level Design Validation and Test Workshop, 2005. Tenth IEEE International*. IEEE, 19–26.

BARAZA, J. C., GRACIA, J., GIL, D., AND GIL, P. J. 2002. A prototype of a vhdl-based fault injection tool: description and application. *Journal of Systems Architecture 47,* 10, 847 – 867.

BENINI, L., MACII, A., MACII, E., OMERBEGOVIC, E., PRO, F., AND PONCINO, M. 2003. Energy-aware design techniques for differential power analysis protection. In *Proceedings of the 40th annual Design Automation Conference*. ACM, 36–41.

BHATTACHARJEE, A., CONTRERAS, G., AND MARTONOSI, M. 2008. Full-System Chip Multiprocessor Power Evaluations Using FPGA-Based Emulation. In *ISLPED 2008*.

BRIER, E., CLAVIER, C., AND OLIVIER, F. 2004. Correlation power analysis with a leakage model. *CHES 2004*, 135–152.

BUCCI, M., LUZZI, R., MENICHELLI, F., MENICOCCI, R., OLIVIERI, M., AND TRIFILETTI, A. 2007. Testing power-analysis attack susceptibility in register-transfer level designs. *IET 2007 1,* 3, 128–133.

CIVERA, P., MACCHIARULO, L., REBAUDENGO, M., REORDA, M., AND VIOLANTE, M. 2002. Exploiting circuit emulation for fast hardness evaluation. *Nuclear Science, IEEE Transactions on 48,* 6, 2210–2216.

COBURN, J., RAVI, S., AND RAGHUNATHAN, A. 2005. Power emulation: a new paradigm for power estimation. In *DAC 2005*. 700–705.

DAVEAU, J., BLAMPEY, A., GASIOT, G., BULONE, J., AND ROCHE, P. 2009. An industrial fault injection platform for soft-error dependability analysis and hardening of complex system-on-a-chip. In *Reliability Physics Symposium, 2009 IEEE International*. IEEE, 212–220.

DEN HARTOG, J. AND DE VINK, E. 2005. Virtual Analysis and Reduction of Side-Channel Vulnerabilities of Smartcards. In *Formal Aspects in Security and Trust*. Springer, 85–98.

GENSER, A., BACHMANN, C., HAID, J., STEGER, C., AND WEISS, R. 2009. An emulation-based real-time power profiling unit for embedded software. In *SAMOS 2009*. 67–73.

GHODRAT, M., LAHIRI, K., AND RAGHUNATHAN, A. 2007. Accelerating system-on-chip power analysis using hybrid power estimation. In *Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE*. IEEE, 883–886.

GRINSCHGL, J., KRIEG, A., STEGER, C., WEISS, R., BOCK, H., AND HAID, J. 2011a. Automatic saboteur placement for emulation-based multi-bit fault injection. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*. 1 –8.

GRINSCHGL, J., KRIEG, A., STEGER, C., WEISS, R., BOCK, H., AND HAID, J. 2011b. Modular fault injector for multiple fault dependability and security evaluations. In *Digital System Design (DSD), 2011 14th Euromicro Conference on*. IEEE, 550–557.

GUNNEFLO, U., KARLSSON, J., AND TORIN, J. 2002. Evaluation of error detection schemes using fault injection by heavy-ion radiation. In *Fault-Tolerant Computing, 1989. FTCS-19. Digest of Papers., Nineteenth International Symposium on*. IEEE, 340–347.

JENN, E., ARLAT, J., RIMEN, M., OHLSSON, J., AND KARLSSON, J. 1994. Fault injection into vhdl models: the mefisto tool. In *Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers., Twenty-Fourth International Symposium on*. 66 –75.

JENN, E., ARLAT, J., RIMÉN, M., OHLSSON, J., AND KARLSSON, J. 2002. Fault injection into VHDL models: the MEFISTO tool. In *Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers., Twenty-Fourth International Symposium on*. IEEE, 66–75.

KAFKA, L. 2008. Analysis of Applicability of Partial Runtime Reconfiguration in Fault Emulator in Xilinx FPGAs. In *DDECS '08: Proceedings of the 2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*. IEEE Computer Society, Washington, DC, USA, 1–4.

KOCHER, P., JAFFE, J., AND JUN, B. 1999. Differential Power Analysis. In *CRYPTO 99*. Springer.

KRIEG, A., BACHMANN, C., GRINSCHGL, J., STEGER, C., AND WEISS, R. 2011. Accelerating Early Design Phase Differential Power Analysis Using Power Emulation Techniques. In *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*. 81 –86.

KRIEG, A., GRINSCHGL, J., STEGER, C., WEISS, R., BOCK, H., AND HAID, J. 2011. Run-time FPGA health monitoring using power emulation techniques. In *Circuits and Systems (MWSCAS), 2011 IEEE 54th International Midwest Symposium on*. IEEE, 1–4.

KRIEG, A., GRINSCHGL, J., STEGER, C., WEISS, R., AND HAID, J. 2011. A side channel attack countermeasure using system-on-chip power profile scrambling. In *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*. IEEE, 222–227.

LEVEUGLE, R. 2002. Fault injection in VHDL descriptions and emulation. In *Defect and Fault Tolerance in VLSI Systems, 2000. Proceedings. IEEE International Symposium on*. IEEE, 414–419.

LEVEUGLE, R. 2007. Early analysis of fault-based attack effects in secure circuits. *IEEE Transactions on Computers*, 1431–1434.

LEVEUGLE, R. AND HADJIAT, K. 2003. Multi-level fault injections in vhdl descriptions: Alternative approaches and experiments. *Journal of Electronic Testing 19,* 5, 559–575.

LI, H., MARKETTOS, A., AND MOORE, S. 2005. Security evaluation against electromagnetic analysis at design time. In *High-Level Design Validation and Test Workshop, 2005. Tenth IEEE International*. 211 – 218.

LOPEZ-ONGIL, C., GARCIA-VALDERAS, M., PORTELA-GARCIA, M., AND ENTRENA, L. 2007. Autonomous fault emulation: A new FPGA-based acceleration system for hardness evaluation. *IEEE Transactions on Nuclear Science 54,* 1, 252.

MANGARD, S., OSWALD, E., AND POPP, T. 2007. *Power analysis attacks: Revealing the secrets of smart cards*. Vol. 31. Springer-Verlag New York Inc.

OPENCORES. 2011. Aes_crypto_core.

POHL, C., PAIZ, C., AND PORRMANN, M. 2009. vMAGIC - Automatic Code Generation for VHDL. *International Journal of Reconfigurable Computing 2009*.

RAVI, S., RAGHUNATHAN, A., KOCHER, P., AND HATTANGADY, S. 2004. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems (TECS) 3,* 3, 461–491.

REGAZZONI, F., BADEL, S., EISENBARTH, T., GROSSSCHAEDL, J., POSCHMANN, A., TOPRAK, Z., MACCHETTI, M., POZZI, L., PAAR, C., LEBLEBICI, Y., AND IENNE, P. 2007. A Simulation-Based Methodology for Evaluating the DPA-Resistance of Cryptographic Functional Units with Application to CMOS and MCML Technologies. In *IC-SAMOS 2007*. 209 –214.

REGAZZONI, F., CEVRERO, A., STANDAERT, F.-X., BADEL, S., KLUTER, T., BRISK, P., LEBLEBICI, Y., AND IENNE, P. 2009. A Design Flow and Evaluation Framework for DPA-Resistant Instruction Set Extensions. In *CHES 2009*. Springer, 205–219.

ROCHE, T., LOMNÉ, V., AND KHALFALLAH, K. 2011. Combined Fault and Side-Channel Attack on Protected Implementations of AES. *Smart Card Research and Advanced Applications*, 65–83.

ROTHBART, K., NEFFE, U., STEGER, C., WEISS, R., RIEGER, E., AND MUEHLBERGER, A. 2004. High level fault injection for attack simulation in smart cards.

SCHAUMONT, P. AND TIRI, K. 2007. Masking and dual-rail logic dont add up. *Cryptographic Hardware and Embedded Systems-CHES 2007*, 95–106.

SCHMIDT, J., HUTTER, M., AND PLOS, T. 2009. Optical fault attacks on AES: A threat in violet. In *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 13–22.

SEGALL, Z., VRSALOVIC, D., SIEWIOREK, D., YASKIN, D., KOWNACKI, J., BARTON, J., DANCEY, R., ROBINSON, A., AND LIN, T. 2002. Fiat-fault injection based automated testing environment. In *Fault-Tolerant Computing, 1988. FTCS-18, Digest of Papers., Eighteenth International Symposium on*. IEEE, 102–107.

SHUMOV, D. AND MONTGOMERY, P. L. 2010. Side Channel Leakage Profiling in Software. In *COSADE 2010*.

TAKAHASHI, J., FUKUNAGA, T., AND YAMAKOSHI, K. 2007. DFA Mechanism on the AES Key Schedule. In *Fault Diagnosis and Tolerance in Cryptography, 2007. FDTC 2007. Workshop on*. 62 –74.

THUILLET, C., ANDOUARD, P., AND LY, O. 2009. A Smart Card Power Analysis Simulator. *CSE 2009 2*, 847–852.

TSAI, T., HSUEH, M., ZHAO, H., KALBARCZYK, Z., AND IYER, R. 2002. Stress-based and path-based fault injection. *Computers, IEEE Transactions on 48,* 11, 1183–1201.

VALDERAS, M., GARCIA, M., CARDENAL, R., ONGIL, L., AND ENTRENA, L. 2007. Advanced Simulation and Emulation Techniques for Fault Injection. In *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*. IEEE, 3339–3344.

ZHENG, H., FAN, L., AND YUE, S. 2008. FITVS: A FPGA-Based Emulation Tool For High-Efficiency Hardness Evaluation. In *Parallel and Distributed Processing with Applications, 2008. ISPA'08. International Symposium on*. IEEE, 525–531.

Characterization and Handling of Low-Cost Micro-Architectural Signatures in MPSoCs

Armin Krieg, Johannes Grinschgl,
Christian Steger and Reinhold Weiss
Institute for Technical Informatics
Graz University of Technology, Austria
{armin.krieg, johannes.grinschgl,
steger, rweiss}@tugraz.at

Andreas Genser, Holger Bock,
Josef Haid
Design Center Graz
Infineon Technologies Austria AG
{andreas.genser, holger.bock,
josef.haid}@infineon.com

***Abstract*—In recent years the wide spread introduction of small embedded systems into every corner of everyday life lead to the strong need for highly reliable and secure computing machines. These machines now affect the safety of humans as well as the security of personal data and consequently money transactions. To ensure the integrity of these systems' operating state, several fault detection mechanisms have been developed to safely correct or stop unforeseen execution behavior. Because of the rise of battery or even field-supplied systems these mechanisms often heavily decrease available power budgets or lead to significantly increased production costs.**

Therefore, this paper introduces novel micro-architectural execution signature characterization and handling techniques for system-on-chip designs providing power estimation hardware. Existing power sensor infrastructure is reused to enable efficient system-state monitoring using micro-architectural hashes to cover a wide range of implemented system functionality. Reduced hashing implementations are characterized for their fault detection efficiency. This hardware-based approach provides a completely transparent solution to counteract faults resulting from emerging wear-out defects or intentional attacks on the execution integrity.

I. Introduction

Research on sophisticated fault detection mechanisms using on-line control flow monitoring has been ongoing for decades. The main driver behind this research has been computing machines for high-availability and security applications. Such threats to system integrity come from two major contributors. First, continuing semiconductor process integration leads to finer transistor structures and therefore to a higher probability of transistor defects. Second, criminal subjects try to gain secret information from secure devices by disturbing normal operation using sophisticated fault attacks.

In case of systems for highly dependable applications it is important to detect control flow changes because of faults resulting of environmental influences or manufacturing defects. These faults are of a random nature and their detection will usually lead to the activation of countermeasures to prevent erroneous actions that could result in loss of money or life. In case of high-security applications like smart-cards, fault detection is essential to identify attacks by an external adversary. The primary function is to prevent the loss of information and to hide the fact that the attack was successful. To guarantee a high fault detection coverage usually modular redundancy techniques are used. By duplicating or tripling processing units, even small control flow changes or data manipulations can be detected or corrected in case of a triple configuration. The disadvantage of such a fault detection approach are high implementation and manufacturing costs. In the high volume smart-card market such additional production costs are prohibitive.

Therefore, a strong need for control flow manipulation detection mechanisms providing a similar detection coverage as modular duplication while relying on a smaller resource footprint rose in the recent past. To gain an area effective implementation of a control flow observation mechanism, existing infrastructure has to be reused and simplified hashing implementations have to be found. Accurate on-line power estimation is also dependent on a view of the current control state of the system and therefore a hybrid monitoring structure is proposed. Hashing mechanisms are simplified by round and structure reduction and have to be characterized for the impact of these simplifications on fault detection efficiency.

The main contributions of this work are:

- Introduction of a novel control-signal signature hardware characterization strategy and hybrid power and fault monitoring infrastructure
- Integration of the proposed methodology into a state-of-the-art power profiling and functional emulation flow
- Case study using an augmented version of an open available system-on-chip implementation

This paper is structured as follows. Section II is giving a short introduction into control-state based hardware power estimation. In Section III common mechanisms for micro-architectural signatures are roughly described and the state-of-the-art concerning these signatures for general purpose architectures is briefly reviewed. Followed by Section IV introducing a novel transparent micro-architectural signature generation methodology. The efficiency of the proposed approach is experimentally investigated in Section V using a common general-purpose-processor system. Finally, our results are concluded and some details about our future work are given in Section VI.

II. Control-State based Hardware Power Estimation

Hardware power estimation has emerged as a rich technique in order to obtain power consumption information in a cycle-accurate manner. Having on-chip power information available to drive power management algorithms is becoming increasingly important for power-constrained embedded systems [1]. In particular in field-supplied systems it is mandatory to have power monitoring mechanisms available in order to track the power consumption of the system and to detect and to prevent the occurrence of harmful events (i.e., power peaks, supply voltage drops).

Additional hardware blocks enable the observation of internal system states, which are then fed to a power model implemented in hardware. Power estimates delivered by the power model can then be exploited by power management mechanisms. The principle of control-state inspection for system monitoring is similar to a signature-based fault detection approach as proposed in this work. Therefore a fusion of both evaluation techniques (power and fault) is possible to optimize resource efficiency. This step includes the relying on a joined activity sensor infrastructure and characterization process.

A. Power Model

The set of internal system states $\mathbf{x}$, each of which denoted by x_i and model coefficients c_i constitute the power model as depicted in (1)

$$P(\mathbf{x}) = \sum_{i=0}^{n-1} c_i x_i + e. \tag{1}$$

The power model's result gives power estimates $\hat{P}(\mathbf{x})$ corresponding to the linear combination of internal system states x_i and the model coefficients c_i. The difference of the power estimate $\hat{P}(\mathbf{x})$ to the real power consumption $P(\mathbf{x})$ is denoted by error e. The selection of a representative set of internal system states as well as the determination of the power model coefficients is performed during a power characterization process [2]. This process includes the execution of benchmarking programs using a gate-level description of the design to create an accurate power consumption abstraction.

B. Power Estimation Architecture

The power estimation architecture as depicted in Figure 1 consists of a number of power sensors that observe and map internal system states x_i to power model coefficients c_i. Power estimates delivered by the power sensors are summed up in the power accumulation unit, which finally provides the power estimate $\hat{P}(\mathbf{x})$.

The accuracy of the power estimate $\hat{P}(\mathbf{x})$ gathered from the power model depends on the number and quality of the observed system states. The approach is flexible in a way that the system state granularity at which the system is observed can be tailored to accuracy requirements of the power estimates. This is also advantageous for the coverage of control-flow based fault detection mechanisms by increasing the number of instruction-decode stage signals that are considered.

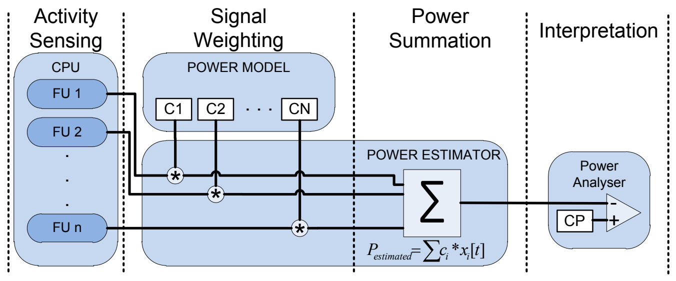


Fig. 1. Control signal based power estimation principle (adapted from [3])

III. Related Work

State-of-the-art research in the field of fault detection mechanisms using on-line control flow monitoring can be split into three main groups.

A. Software-based signature mechanisms

Purely software-based approaches mostly rely on the automatic or semi-automatic generation of code signatures to detect program flow changes caused by tampering or environmental influences. These signatures are directly embedded into the executed binaries. For the signature checking procedure itself processor internal resources have to be reused resulting in a high impact on execution performance.

Pure software solutions have been proposed using extensive redundancy as shown in [4] or by applying additional state checking to catch unforeseen execution control behavior as presented in [5] and [6]. An approach solely based on software signatures generated during compile time is proposed in [7]. The main disadvantage of these software-only approaches is their significant impact on execution performance. Furthermore, software checks could be manipulated by an adversary if such techniques are used in security critical applications.

B. Hardware-based signature mechanisms

Most commonly used hardware blocks for integrity checking are hardware monitors. Depending on the monitored unit these monitors can be implemented very efficiently. This advantage is counteracted if large memories are integrated to store precomputed signatures. Another problem concerning the state-of-the-art in this field is the selection of the monitored system region. Pipeline-only approaches may be insufficient for system-on-chips and the use of multiple monitors increases system complexity and decreases area efficiency.

To reduce the impact of the monitoring process on the operating performance and to gain direct access to hardware resources, control flow-monitoring approaches using dedicated monitoring hardware have been introduced. Such dedicated monitoring hardware could be watchdog-type modules as shown in [8] and [9] or smaller specialized monitoring circuits covering only selected parts of the system as presented in [10], [11] and [12]. Other approaches rely on modification of the processor pipeline to enable the observation of the control flow [13]. While being very effective in processor-only applications, it is often not sufficient for system-on-chips including a wide range of different processing units.

C. Hardware-Software co-design solutions

To ensure a wide detection coverage several co-design based approaches have been published. In this case the possibility to adapt both software support and hardware structure is used to reduce memory overhead and performance penalties. Such systems are generally based on application-specific instruction processors (ASIP) as published in several recent papers [14]–[16]. By design it is only possible to apply such techniques if the target architecture is either highly adaptable or the design process is at a very early stage enabling instruction set changes.

Another possibility would be to augment existing hardware to generate representative values to track control changes inside the system-under-test. Such reusable hardware blocks could be scan-out-chains as presented in [17]. This approach while having only a small impact on the complexity of the targeted system relies on periodic tests and therefore is not a real on-line testing solution. Another possibility would be fingerprinting techniques generating hash-values from the complete architectural state as published in [18]. The proposed fingerprinting approach while providing very accurate fault detection mechanisms leads to high demands on circuit bandwidth.

D. Contributions

Our proposed approach can be categorized into group number three supporting system-internal fault detection mechanisms as well as software controlled solutions. Compared to existing work our implementation does not rely on large architecture augmentations as necessary in group two and does not imply performance degradations of group one. It can be applied to any hardware containing state-dependent power estimation (PE) units without relying on large hash generation circuitry. This co-existence with PE hardware should also result in a very limited amount of additional needed area resources. Furthermore, our approach not only covers a microprocessor's pipeline but large parts of the entire system-on-chip to guarantee a wide fault detection coverage. The general applicability only relies on the availability of the RTL system description.

IV. Signature Mechanism for Resource Constraint Systems

In resource constraint systems like smart-cards huge memory, runtime or hardware overhead is prohibitive. Therefore, large control-flow graphs, embedded code signatures or extensive monitoring hardware cannot be employed to ensure correct system behavior. On the other hand the continuing integration of additional components into system-on-chip devices also adds a wide range of additional targets for attacks and points where degradation could lead to system failure.

Therefore, we propose a complete signature element selection and hardware description augmentation methodology using passive signature generation and comparison hardware. The goal of this approach is to provide high control flow security compared with low performance and area overhead. An overview of this proposed characterization and generation methodology is depicted in Figure 2.

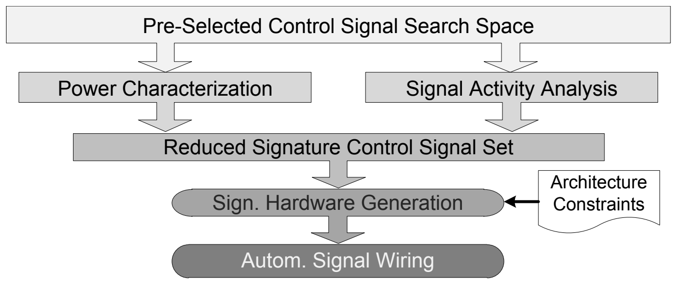

Pre-Selected Control Signal Search Space
Power Characterization
Signal Activity Analysis
Reduced Signature Control Signal Set
Sign. Hardware Generation
Architecture Constraints
Autom. Signal Wiring

Fig. 2. Signature architecture augmentation methodology

A. Control signal selection

This part of the signature implementation process is crucial to keep hardware effort low while enabling reliable and efficient system monitoring. The following two main criteria are important when selecting control signals for monitoring purposes:

- **Relevance**: Significant control signals should show strong activity in a large selection of standard applications. This requirement is also of importance considering control signal based power estimation to gain good estimation accuracy. Therefore, a power model characterization process will be employed to determine good candidates for hardware block signatures. A good example for such a characterization process is given in [2].
- **Determinism**: Signal activity must be directly dependent on the executed instruction sequence. In this case similar criteria are important for fault injection campaigns to simulate device degradation effects caused by high transistor activity in systems manufactured using very fine semiconductor technologies. Therefore RTL-simulation based activity analysis techniques will be used to determine good execution-signal correlation. The principle analysis techniques are similar to those shown in [19].

B. Signature type selection

Selection of a proper signature type is crucial to keep hardware overhead low without increasing the risk of collisions over an unacceptable level. Most hash signature algorithms, while being very robust against collisions and providing good coefficient mixing, need a significant amount of area in their hardware implementations. In case of system-internal execution signatures collisions are of lower concern than constraints on coefficient mixing. Therefore, the design can rely on hash algorithms with good avalanche behavior, meaning large output changes at small input changes, while being less optimal in means of collision resistance. Another important point when selecting a good hardware signature algorithm is calculation latency. Arora et. al. solved the problem of limiting maximum calculation latencies by adding hardware and limiting the maximum block size to 512 [20]. Still their MD4 and MD5 implementations had a significant impact on operating performance because the pipeline stalled until the hash generation is finished. An overview over small and fast hash functions for hash table lookup including MD4 is shown in [21]. To gain a lightweight memory-free hardware implementation only functions are considered that do not rely on coefficient tables. This excludes the simple SBOX-function that relies on large data tables while being computationally simple. All chosen signature implementations have been configured to result into a compression factor of 2:1. Meaning that the resulting hash value is only half as wide as the input vector. All hash implementations have been round reduced and simplified to retrieve a highly efficient hardware integration. Specifically a single-round implementation has been targeted to gain fast detection of execution variations.

For our exemplary implementations the following algorithms have been chosen:

CRC hashing: Well documented and automatically generatable for a wide variety of bit-widths are cyclic-redundancy-check (CRC) code generator-modules. For the VHDL code generation all even coefficients have been chosen to retrieve a constant hardware structure [22].

One-at-a-time hash function: The one-at-a-time hash algorithm shown by Jenkins et al. in [21] has been chosen because of its simple structure and because it can be simply implemented in hardware without additional execution latency. Compared to CRC one-at-a-time provides very good avalanche behavior while more hardware effort is needed.

MD5 secure hash: The well known MD5 hash algorithm has been reduced to retrieve a small and fast implementation. This reduction is achieved by only using one simplified round of MD5.

SHA secure hash: The basic structure of this signature implementation is very similar to the MD5-based one. Therefore, the same restrictions and characteristics apply for this low-cost hash function.

C. Signature segmentation

The possibility of fault location extraction from an execution signature is of vital importance for further fault analysis processes. Therefore, it has to be segmented depending on the available systems elements. After control signals have been grouped depending on their original location, individual signatures are generated and concatenated as shown in Figure 3. Such groups would be individual processor-cores in multi-core systems or system-on-chip elements like network controllers or caching control blocks. The selection of group members is done using a component-aware power characterization process to identify deterministic and relevant module control signals.

The efficiency of the segmented signature approach depends on the scalability of the hashing implementation and the granularity of the signature segments.

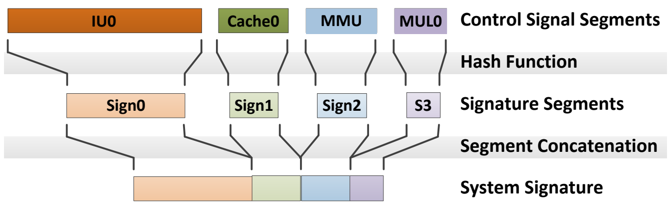


Fig. 3. Signature segmentation approach

D. Signature-based execution architecture

Depending on the available memory resources on-line or off-line signature checking techniques can be used.

Off-line techniques would include a software characterization process to generate execution signatures of targeted code regions. In this case a set of the calculated hashes has to be saved to the target device to be compared in hardware during regular operation. The advantage of this approach is the lack of multiple signature generator circuits as the reference is produced using a system emulator provided to the software developer. On the other hand large signature memories are needed to store the control flow history of the complete code execution.

On-line approaches include both, the reference and control signature generation, inside the target hardware. This can be implemented in a completely memory-free manner using a reference signature generation step during the fetch stage of the pipeline. Another possibility would be to use a system emulator for instruction characterization and in-system comparison at the end of the processor's pipeline. The memory-free approach will rely on a complex signature generation circuit predicting the future of the control signal state at a later stage. Instruction-only characterization constitutes a good compromise between needed circuit and memory requirements.

For our experimental evaluations an off-line approach using complete pre-calculated signatures has been chosen to demonstrate the feasibility of our technique. These signatures have been derived using a golden model run of the selected benchmark programs. The selected architecture is depicted in Figure 4.

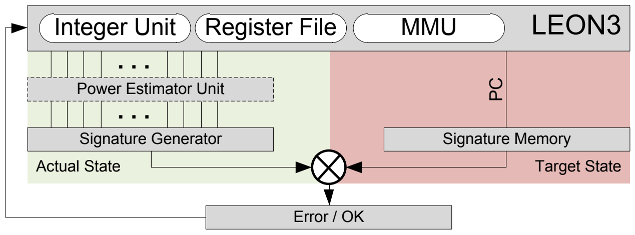


Fig. 4. Signature generation and comparison architecture

V. Experimental Results

The applicability of our approach is shown by using a widely known hardware platform based on an open source implementation of the SPARC v8 architecture developed by Aeroflex Gaisler (www.gaisler.com).This system-on-chip example has been synthesized using Xilinx ISE software and tested on the ML507 evaluation board provided by Xilinx. This processor type constitutes not a classical example of a smart-card processor because of its complexity but it is still used to prove the general applicability of the chosen approach. Our analysis platform consists of MathWorks Matlab 2010b on a six-core 3.2 GHz AMD Phenom-II machine.

Using the semi-custom design flow and power simulation tools provided by our industrial partner a power macro model containing 63 coefficients has been derived. The power estimation module itself implements this power model using a three stage pipelined adder structure.

The following experimental evaluations of the signature generator modules have been done using 100000 randomly generated input vectors. To enable a common simulation environment all blocks have been implemented into a single testbench scaled for different input widths.

A. Signature performance

First the signature hardware selection has been evaluated to determine if every input change also results in a signature change. Especially when using small signature widths or reduced implementations suboptimal hash generator hardware will lead to increased amount of such collisions. Figure 5 shows the behavior of every signature implementation for different hash widths. It can be clearly seen that while one-at-the-time is the worst and CRC-based the best, SHA-based and MD5-based modules result in similar results.

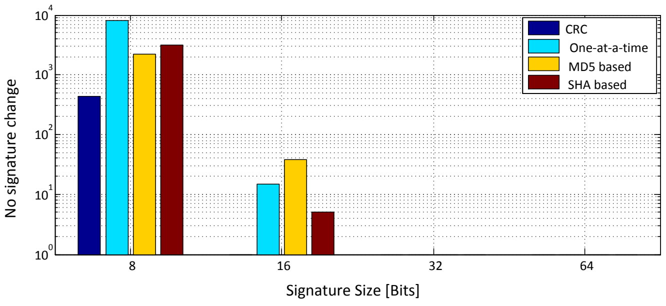


Fig. 5. No signature switching on input change

Second the amount of switched input bits compared to switching output bits has been evaluated. This ratio is supposed to be as low as possible, meaning that an input signal change results into a large change of the output value. The results are shown in Figure 6 concluding to a similar behavior of CRC-based, MD5-based and one-at-a-time implementations. Our SHA-based module had a significant less optimal performance than the other candidates.

B. Area Requirements of different signature implementations

In a resource-constrained environment hardware resource usage plays an important role, especially if the inclusion of a large amount of control signals should be enabled. Therefore, all modules have been synthesized for the Xilinx Virtex5 FPGA using different signature widths. Figure 7 shows the results of these synthesis runs concluding that all implementations scale as expected and one-at-a-time is needing the largest amount of FPGA slices. The lowest resource use was determined for all CRC-based modules.

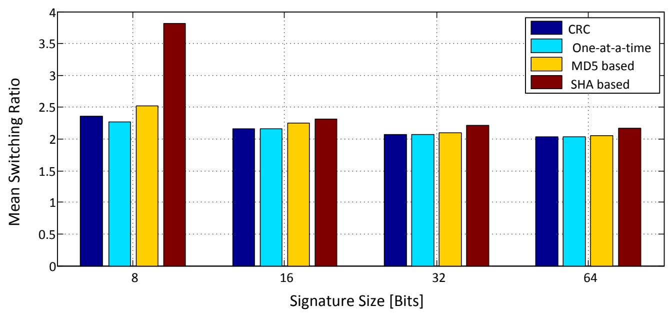


Fig. 6. Signature implementation input to output switching ratio

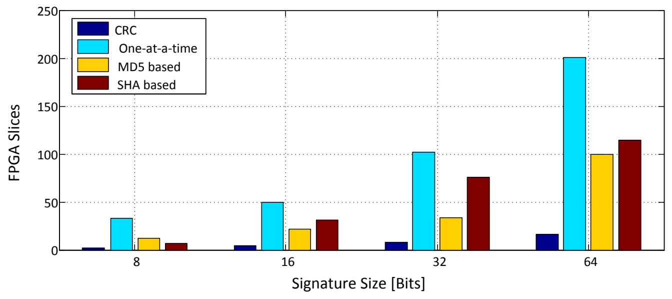


Fig. 7. Signature hardware resource usage

C. Evaluation using fault injection

After the signature hardware selection phase CRC proved to be the most promising candidate concerning hardware resource usage, scalability and hash performance. Therefore, a tree-structure has been implemented to generate a segmented hash containing information of three integrated processor cores. Each core outputs 63 state signals containing information about the processor's integer pipeline as well as register file, instruction cache, data cache, MMU and divider unit. This could be easily extended for further system units such as bus and network controllers. The monitored signals have been grouped into four segments per processor, resulting in five signature blocks per processor finally resulting in 15 hash generating hardware units. For small signature sizes (three, four, five) no compression has been implemented to avoid high collision ratios. The retrieved signatures contain 34 hash bits per processor core, resulting in a 102 bit system hash value.

Table I shows the resource usage of all signature generators combined in comparison to the complete system and one processor core. Furthermore, it provides a comparison to the implementations presented by Arora et. al. in [20]. Each of the monitored hardware units has been augmented using saboteur blocks to enable run-time fault injection.

TABLE I
RESOURCE USAGE COMPARISON

Unit	Slices	Slice Regs	LUTs	Resource [%]
System	15937	15017	30768	-
Core	3342	3248	6633	21.6
Signature Generator	103	102	263	0.85
Signature Controller	234	146	451	1.47
Arora et.al [20]	-	-	-	3-9[a]

[a]In [20] several different techniques have been proposed

To reduce the evaluation complexity only the first processor has been penetrated using 10000 randomly chosen fault patterns during our fault injection campaign. Table II gives an overview comparing our approach to the ones presented in [20] and [11].

TABLE II
FAULT DETECTION EFFICIENCY COMPARISON

Approach	Det. Bit Flips [%]	Det. latency [Instr.]
This work[a]	>99	1
Mao et.al Control flow [11]	26	23.6
Mao et.al Hash4 [11]	94	1
Arora et.al [20]	>99	6

[a]Single-round reduced CRC implementation

Our approach only needs a negligible amount of the system's resources while providing higher or similar detection rates than previously presented work. This result is achieved without influencing the execution performance as signature generation and comparison have been done in-system using dedicated memories. Communication to the integrity checking hardware is minimized to detection (de-)activation commands. Furthermore, in contrast to the earlier shown existing solutions our approach does not rely on any modification of the source or binary code.

VI. CONCLUSION

This paper presented a novel micro-architectural execution signature handling and characterization methodology for architectures providing existing control-state monitoring infrastructure. Our approach works fully transparent and does not rely on any changes of the executed software to detect unforeseen control flow changes. All presented hardware extensions have a low-impact on logic resources and can be efficiently integrated into existing power estimation infrastructure. Its modular design allows to generate execution signatures with varying signing granularity. Resulting signatures can then be used for transparent control-flow checking as well as for continuing modular-redundancy checking in high-security systems. The segmented approach allows for the direct localization of execution manipulations in a wide selection of system-on-chip submodules. All proposed circuits are fully synthesizable and have been successfully tested for their fault detection capability using an FPGA-based evaluation platform. Compared to existing state-of-the-art signature-based fault detection mechanisms our approach promises high detection efficiency without the need of additional watchdog hardware. The needed software overhead can be scaled according to the requirements of the chosen application.

ACKNOWLEDGMENT

The authors would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology, which funded the Power-Modes Project under the FIT-IT contract FFG 825749. We would also like to thank our project partners Infineon Technologies Austria AG and Austria Card GmbH.

REFERENCES

[1] A. Bhattacharjee, G. Contreras, and M. Martonosi, "Full-System Chip Multiprocessor Power Evaluations Using FPGA-Based Emulation," in *ISLPED*, 2008, pp. 335–340.

[2] C. Bachmann, A. Genser, C. Steger, R. Weiss, and J. Haid, "Automated Power Characterization for Run-Time Power Emulation of SoC Designs," in *DSD*, 2010, pp. 587–594.

[3] A. Krieg, J. Grinschgl, C. Steger, R. Weiss, and J. Haid, "A side channel attack countermeasure using system-on-chip power profile scrambling," in *IOLTS*. IEEE, 2011, pp. 222–227.

[4] M. Rebaudengo, S. Reorda, M. Torchiano, and M. Violante, "Soft-error detection through software fault-tolerance techniques," in *DFT*. IEEE, 1999, pp. 210–218.

[5] O. Goloubeva, M. Rebaudengo, M. Reorda, and M. Violante, "Soft-error detection using control flow assertions," 2003.

[6] R. Venkatasubramanian, J. Hayes, and B. Murray, "Low-cost on-line fault detection using control flow assertions," in *On-Line Testing Symposium, 2003. IOLTS 2003. 9th IEEE*. IEEE, 2003, pp. 137–143.

[7] N. Oh, P. Shirvani, and E. McCluskey, "Control-flow checking by software signatures," *Reliability, IEEE Transactions on*, vol. 51, no. 1, pp. 111–122, 2002.

[8] S. Daniels, "A concurrent test technique for standard microprocessors," *Dig. Papers Compcon Spring*, vol. 83, pp. 389–394, 1983.

[9] V. Iyengar and L. Kinney, "Concurrent fault detection in microprogrammed control units," *Computers, IEEE Transactions on*, vol. 100, no. 9, pp. 810–821, 1985.

[10] K. Wilken and T. Kong, "Concurrent detection of software and hardware data-access faults," *Comp., IEEE Tr.*, vol. 46, no. 4, pp. 412–424, 1997.

[11] S. Mao and T. Wolf, "Hardware support for secure processing in embedded systems," in *DAC*. ACM, 2007, pp. 483–488.

[12] S. Lukovic, P. Pezzino, and L. Fiorin, "Stack Protection Unit as a step towards securing MPSoCs," in *IPDPSW*. IEEE, 2010, pp. 1–4.

[13] S. Kim and A. Somani, "On-line integrity monitoring of microprocessor control logic," *Microelectronics*, vol. 32, no. 12, pp. 999–1007, 2001.

[14] R. Ragel, S. Parameswaran, and S. Kia, "Micro embedded monitoring for security in application specific instruction-set processors," in *CASES*. ACM, 2005, pp. 304–314.

[15] Y. Fei and Z. Shi, "Microarchitectural support for program code integrity monitoring in application-specific instruction set processors," in *DATE*. EDA Consortium, 2007, pp. 815–820.

[16] K. Patel, S. Parameswaran, and R. Ragel, "Architectural frameworks for security and reliability of mpsocs," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, no. 99, pp. 1–14, 2010.

[17] J. Smolens, B. Gold, J. Hoe, B. Falsafi, and K. Mai, "Detecting emerging wearout faults," in *SELSE*. Citeseer, 2007.

[18] J. Smolens, B. Gold, J. Kim, B. Falsafi, J. Hoe, and A. Nowatzyk, "Fingerprinting: bounding soft-error detection latency and bandwidth," in *ACM SIGARCH C.A.N.*, vol. 32, no. 5. ACM, 2004, pp. 224–234.

[19] A. Krieg, C. Bachmann, J. Grinschgl, C. Steger, R. Weiss, and J. Haid, "Accelerating early design phase differential power analysis using power emulation techniques," in *HOST*. IEEE, 2011, pp. 81–86.

[20] D. Arora, S. Ravi, A. Raghunathan, and N. Jha, "Secure embedded processing through hardware-assisted run-time monitoring," in *DATE*. IEEE Computer Society, 2005, pp. 178–183.

[21] B. Jenkins, "Hash functions for hash table lookup," Online, August 2011. [Online]. Available: http://www.burtleburtle.net/bob/hash/evahash.html

[22] Outputlogic.com, "CRC Generator," September 2011.

Acceleration of Fault Attack Emulation by Consideration of Fault Propagation

Armin Krieg, Johannes Grinschgl,
Christian Steger and Reinhold Weiss
Institute for Technical Informatics
Graz University of Technology, Austria
{armin.krieg, johannes.grinschgl,
steger, rweiss}@tugraz.at

Holger Bock, Josef Haid
Design Center Graz
Infineon Technologies Austria AG
{josef.haid, holger.bock}@infineon.com

***Abstract*—In recent years the number of deployed embedded systems increased significantly. These system-on-chips are widely used for high-availability as well as security applications. Therefore, the reliable operation of these devices plays a vital role and disturbed operation can lead to loss of confidence and trust. To ensure correct operation during random or intentional fault events, injection techniques for system simulation and emulation have been presented. The targeted use of these approaches is often difficult because of the device complexity and the lack of knowledge about internal processes after a fault has been activated. To improve the current state-of-the-art in this field this paper presents fault propagation analysis and hardware checker generation techniques based on static VHDL code analysis. These help to gain a deeper understanding of system internal propagation paths and their influence on normal operation. Physical layout data is included to enable the mapping of a fault attack location to its corresponding logic gates. Hardware checkers enable higher fault injection evaluation efficiency by removing masked system parts from the target space.**

I. Introduction

The continuing introduction of smart-cards into the monetary cycle and for personal ID applications created a viable target for criminal subjects [1]. This led to intense research concerning attack methods and corresponding countermeasures to prevent the leakage of vital information to the environment. Basically, two different kinds of attacks can be differentiated, passive ones, listening to the device's behavior and active ones, influencing normal operation by the introduction of faults. Such fault-based attacks are used to drive the system into an unintended state to disable existing countermeasures and to disrupt the normal mode of cryptographic operation [2].

These issues resulted in a wide range of research concerning fault simulation and emulation techniques. Using these methodologies software and hardware developers should be enabled to test their implementations for its robustness against operational errors. The injection of single and multi-bit faults has been refined to enable high performant evaluation campaigns. These evaluation platforms have been used to replicate the effects of soft and hard errors on embedded system implementations. Unfortunately there is a distinctive lack of research concerning the link between low-level faults and high-level erroneous results. Especially when testing large system-on-chips the success of a fault injection can only be seen through errors manifesting themselves on system outputs or memories. Faults that have been temporarily masked could lead to errors during later operation and hence, could result in false evaluation outcomes. Such problems are of increasing concern to semiconductor manufacturers during pre- and post-certification testing.

Therefore, there is a strong need for static code analysis techniques to determine fault propagation paths and to provide efficient injection platforms. In this work a novel methodology is presented, combining knowledge about fault attack locations and internal propagation paths to enable real-time emulation of logic fault effects on system operation (depicted in Figure 1). The incorporation of the precise fault attack location is specifically important in case of accurate laser attacks as used by test laboratories and adversaries [3].

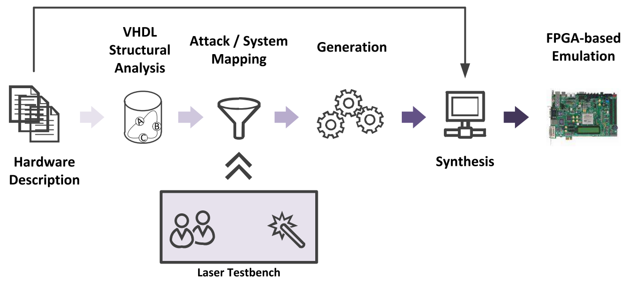


Fig. 1. Global fault attack emulation flow

The main contributions of this work are:

- A static VHDL analysis flow to provide the automatic generation of fault injection emulation support modules.
- Introduction of a novel fault attack propagation analysis flow for post-laser injection evaluation analysis.

This paper is structured as follows. First, Section II will give a brief introduction into static analysis methods of VHDL code.Followed by Section III introducing our methodology for efficient emulation of fault-based attacks. Section IV compares our methodology and results with recent developments in this area. In Section V experimental results are shown to prove the applicability of our approach. Finally, Section VI concludes this work and gives a short overview over our future work.

II. Static system analysis

For security evaluations not only information of the basic system structure is sufficient, therefore our approach includes also information from the device layout and the laser attack coordinates used on a physical test bench.

A. Static code analysis

The first step includes the automatized extraction of the signal and variable dependency graphs. This process consists of three stages.

VHDL File Parsing : Existing digital design flows already include simple interdependency checking mechanisms to optimize their compilation processes. While our proposed VHDL parsing engine is able to process large VHDL file compilation, such pre-processing stages can be used to reduce the parsing effort needed during this stage. Basically all significant VHDL file elements are extracted e.g. signals, variables, process and constants. This includes eventual assignment targets and sources or elements inside sensitivity-lists.

Local Dependency Extraction : After all significant elements are available inside an internal database, local dependency extraction can be started. This process consists of processing all assignment source and target lists to generate local (module-internal) dependency graphs.

Global Dependency Extraction : The result of the local extraction step are individual objects containing dependency graphs. These are finally connected to gain a complete system graph including all found objects and their relations.

B. Layout analysis

After the hardware description has been analyzed a connection between layout position data and the laser attack coordinates has to be established. Again this flow includes three basic stages that have to be executed to provide an accurate fault model.

Layout-Location-Data : The first step contains the extraction of the logic gate positions inside the finalized layout. This information is included in the DEF-file produced by layout tools such as those provided by the Cadence digital implementation flow.

Laser-Grid-Positions : These laser attack positions are provided by the laser bench test engineer team. Basically every order of positions can be used, most likely this will be a systematic grid.

Attack-Position-Mapping : A mapping process is finally used to identify logic gates that are affected by the chosen laser attack. The effect radius depends on the accuracy of the chosen laser beam. Additionally knowledge of physical design engineers can be included into the determination of affected gates. Such information is depending on laser spot size, shot duration and pulse strength.

C. Location to RTL mapping

The results of the former two process steps now has to be connected to retrieve a connection between the physical layout and the higher-level hardware description. This mapping process consists of three stages.

Signal Filtering : The layout processing stage resulted in a list of affected gates and signals and their corresponding positions on the layout. This information is now filtered, because not every element known at the RTL level is also visible after synthesis.

Signal-to-Node Mapping : Now only elements are available that also visible in the RTL description, therefore it is possible to map the physical positions to internal nodes existing in the dependency graphs.

Node-to-Graph Mapping : Finally, these nodes are mapped into the global design dependency graph to enable further dependency analysis.

After RTL and layout extraction and mapping processes, a complete database is available containing affected elements and their relations. This information can now be used to generate checker modules and the optimum positions for saboteur blocks to emulate laser attack effects.

III. Fault attack emulation

An accurate physical fault model now has been established and can be included into an FPGA-based emulation platform. The flow starting with the generation of saboteur modules and resulting in evaluation conclusions is divided into four steps.

Generation Process : First, saboteurs and checker modules have to be generated depending on the amount of signals that have to be disturbed or monitored. This information is taken from the local dependency graphs of the selected attack targets. Manipulations are done using free configurable saboteurs and system properties are monitored using trigger modules (similar to the approach taken in [4]).

Placement Process : During the placement process the generated modules are placed into the RTL description of the target system and are connected to analysis support blocks. Such blocks include the fault injection controller which is handling all the saboteurs and controlling the injection process.

Injection Process : At this stage the RTL source code has been prepared and synthesized to fit into the FPGA of the evaluation platform. The fault injection control is configured with all selected attack patterns and starts with the evaluation the system-under-test. Results from these injection campaigns can be directly derived from placed checker modules or by monitoring selected outputs and memories.

Analysis Process : All stored evaluation results are finally stored onto a connected PC and analyzed to identify unprotected information paths. Also this information helps to understand possible failures identified during laser bench tests that are part of system certification.

The final emulation architecture includes a controlling micro-processor to ensure a fast and independent injection process as shown in Figure 2. This system-external controller can also be used to efficiently monitor certain aspects of the target system like state signals or memory buses. Fault attack effects are mapped to the design using saboteur modules that emulate various signal manipulation types.

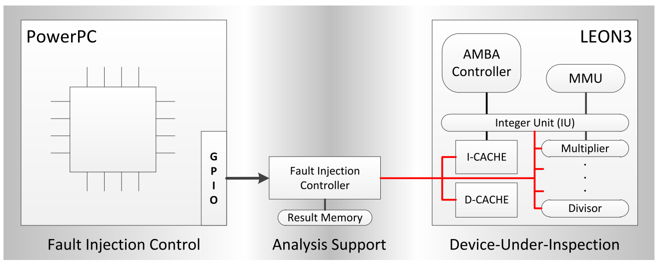


Fig. 2. FPGA-based fault injection emulation architecture - High performance fault injection evaluation for secure software verification

IV. Related work

A. Fault simulation and emulation

An early implementation of such a simulation environment under fault conditions was called MEFISTO [5]. In case of MEFISTO the behavior of the system is directly influenced by small saboteur modules placed between target signal and variable sources and sinks. The following years various improvements to this simulation approach have been presented, mainly to improve simulation speed, e.g., the VFIT tool presented in [6]. In the last years also a wide range of SystemC-based approaches have been proposed [7], [8]. At the cost of injection flexibility hardware-accelerated FPGA-based approaches have been presented to increase performance and to enable large-scale dependability investigations [9]. In [10], authors showed that this single-fault model is not sufficient if intentional faults, as expected in security evaluations, are concerned. Specifically targeting fault simulation of optical fault attacks, an approach has been presented by the authors of [11] and [12]. As test costs demanded by certification labs increased dramatically, high speed software verification using hardware-accelerated methods became important. A modular methodology for such an emulation-based platform has been shown in [4].

B. Fault propagation modeling

For many years low-level logic evaluation for soft-error reliability (SER) has been done using slow but very accurate SPICE-based simulations. Few years ago the authors of [13] introduced various new techniques based on binary decision diagrams to improve such investigations performance-wise. Similarly, a signature-based approach has been introduced in [14] to analyze SER and logic masking in combinational and sequential circuits. Especially of interest in the security domain is the work proposed in [15] specifically targeting multiple transient faults as they would be expected during complex fault-attacks.

C. Static VHDL code analysis

Timing and particularly worst-case execution time (WCET) analysis is an important part of the evaluation of hard real-time and safety systems. To automate this error-prone process, which includes a hard engineering effort, the authors of [16] presented a frame work for the static analysis of VHDL code. In this early work the VHDL description is treated like a sequential program to derive a control flow description in CRL2. The same research group introduced an abstraction-aware compiler for such hardware description models in [17]. Finally, this semi-automatic process of hardware model timing analyses has been completed with the framework shown in [18]. In the security domain not only control flow, but also flow of information is of essential importance. Therefore, a state-machine-based approach has been presented in [19] to determine information flow at a higher abstraction level.

V. Experimental Results

Our proposed methodology is implemented and demonstrated using an open source implementation of the SPARC v8 architecture developed by Aeroflex Gaisler.This system-on-chip example has been synthesized using Xilinx ISE software and is functionally tested on the ML507 evaluation board provided by Xilinx.

A. Attack mapping

The first step includes the mapping of a laser-based attack to the hardware description of the emulation system. To increase demonstration quality the chosen attack-path is very short and locally limited. Such a sample file includes information about laser impact radius and targeted coordinates.

This file is parsed and influenced hardware elements are identified using the DEF-file of the place&route process. In this case a 32-bit wide register inside the AHB-controller of the LEON3 processor has been disturbed. The resulting dependency graph for this node is depicted in Figure 3. Full black arrows mean signal or variable assignments from one node to another. Dotted lines indicate that the target node is sensitive on the given source node because the assignment happens inside a process. Bold grey lines depict enable relation-ships if the assignment is located inside an if-structure.

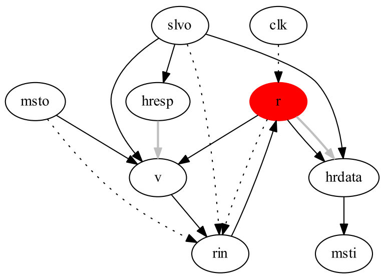


Fig. 3. Simplified isolated dependency graph of influenced node

The influenced node is part of a ring-dependency, meaning that the reading of **rin** is triggered by the two inputs **msto** and **slvo** while writing into node **r** is triggered only by the clock-signal. The backward connection is realized via variable **v**, which is the only externally written part. The writing of **v** is enabled via variable **hresp**. Finally the result residing in **r** is written to the output **msti**. This information can now be used for the generation of hardware modules for efficient emulation of this chosen attack. This includes saboteur modules to manipulate this register in a way our chosen laser impulse would and hardware checkers to enable these saboteurs only when this attack would not have been masked.

B. Attack emulation and effectiveness evaluation

Such isolated dependency graphs are now processed for the further generation of VHDL hardware blocks like saboteurs or checkers. In the former case the simple manipulation of the targeted element can be relatively simply implemented and several approaches using saboteurs have been shown in Section IV. On the other hand, checker modules allow more in-depth analysis of internal processes after fault activation.

In this combinatorial block a change of **r** directly triggers the writing of **rin**. The graph in this case (record types) is a bit misleading because only a very small part of **r** is influenced. One control signal of this record is also enabling the writing of **hrdata** and hence the fault propagates to the output. Such a checker generated trace can be seen in Figure 4, visualizing that there are large parts of the execution that are insensitive to this attack. As seen in Table I only 11 percent of all accesses to this register would lead to a propagation of this fault because in all other cases there is no enable.

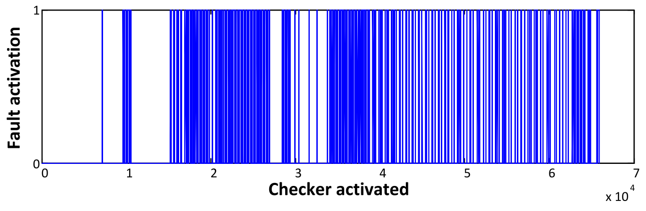


Fig. 4. Checker result trace - CPU startup and AES encryption

TABLE I
CHECKER EVALUATION RESULT FOR A SINGLE ATTACKED NODE

Checker Name	Checker activ.	Fault sens.	Fault sens. [%]
AHBCTRL_HRDATAS	65896	7261	11

Besides the observation of node-activity of our attacked system blocks, we are enabled to implement saboteur modules at this point to emulate signal manipulations. In such a case the results of the placed support modules, like these checker blocks, will provide input to the injection control to trigger attacks at fault sensitive points of time.

VI. Conclusion

In this work a novel methodology for the efficient generation of hardware checker modules and fault emulation support blocks is presented. Static VHDL code analysis methods have been applied to extract fault propagation paths from the investigated system and combined with information gained after physical evaluation. Physical properties of the laser point of attack are abstracted into a single region of logic interaction. This way a gap between physical attacks using laser impulses and the RTL design is closed to enable a better understanding of effects caused by physical fault injection.

Our approach has been tested using open available source of a system-on-chip implementation and its modular software design allows for the integration into existing JAVA-based design tools.

Acknowledgments

The authors would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology, which funded the Power-Modes Project under the FIT-IT contract FFG 825749. We would also like to thank our project partners Infineon Technologies Austria AG and Austria Card GmbH.

References

[1] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in embedded systems: Design challenges," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 3, pp. 461–491, 2004.

[2] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, and S. Moderator-Ravi, "Security as a new dimension in embedded system design," in *Design Automation Conference*. ACM, 2004, pp. 753–760.

[3] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, 2006.

[4] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, and J. Haid, "Modular fault injector for multiple fault dependability and security evaluations," in *Euromicro DSD*. IEEE, 2011, pp. 550–557.

[5] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into VHDL models: the MEFISTO tool," in *Fault-Tolerant Computing, Twenty-Fourth International Symposium on*. IEEE, 1994, pp. 66–75.

[6] J. Baraza, J. Gracia, S. Blanc, D. Gil, and P. Gil, "Enhancement of fault injection techniques based on the modification of VHDL code," *VLSI, IEEE Trans. on*, vol. 16, no. 6, pp. 693–706, 2008.

[7] K. Rothbart, U. Neffe, C. Steger, R. Weiss, E. Rieger, and A. Muehlberger, "High level fault injection for attack simulation in smart cards," in *Test Symposium, 2004. 13th Asian*. IEEE, 2004, pp. 118–121.

[8] S. Misera, H. Vierhaus, and A. Sieber, "Simulated fault injections and their acceleration in SystemC," *Microprocessors and Microsystems*, vol. 32, no. 5-6, pp. 270–278, 2008.

[9] D. de Andrés, J. Ruiz, D. Gil, and P. Gil, "Fault emulation for dependability evaluation of VLSI systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 4, pp. 422–431, 2008.

[10] R. Leveugle, "Early analysis of fault-based attack effects in secure circuits," *Comp., IEEE Trans. on*, vol. 56, no. 10, pp. 1431–1434, 2007.

[11] S. Skorobogatov, "Semi-invasive attacks-a new approach to hardware security analysis," *Technical report*, 2005.

[12] H. Li and S. Moore, "Security evaluation at design time against optical fault injection attacks," *Information Security, IEE Proceedings*, vol. 153, no. 1, pp. 3 – 11, march 2006.

[13] N. Miskov-Zivanov and D. Marculescu, "Modeling and optimization for soft-error reliability of sequential circuits," *CAD of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 5, pp. 803–816, 2008.

[14] S. Krishnaswamy, S. Plaza, I. Markov, and J. Hayes, "Signature-based SER analysis and design of logic circuits," *CAD of Integrated Circuits and Systems, IEEE Trans. on*, vol. 28, no. 1, pp. 74–86, 2009.

[15] N. Miskov-Zivanov and D. Marculescu, "Multiple transient faults in combinational and sequential circuits: a systematic approach," *CAD of ICs and Systems, IEEE Trans. on*, vol. 29, no. 10, pp. 1614–1627, 2010.

[16] M. Schlickling and M. Pister, "A Framework for Static Analysis of VHDL Code," in *7th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*, C. Rochange, Ed., Dagstuhl, Germany, 2007.

[17] M. Maksoud, M. Pister, and M. Schlickling, "An abstraction-aware compiler for VHDL models," in *Computer Engineering & Systems. International Conference on*. IEEE, 2009, pp. 3–9.

[18] M. Schlickling and M. Pister, "Semi-automatic derivation of timing models for WCET analysis," in *ACM SIGPLAN Notices*, vol. 45, no. 4. ACM, 2010, pp. 67–76.

[19] X. Li, M. Tiwari, B. Hardekopf, T. Sherwood, and F. Chong, "Secure information flow analysis for hardware design: Using the right abstraction for the job," in *Program. Lang. and Anal. for Sec.* ACM, 2010.

Bibliography

[1] International Technology Roadmap for Semiconductors. System driver chapter 2010 updates. http://www.itrs.net/, 2010.

[2] Smart Card Alliance. Rfid tags and contactless smart card technology: Comparing and contrasting applications and capabilities. http://www.smartcardalliance.org/, 2012.

[3] Marc Bertin. Market trends and forecasts. World Card Summit CARTES exhibition, http://www.eurosmart.com/, 2011.

[4] GAO. National partnership offers benefits, but faces considerable challenges. *United States Government Accountability Office*, 2006.

[5] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, and J. Haid. Modular fault injector for multiple fault dependability and security evaluations. In *Digital System Design (DSD), 2011 14th Euromicro Conference on*, pages 550 –557, 31 2011-sept. 2 2011.

[6] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, and J. Haid. Automatic saboteur placement for emulation-based multi-bit fault injection. In *Proc. 6th Int Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC) Workshop*, pages 1–8, 2011.

[7] Johannes Grinschgl, Armin Krieg, Christian Steger, Reinhold Weiss, Holger Bock, and Josef Haid. Efficient fault emulation using automatic pre-injection memory access analysis. In *SOC Conference (SOCC), 2012 IEEE International*, pages 277 –282, sept. 2012.

[8] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, and J. Haid. Efficient fault emulation based on post-injection fault effect analysis (pifea). In *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on*, pages 526 –529, aug. 2012.

[9] Armin Krieg, Johannes Grinschgl, Christian Steger, Reinhold Weiss, Holger Bock, and Josef Haid. Power-modes: Power-emulator- and model-based dependability and security evaluations. *ACM Trans. Reconfigurable Technol. Syst.*, 5(4):19:1–19:21, December 2012.

[10] A. Krieg, J. Grinschgl, C. Steger, R. Weiss, A. Genser, H. Bock, and J. Haid. Characterization and handling of low-cost micro-architectural signatures in mpsocs. In *Test Symposium (ETS), 2012 17th IEEE European*, pages 1 –6, may 2012.

[11] Johannes Grinschgl, Armin Krieg, Christian Steger, Reinhold Weiss, Holger Bock, Josef Haid, Thomas Aichinger, and Christiane Ulbricht. Case study on multiple fault dependability and security evaluations. *Microprocessors and Microsystems*, (0):in press, 2012.

[12] Metasec - mobile energy-efficient trustworthy authentication system with elliptic curve based security. fit-it project proposal cooperation research projects, graz university of technology, 2011.

[13] Cambridge University Press. Cambridge dictionaries online. http://dictionary.cambridge.org/, 1 2013.

[14] B. Vigna. More than moore: micro-machined products enable new applications and open new markets. In *Proc. IEDM Technical Digest Electron Devices Meeting IEEE Int*, 2005.

[15] G. Q. Zhang, F. van Roosmalen, and M. Graef. The paradigm of "more than moore". In *Proc. 6th Int Electronic Packaging Technology Conf*, pages 17–24, 2005.

[16] Smart Card Alliance. Smart cards applications. http://www.smartcardalliance.org/, 2012.

[17] 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, feb 2006.

[18] David De Andres, Juan Carlos Ruiz, Daniel Gil, and Pedro Gil. Fades: a fault emulation tool for fast dependability assessment. In *Proc. IEEE Int. Conf. Field Programmable Technology (FPT 2006)*, pages 221–228, 2006.

[19] A. Benso and B. Prinetto. *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation.* Kluwer Academic Publishers, 2003.

[20] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, and M. Violante. An fpga-based approach for speeding-up fault injection campaigns on safety-critical circuits. *Journal of Electronic Testing*, 18(3):261–271, 2002. 10.1023/A:1015079004512.

[21] J. Guthoff and V. Sieh. Combining software-implemented and simulation-based fault injection into a single fault injection method. In *Proc. Twenty-Fifth Int Fault-Tolerant Computing FTCS-25. Digest of Papers. Symp*, pages 196–206, 1995.

[22] K. Rothbart, U. Neffe, Ch. Steger, R. Weiss, E. Rieger, and A. Muehlberger. High level fault injection for attack simulation in smart cards. In *Test Symposium, 2004. 13th Asian*, pages 118 – 121, nov. 2004.

[23] Y. Torroja, T. Riesgo, E. de la Torre, and J. Uceda. A comparative analysis of different fault simulation techniques for vlsi circuits testing. In *Proc. IECON '91. Conf. Int Industrial Electronics, Control and Instrumentation*, pages 1226–1231, 1991.

[24] Silvio Misera, Heinrich Theodor Vierhaus, Lars Breitenfeld, and Andr? Sieber. A mixed language fault simulation of vhdl and systemc. *Digital Systems Design, Euromicro Symposium on*, 0:275–279, 2006.

[25] Johan Karlsson and Peter Folkesson. Application of three physical fault injection techniques to the experimental assessment of the mars architecture. pages 267–287. IEEE Computer Society Press, 1995.

[26] Uros Legat, Anton Biasizzo, and Franc Novak. Automated seu fault emulation using partial fpga reconfiguration. In *Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on*, pages 24 –27, april 2010.

[27] A. Krieg, C. Preschern, J. Grinschgl, C. Kreiner, C. Steger, R. Wei and, H. Bock, and J. Haid. Power and fault emulation for software verification and system stability testing in safety critical environments. *Industrial Informatics, IEEE Transactions on*, PP(99):1, 2012.

[28] C. Dunbar and K. Nepal. Fault emulation and test pattern generation using reconfigurable computing. In *Circuits and Systems (MWSCAS), 2010 53rd IEEE International Midwest Symposium on*, pages 797 –800, aug. 2010.

[29] M. Portela-Garcia, C. Lopez-Ongil, M. Garcia-Valderas, E.S. Millan, and L. Entrena. Fast ser evaluation of embedded rams in fault emulation systems. In *Radiation and Its Effects on Components and Systems (RADECS), 2009 European Conference on*, pages 256 –259, sept. 2009.

[30] J.C. Baraza, J. Gracia, D. Gil, and P.J. Gil. Improvement of fault injection techniques based on vhdl code modification. In *Tenth IEEE International High-Level Design Validation and Test Workshop 2005*, pages 19 – 26, 30 2005.

[31] S. Bayar and A. Yurdakul. Self-reconfiguration on spartan-iii fpgas with compressed partial bitstreams via a parallel configuration access port (cpcap) core. In *Proc. Ph.D. Research in Microelectronics and Electronics PRIME 2008*, pages 137–140, 2008.

[32] P. Kenterlis, N. Kranitis, A. Paschalis, D. Gizopoulos, and M. Psarakis. A low-cost seu fault emulation platform for sram-based fpgas. In *Proc. 12th IEEE Int. On-Line Testing Symp. IOLTS 2006*, 2006.

[33] Leos Kafka. Analysis of applicability of partial runtime reconfiguration in fault emulator in xilinx fpgas. In *DDECS '08: Proceedings of the 2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pages 1–4, Washington, DC, USA, 2008. IEEE Computer Society.

[34] L. Sterpone and M. Violante. A new partial reconfiguration-based fault-injection system to evaluate seu effects in sram-based fpgas. *Nuclear Science, IEEE Transactions on*, 54(4):965 –970, 2007.

[35] Bradley F. Dutton, Mustafa Ali, Charles E. Stroud, and John Sunwoo. Embedded processor based fault injection and seu emulation for fpgas. In *Proc. International Conf. on Embedded Systems and Applications 2009*, pages 183–189, 2009.

[36] Cristiana Bolchini, Antonio Miele, and Donatella Sciuto. Fault models and injection strategies in systemc specifications. In *Digital System Design Architectures, Methods and Tools, 2008. DSD '08. 11th EUROMICRO Conference on*, pages 88 –95, sept. 2008.

[37] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert. Statistical fault injection: Quantified error and confidence. In *Proc. DATE '09. Design, Automation & Test in Europe Conf. & Exhibition*, pages 502–506, 2009.

[38] R. Leveugle, A. Calvez, P. Vanhauwaert, and P. Maistri. Precisely controlling the duration of fault injection campaigns: a statistical view. In *Proc. 4th Int. Conf. Design & Technology of Integrated Systems in Nanoscal Era DTIS '09*, pages 149–154, 2009.

[39] Raul Barbosa. Fault injection optimization through assembly-level pre-injection analysis. Master's thesis, CHALMERS UNIVERSITY OF TECHNOLOGY Department of Computer Engineering Gteborg 2004, 2004.

[40] Raul Barbosa, Jonny Vinter, Peter Folkesson, and Johan Karlsson. Assembly-level preinjection analysis for improving fault injection efficiency. In *in Proceedings of the Fifth European Dependable Computing Conference (EDCC-5*, 2005.

[41] L. Berrojo, I. Gonzalez, F. Corno, M. S. Reorda, G. Squillero, L. Entrena, and C. Lopez. New techniques for speeding-up fault-injection campaigns. In *Proc. Design, Automation and Test in Europe Conf. and Exhibition*, pages 847–852, 2002.

[42] A. Gofuku, S. Koide, and N. Shimada. Fault tree analysis and failure mode effects analysis based on multi-level flow modeling and causality estimation. In *SICE-ICASE, 2006. International Joint Conference*, pages 497 –500, oct. 2006.

[43] R. Leveugle. Early analysis of fault-based attack effects in secure circuits. *IEEE Transactions on Computers*, 56(10):1431–1434, 2007.

[44] Xilinx. Virtex-5 fpga family. http://www.xilinx.com/, 07 2010.

[45] Gaisler. Leon3 processor. http://www.gaisler.com/, 07 2010.

[46] M. Neve, E. Peeters, D. Samyde, and J.-J. Quisquater. Memories: A survey of their secure uses in smart cards. In *Proc. Second IEEE Int. Security in Storage Workshop SISW '03*, 2003.

[47] A. Pellegrini, V. Bertacco, and T. Austin. Fault-based attack of rsa authentication. In *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pages 855–860, 2010.

[48] B. Gladman. Implementations of AES (Rjindael) in C/C++ and assembler. http://gladman.plushost.co.uk/, December 2010.