MetaFS: Model-driven Fault Simulation Framework

Endri Kaja†, Nicolas Gerlin†, Monideep Bora‡,
Keerthikumara Devarajegowda†, Dominik Stoffel†, Wolfgang Kunz‡, Wolfgang Ecker†
†Infineon Technologies AG, Germany – ‡Technische Universität Kaiserslautern, Germany
§Albert-Ludwigs-Universität Freiburg, Germany – †Technische Universität München, Germany

Abstract—The adoption of new technologies by the automotive industry drives the need for electronic component suppliers to assess and scrutinize the risk of technologies that are being integrated into the safety-critical systems. To cope with these challenges, engineers are constantly looking for highly automated and efficient functional safety approaches to achieve the required certifications for their designs. In this paper, we propose MetaFS, a metamodel-based simulator-independent fault simulation framework that provides multi-purpose fault injection strategies such as statistical fault injection, direct fault injection, and exhaustive fault injection. The framework enables the injection of stuck-at faults, single-event transients, single-event upsets as well as timing faults. The proposed approach scales to a wide range of RISC-V based CPU subsystems with support for various RISC-V ISA standard extensions and, additionally, safety and security related custom instruction extensions. The subsystems were running the Dhrystone application and a specific in-house Fingerprint calculation application respectively. A minimal effort of 1 person-day was required to conduct 22 different fault simulation campaigns, providing significant data regarding subsystem failure rates.

Index Terms—Model-driven fault simulation, Safety analysis

I. INTRODUCTION AND BACKGROUND

Natural radiation, aging, mechanical stress, processing defects, and other events may impact the intended behavior of a semiconductor device by introducing numerous faults. When an induced fault causes an incorrect behavior of the device, a failure occurs, which may lead to a disastrous outcome. Due to the increased complexity and growing transistor density of modern System-on-Chips (SoCs), the probability of failures is noticeably higher [8], ISO26262, the functional safety standard for the automotive domain, recommends fault injection as the preferred technique to validate circuit and system dependability. Fault injection is defined as the validation technique of fault-tolerant systems by observing the behavior of the system in the presence of injected faults [2], [17]. Fault models are used to abstract the effect of faults onto different abstraction levels of digital designs. The widely applied fault models on the circuit level of a design are permanent and transient faults. Simulation-based fault injection approaches need to have a standard for the automotive domain, recommends fault injection as the preferred technique to validate circuit and system dependability. Fault injection is defined as the validation technique of fault-tolerant systems by observing the behavior of the system in the presence of injected faults [2], [17]. Fault models are used to abstract the effect of faults onto different abstraction levels of digital designs. The widely applied fault models on the circuit level of a design are permanent and transient faults.

Simulation-based fault injection approaches need to have a standard for the automotive domain, recommends fault injection as the preferred technique to validate circuit and system dependability. Fault injection is defined as the validation technique of fault-tolerant systems by observing the behavior of the system in the presence of injected faults [2], [17]. Fault models are used to abstract the effect of faults onto different abstraction levels of digital designs. The widely applied fault models on the circuit level of a design are permanent and transient faults. Simulation-based fault injection approaches need to have a standard for the automotive domain, recommends fault injection as the preferred technique to validate circuit and system dependability. Fault injection is defined as the validation technique of fault-tolerant systems by observing the behavior of the system in the presence of injected faults [2], [17]. Fault models are used to abstract the effect of faults onto different abstraction levels of digital designs. The widely applied fault models on the circuit level of a design are permanent and transient faults. Simulation-based fault injection approaches need to have a standard for the automotive domain, recommends fault injection as the preferred technique to validate circuit and system dependability. Fault injection is defined as the validation technique of fault-tolerant systems by observing the behavior of the system in the presence of injected faults [2], [17]. Fault models are used to abstract the effect of faults onto different abstraction levels of digital designs. The widely applied fault models on the circuit level of a design are permanent and transient faults. Simulation-based fault injection approaches need to have a standard for the automotive domain, recommends fault injection as the preferred technique to validate circuit and system dependability. Fault injection is defined as the validation technique of fault-tolerant systems by observing the behavior of the system in the presence of injected faults [2], [17]. Fault models are used to abstract the effect of faults onto different abstraction levels of digital designs. The widely applied fault models on the circuit level of a design are permanent and transient faults.
II. EXTENDING THE FAULT INJECTION FLOW

In the following, we describe the extension of the fault injection flow with timing fault and SEUs.

A. Timing Faults

Timing faults, also known as transition-delay faults, are a result of manufacturing defects ([13]) and manifest as a delayed response in the system. The affected parts of the design calculate the correct result, but the response is observed at a later timepoint than in the fault-free scenario.

Consider a two input logic AND gate (shown in left side of Fig. 2(a)) with input A having a logical value of 1'b1 for three consecutive clock cycles. Input B becomes 1'b1 in the next clock cycle after A becomes 1'b1 and, consequently, C becomes 1'b1 at the same clock cycle. Due to a timing fault at the AND gate input, the rising edge of input B is delayed by one clock cycle and, therefore, the correct value at the output port is delayed as well. To replicate the behavior displayed above, the existing saboteurs in [7] are extended as shown in Fig. 2(b). Since the injection flow utilizes a mixed RTL/GL granularity, it is possible to use flip-flop stages, to induce a one or more clock cycle delays. When least significant bit of control signal CTRL is set to 1'b1, the output Out is equal to IN value delayed by one clock cycle. Hence, a timing fault is injected to the desired signal location.

![Fig. 2. Timing fault saboteur](image)

B. Single Event Upset

Single event upset faults occur when radiation causes enough disturbance to flip the logical value of a memory cell (latch or flip-flop). Soft errors are the major source to cause transforming an enabled D flip-flop is shown in Fig. 3. Both the circuits are equivalent, i.e. the cell is written only when the enable signal is activated. Therefore, by adding the saboteur at the input of the cell, as shown in color red in Fig. 3, the value of the cell is flipped independently of the enable signal.

![Fig. 3. Injection of SEU into a D flip-flop with an enable input](image)

The existing saboteurs in the presented fault injection flow trigger a bitflip on the intended signal. However, some state-saving GL technology cells (registers, latches) depend on an enable signal to activate the fault, i.e. if the cell is not activated, the fault is masked. If the testing pattern does not consider the enable signal of a memory cell, it could potentially lead to a dangerous behavior of the design and affect the dependability analysis. To tackle this scenario, we propose to transform enabled cells with simple cells, the input of which is connected to the output of a multiplexer that has the enable signal as a select signal. An example of

III. MODEL-DRIVEN FAULT SIMULATION FRAMEWORK

The integral part of the simulation framework is the MetaFS metamodel, shown in Fig. 4. A model represents a system at a certain abstraction level. Similarly, a metamodel represents the structure of a model and the relation between the elements of a model [6]. The rootnode of the metamodel MetaFS has relations to different sets of classes, which can be categorized into three main components: Simulation Controller, Fault List, Simulation Analyzer.

Simulation Controller describes a set of classes and class attributes to define the fault simulation purpose and fault models. Simulation Controller class has three attributes: Top Module: Top module of the RTL to perform fault simulation, Simulation Time: determines for how many clock cycles to run the simulation, Timing Fault Active: defines whether timing faults are supported in order to add the extra hardware as presented in section II. This class has a composition relation to three classes: SFI, DFI, and ExhaustiveFI. These three classes narrate the most common fault simulation purposes such as Statistical Fault Injection (SFI), Direct Fault Injection (DFI), and Exhaustive Fault Injection (ExhaustiveFI).

SFI is one of the widely used techniques to determine the dependability of safety-critical designs. During an SFI campaign, only a certain random subset of all possible errors is injected. SFI class allows the user to select the total number of the injected faults via the attribute Sim Total. Fault Per Sim depicts the total number of faults to inject per single simulation, thus allowing single and multiple faults per simulation. SEU and Timing Fault are boolean attributes and, when set to True, inject only Single Event Upsets or Timing faults respectively. The fault model, injection time, and location are all randomized.

For particular applications, it is necessary to analyze the effects of the injected fault at specific locations, time and fault model. DFI class permits the user to select the certain fault model to inject via Fault Model attribute. This attribute has a relation to the Model enumeration class that supports four different fault models. ID attribute lists the set of the faults to inject per simulation, i.e. fault signals with the same ID will be injected at the same simulation. Signal Name class determines the fault location. This class has a relation to Time class that specifies the clock cycle to inject and release the fault.

Design for Test (DFT) techniques require an exhaustive fault simulation, i.e. to inject all possible stuck-at faults at all possible locations on the desired design component. ExhaustiveFI class defines attributes to enable injecting all stuck-at faults at all signals present in the fault list. Injection Time and Release Time attribute determine the clock cycle to inject and release the fault.

Fault List represents the model-based fault list description. Fault List class has a 1..* (one to many) relation to the Signal class. It means that every signal of the intended component is considered for fault simulation and will be added to the fault list. Control attribute represents the saboteur control line that manages the fault to inject and Sequential attribute determines whether the signal is the input of a register/latch, necessary information to inject SEUs. Fault Model shows all possible faults to inject at the signal.
A. Exhaustive Fault Injection

Table I shows the results of performing exhaustive fault injection into RISC-V CPU components of the CPU subsystems. Both stuck-at-0 and stuck-at-1 faults were injected at all netlist signals. Since fault collapsing is applied to the fault list, the total amount of injectable stuck-at faults is reduced.

Observations: Dhrystone application was more susceptible to stuck-at failures and a notable difference was observed while injecting faults at Execute (EX) stage. Faults injected into the fetch stage (IF) were responsible for the least percentage of failures on both subsystems.

| Component | Fault set | Total simulations | Fail | Silent | Runtime h:min |
|-----------|-----------|------------------|------|--------|---------------|
| F-ALU     | 2675      | 4713             | 45.0%| 55.0%  | 02:27         |
| F-Prefetcher | 2546    | 4547             | 55.3%| 44.5%  | 03:18         |
| F-IF stage | 2675    | 4547             | 40.3%| 59.7%  | 11:00         |
| F-EX stage | 1184    | 20897            | 25.3%| 74.7%  | 24:24         |
| D-ALU     | 2675      | 4713             | 52.1%| 47.9%  | 02:36         |
| D-Prefetcher | 2546    | 4547             | 71.6%| 28.4%  | 03:57         |
| D-IF stage | 2546    | 4547             | 23.4%| 76.6%  | 08:45         |
| D-EX stage | 12322   | 21001            | 65.9%| 34.1%  | 10:21         |

B. SEU Fault Injection

Table II presents the results of randomly injecting only SEU faults into the CPU subsystems components. Observations: Similarly, Dhrystone application is more susceptible to faults, but as can be seen from the tables, the rate of failures is fairly low for both applications.

| Component | Fault set | Total simulations | Fail | Silent | Runtime h:min |
|-----------|-----------|------------------|------|--------|---------------|
| F-ALU     | 2675      | 5000             | 14.3%| 85.6%  | 00:16         |
| F-Prefetcher | 208     | 5000             | 5.3% | 94.7%  | 01:35         |
| F-IF stage | 959     | 2000             | 1.8% | 98.2%  | 01:42         |
| F-EX stage | 208     | 5000             | 16.2%| 83.8%  | 00:13         |
| D-Prefetcher | 857     | 2000             | 6.4% | 93.5%  | 01:12         |
| D-IF stage | 926     | 2000             | 2.2% | 97.8%  | 01:32         |

C. Random timing faults

As the last experiment, we injected randomly only timing faults into the subsystems and the results are shown on Table III. Similarly, failure rate is higher for Prefetcher injected faults.

| Component | Fault set | Total simulations | Fail | Silent | Runtime h:min |
|-----------|-----------|------------------|------|--------|---------------|
| F-ALU     | 2675      | 5000             | 15.6%| 84.4%  | 10:48         |
| F-Prefetcher | 2546    | 5000             | 27.0%| 73.0%  | 08:30         |
| F-IF stage | 2546    | 5000             | 6.0% | 94.0%  | 13:11         |
| F-EX stage | 2546    | 5000             | 3.1% | 96.9%  | 08:43         |
| D-Prefetcher | 7948    | 5000             | 22.5%| 77.5%  | 09:24         |
| D-IF stage | 7948    | 5000             | 6.2% | 93.8%  | 08:43         |
| D-EX stage | 12322   | 5000             | 13.0%| 87.0%  | 09:20         |

D. General observations

The automated flow facilitated fault injection campaigns with a minimal effort, spent mostly on configuring the fault campaign attributes. Moreover, [7] shows that the existing fault injection flow introduces an overhead of 16%-99% of overall runtime compared to the original design, but an increase of
3.5x-8.4x in the fault simulation performance compared to full GL simulation. IF stage was the component with most silent faults due to the fact that one of the biggest modules of IF stage is the Exception Unit. Since no exception was detected during the applications run, the fault injected into it had no real effect on the designs. Noticeably, Prefetcher had the highest rate of failure on every fault simulation campaign, because it is the component that controls the PC. The failure rate of the applications is low because the random interval (10k clock cycles) of instruction sequences was repeating many series of instructions due to software loops. Timing faults introduced an overhead of 3-16x slower, considering the increased design area due to added registers.

V. RELATED WORK

Pravadelli et al. [9] present an automated non-intrusive simulation-based fault injection framework based on QEMU, an emulator for microprocessor architectures. The authors injected faults into the processor by masking data structures inside QEMU to mimic the behavior of the faults. The framework supports stuck-at faults, timing faults, as well as bit-flips, and experiments were conducted in x86 and ARM processors. This approach provides a fast simulation flow for CPU designs but could lack some accuracy compared to cycle-accurate fault injection and further estimations are needed. An approach to inject faults on microarchitectural simulators is presented in [11]. Authors have extended MARSS and Gem5 simulators to support fault injection (namely MaFIN and GeFIN respectively). A Fault Mask Generator is implemented that can produce a random set of fault masks for different types of faults such as bitflips, permanent and intermittent faults. An Injection Campaign Controller reads the masks and sends injection requests to the Injector Dispatcher, a module that communicates directly with the simulators. Experiments were run on x86 and ARM processors, and the technique is only suitable for processor fault simulation and further extensions are required for non-core components. In [12] authors propose a framework for Verilog-based fault injection based on Verilog Programming Interface (VPI). A single system function allows the user to determine the fault type, location, and duration of the fault. With the help of callback functions, i.e. using VPI to control the Verilog simulator, the selected fault is injected. The technique can only be applied to Verilog and VPI-compliant simulators. [4], [3], [10] modify the VHDL description of the design with the help of saboteurs and mutants. These techniques provide a high degree of controllability but are limited to RTL and VHDL description language. A fault injection tool, SINJECT, based on the synthesizability of HDL models is proposed in [16]. The tool provides injection of bitflips and permanent faults into the Verilog and VHDL models. The authors ran experiments on two different small processors, ARP and DP32. LIFTING [1] is an open source fault simulator capable of injecting single/multiple stuck-at faults and SEUs in Verilog designs. Each circuit gate is modelled as a class, and with the help of virtual methods, the tool forces specific inputs to the intended value to model the selected fault.

VI. CONCLUSION

We introduced in this paper a fully automated model-based in-house fault simulation framework, namely MetaFS, supporting different fault models. Furthermore, MetaFS enables running automatically various fault injection campaigns such as EFI, SFI, and DFI. Numerous experiments were conducted on different components of two CPU subsystems. Only 1 person-day was required to run 22 distinct fault injection campaigns. Additionally, Prefetcher component is most susceptible to failures, and protection mechanisms are recommended to be integrated into this component. Timing faults introduced an overhead of 3-16x due to the increased design area. Reduction of timing faults overhead by dynamically modifying the saboteurs, applying fault simulation to other subsystems such as accelerators and GPUs, running various safety-critical applications and identifying components that are prone to failures are subject to future work. Further comparisons to other commercial and open-source fault simulation tools are subject to future work as well.

VII. ACKNOWLEDGEMENTS

Part of the work has been performed in the project ArchitectureEUCA2030 under grant agreement No 877539. The project is co-funded by grants from Germany, Netherlands, Czech Republic, Austria, Norway, France and Electronic Component Systems for European Leadership Joint Undertaking (ECSL JU). Part of the work described herein is also funded by the German Federal Ministry of Education and Research (BMBF) as part of the research project Scale4Edge (16ME0122K).

REFERENCES

[1] A. Bosio and G. Di Natale. Lifting: A flexible open-source fault simulator. In 2008 17th Asian Test Symposium, pages 35–40, 2008.
[2] J. Arlat, Y. Crouzet, and J.-C. Laprie. Fault injection for dependency validation of fault-tolerant computing systems. In The 19th ISFICT, pages 348–355, 1989.
[3] J.C. Baraza, J. Gracia, D. Gil, and P.J. Gil. Improvement of fault injection techniques based on vhdl code modification. In 10th IEEE International HLDVT, pages 19–26, 2005.
[4] Juan-Carlos Baraza, Joaquín Gracia, Sara Blanc, Daniel Gil, and Pedro J. Gil. Enhancement of fault injection techniques based on the modification of vhdl code. IEEE Transactions on VLSI Systems, 16(6):693–706, 2008.
[5] R.C. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. IEEE Transactions on Device and Materials Reliability, 5(3):305–316, 2005.
[6] W. Ecker and J. Schreiner. Metamodelling and Code Generation in the Springer Science+Business Media Dordrecht, pages 1–41, 01 2016.
[7] Endri Kaja et al. Towards fault simulation at mixed register-transfer/gate-level models. In IEEE International Symposium on DFT in VLSI and Nanotechnology Systems, pages 1–6, 2021.
[8] Oettjens et al. Safety simulation of automotive electronics using virtual prototypes: State of the art and research challenges. In 51st ACM/EDAC/IEEE Design Automation Conference, pages 1–6, 2014.
[9] D. Ferraretto and G. Pravadelli. Efficient fault injection in qemu. In 2015 16th Latin-American Test Symposium (LATES), pages 1–6, 2015.
[10] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson. Fault injection into vhdl models: the mefisto tool. In Proceedings of IEEE 24th International Symposium on Fault-Tolerant Computing, pages 66–75.
[11] M. Kiliorakis, S. Tselonis, A. Chatzidimitriou, N. Foutris, and D. Gizopoulos. Differential fault injection on microarchitectural simulators. In 2015 IEEE International Symposium on Workload Characterization.
[12] D. Kammuser, J. Guan, G. Ascheid, R. Leupers, and H. Meyr. A fast and flexible platform for fault injection and evaluation in verilog-based simulations. In 2009 Third IEEE International Conference on Secure Software Integration and Reliability Improvement, pages 309–314, 2009.
[13] G.M. Luong and D.M.H. Walker. Test generation for global delay faults. In Proceedings International Test Conference 1996. Test and Design Validation, pages 433–442, 1996.
[14] L. M. Luzza, A. Rusopo, A. Bosio, E. Sanchez, and L. Dilillo. A model-based framework to assess the reliability of safety-critical applications. In 2021 24th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), pages 41–44, 2021.
[15] E. Schreiner, R. Findenig, and W. Ecker. Design Centric Modeling of Digital Hardware. In IEEE International HLDVT 2016, pages 46–52.
[16] H.R. Zarandi, S.G. Miremadi, and A. Ejlali. Dependability analysis using a fault injection tool based on synthesizability of hdl models. In Proceedings 18th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems, pages 485–492, 2003.
[17] H. Ziaede, R. Ayoubi, and R. Velazco. A survey on fault injection techniques. Int. Arab J. Inf. Technol., 1:171–186, 01 2004.