ETISS-ML: A Multi-Level Instruction Set Simulator with RTL-level Fault Injection Support for the Evaluation of Cross-Layer Resiliency Techniques

Daniel Mueller-Gritschneder*, Martin Dittrich*, Josef Weinzierl*,
Eric Cheng†, Subhasish Mitra† and Ulf Schlichtmann*
*Chair of Electronic Design Automation, Technical University of Munich, Germany, Email: daniel.mueller@tum.de
†Stanford University, USA, Email: eccheng@stanford.edu

***Abstract*—ETISS is an instruction set simulator (ISS) for Virtual Prototypes (VPs) modeled with SystemC/TLM. In this paper, we propose the extension ETISS-ML, which enables a multi-level simulation that switches between ISS-level and register transfer level (RTL) to accurately evaluate the impact of soft errors in the pipeline of a RISC processor. ETISS-ML achieves close-to-RTL-accurate fault injection simulation results with close-to-ISS simulation performance with a speed up gain up to 100x compared to RTL. For this, we propose an approach to dynamically determine the length of the RTL simulation period. The high simulation performance of ETISS-ML enables an ultra-efficient and accurate evaluation of cross-layer resiliency techniques for embedded applications, which requires running a large number of fault injections for long simulation scenarios. This is demonstrated on a case study of a Microcontroller Unit (MCU) executing a control algorithm for adaptive cruise control.**

I. Introduction

Two major concerns need to be addressed in the design of embedded systems for safety-critical applications. Firstly, the safety of the intended functionality (SoiF) must be guaranteed. This relates to the ability of the system to correctly comprehend its environment and behave safely. Secondly and not less important, the system must also behave safely in the presence of random HW faults. For this, the system must be able to detect and handle random HW errors in order to prevent system failures [9].

In this paper we will focus on the impact of radiation-induced soft errors in the embedded processor of a Micro-Controller Unit (MCU). MCUs are common computing platforms, e.g, for many automotive and industrial applications. Soft errors can be modeled as bit flips in the flip-flops (FFs) of the processor and their impact can be evaluated by simulation at Register Transfer Level (RTL) using fault injection (FI). In contrast to benchmark programs, the evaluation of embedded applications may require long simulation scenarios as the fault may cause a system failure long after its occurrence. As an example, the settling time of a control algorithm can be anything from μs to s real-time. A large number of RTL FI simulations for scenarios in the seconds-range takes prohibitively too long.

In order to overcome this challenge, we propose a highly performance-optimized FI simulation environment. We use the Extendable Translating Instruction Set Simulator (ETISS) [8], which is based on fast dynamic binary translation (DBT), for modeling processors as part of a SystemC Virtual Prototype (VP). In order to achieve high accuracy for resiliency evaluation, our extension ETISS-ML applies fast multi-level simulation to enable RTL-level FI Support.

As contributions of this paper, we show (1) a method to dynamically determine the length of the RTL simulation phase, which improves simulation speed, (2) a method for micro-architectural error tracking at the pipeline outputs, which can be used to analyze how soft errors inside the pipeline impact the pipeline's outputs, (3) a brief case study of an adaptive cruise control (ACC) system to demonstrate how ETISS-ML can be applied to evaluate cross-layer resiliency techniques [2] for embedded applications as part of a full-system VP.

In the experimental results, we show that ETISS-ML has close-to-RTL accuracy. For long simulation scenarios in the range of seconds, ETISS-ML achieves close-to-ISS simulation performance, which is 100x faster than RTL simulation.

II. Related Work

The work in [12] already proposed the idea of multi-level simulation for faster FI using a functional and RTL model of a super-scalar processor. Yet in [12], a fixed-length RTL cool-down phase is used. We increase simulation speed and accuracy by proposing a simulation setup with two pipelines to dynamically determine the length of the RTL cool-down. The work in [7] also uses multi-level simulation referred to as hierarchical simulation between RTL and gate level. The focus of this work lies on capturing stuck-at faults. Permanent faults change the functionality of the system permanently and, hence, simulation methodology differs from our work on transient soft errors. FI results presented for example in [3] showed that ISS are not sufficiently accurate for resiliency evaluation of soft errors. We support these findings in great detail by tracking the error patterns at the pipeline outputs.

In terms of FI and evaluation of cross-layer resiliency, there has been much work, e.g., [6], [3], [4], [2], using either benchmarks or single-level simulation for resiliency evaluation. The authors in [5] propose to correlate RTL and instruction level fault injection scenarios for automotive processors to propagate information up to ISS-level. In contrast, in our work, we do not try to accelerate injection on a single level or propagate information up the levels. Instead, we switch processor models during simulation. Work in [13] also uses full system simulation with SystemC but focuses on memory errors, which can be represented at ISS-level. ETISS-ML focuses on handling micro-architectural registers.

This work was partially supported by the BMBF in the project EffektiV (01IS13022) and by the DFG SPP 1500 Dependable Embedded Systems.

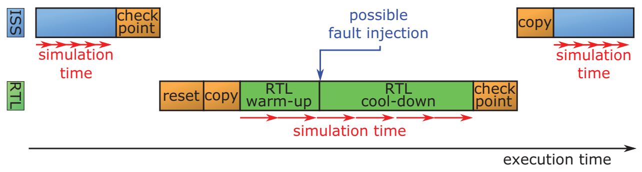

Fig. 1. Multi-level simulation flow

III. ETISS-ML Approach

The simulation process of the multi-level simulation with ETISS-ML is depicted in Fig. 1. In this section, we will describe each stage of the multi-level simulation and FI process in detail.

ISS start mode: The multi-level simulation starts in fast ISS mode.

Switch from ISS to RTL mode: At the point of interest, the system switches to RTL mode. First, a checkpoint of the architectural state of the processor is created from the ISS model. This checkpoint, which captures the state of all architectural registers of the processor, is copied to the RTL model. The copying is done by writing FF values inside the RTL and executing a set of MOV instructions. The copy mechanism requires to have a mapping between the architectural state within the ISS and the FFs of the RTL implementation of the processor.

RTL Warm-up: Many FFs of the RTL implementation of the processor cannot be obtained from the ISS checkpoint. In the following, we will refer to these FFs as micro-architectural FFs. These FFs include pipeline buffers in the data path as well as configuration and state FFs in the control path. For RTL warm-up, the micro-architectural FFs are first set to their reset values and then program execution is resumed after the last instruction executed in ISS mode. The values of the micro-architecture FFs are overwritten as the pipeline warms up. There is no 100% guarantee that the processor takes the same state as the processor would have taken with a pure RTL simulation without ISS phase. Yet, as we show in the experimental results, after a warm-up phase executing a certain number of instructions, no difference in state is observed for the RISC processor studied.

Fault Injection: We focus on radiation-induced soft errors. These can be modeled as a bit flip in a single FF of the processor [3]. The bit flip is applied to an FF inside the RTL processor model after the RTL warm-up phase.

RTL cool-down: The impact of the soft error is simulated accurately in the RTL phase. As a major feature, ETISS-ML dynamically determines the required length of the RTL cool-down as will be detailed in Sec. III-A.

Switch from RTL to ISS mode: When the RTL processor fetches an instruction, it requires several cycles to execute inside the pipeline. Hence, after RTL cool-down NOP instructions need to be pushed into the pipeline to clear in-flight instructions and obtain a defined program point to create an RTL checkpoint.

ISS finish mode: After the RTL checkpoint is copied to the ISS, the simulation can continue at ISS level.

A. *Dynamic RTL Cool-down Length*

During the switch from RTL to ISS, all micro-architectural FF states are lost. Hence, one needs to ensure that this switch does not lead to a wrong evaluation of the fault impact. In order to track faulty FF values, two models of the pipeline are simulated in parallel with individual copies of all internal FFs as is illustrated in Fig. 2 for a standard five-stage RISC pipeline. We determine the RTL cool-down length as follows:
(1) The fault (bit flip) is injected into the FF copy in the faulty pipeline (FP). In contrast, the tracking pipeline (TP) simulates without FI.
(2) Whenever the FP and TP write different values to their outputs, only the corrupted value of the FP is committed. In this case the external state is marked as corrupted.
(3) Whenever the FP and TP read values at their inputs, both read the external state (possibly) corrupted by the committed outputs from the FP.
(4) We regularly check (e.g. every 100 cycles) the condition whether the micro-architectural FFs of the FP and TP have exactly the same state or whether a maximal pre-defined RTL cool down length is reached. If this condition is met, three cases can be differentiated:
In the first case, the FP and the TP have the same FF states and the external state is not marked as corrupted. In this case the simulation can be terminated because the fault is masked within the pipeline before propagating to any output.
In the second case, the FP and the TP have the same FF states and the external state is marked as corrupted. In this case we can stop RTL-cool down and switch to ISS-level because the fault fully propagated out of the FP. The FP state may still be different from a fault-free run, but these differences are not due to the direct impact of the original injected soft error. Rather, they are caused by inputs from the corrupted external state. This is observable because the TP has the same FF states but was only fed with the corrupted external state, not with the initial soft error. There is no loss in accuracy compared to a pure RTL simulation since the corrupted external state is fully captured within the ISS simulation.
In the third case, the maximal RTL cool-down length is reached because the fault is neither masked nor propagates out of the FP. This can be caused, e.g., by the corruption of a configuration FF that is never re-configured in the software (maybe only during system boot or reset). In this case one can either simulate to the end at RTL or stop and log this run as a possible system failure since it causes a long-term modification of the pipeline. A fault impact, which is also frequently observed, is a pipeline deadlock. As the pipeline stops, this fault never propagates out of the FP when evaluating an unprotected system. Yet, a common feature for MCUs is a Safety Unit that resets the processors whenever an error is detected. After such a reset, the FP and TP have the same reset state. Hence after such a reset, the switch to ISS-mode can be performed without loss in accuracy. This fault impact can be exploited also for higher simulation speed by adding an additional FP deadlock detection. In case of a

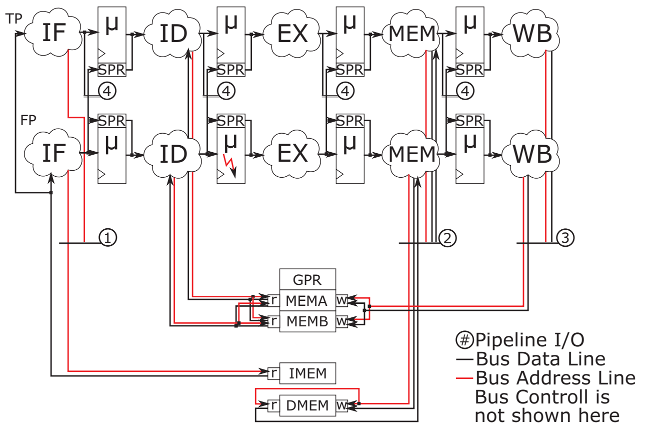

Fig. 2. Tracking pipeline (TP) and Faulty Pipeline (FP)

detected deadlock in the FP, the simulation is continued in fast-forward mode, where the processor continuously stalls unless it receives an external reset.

B. Micro-architectural Error Tracking

The FP/TP setup from Fig. 2 can be exploited to classify the impact of soft errors at micro-architectural level. The pipeline outputs are, as depicted in Fig. 2, all instruction fetch bus signals (1), the data bus signals (2) and the signals for write back to the register file (3) and special function registers (4). The possible FI points at ISS-level are covered by these micro-architectural pipeline outputs (register file errors by write back errors, instruction word errors by instruction bus signal errors etc.). At ISS-level, usually single-bit FI is used to avoid the explosion of possible multi-bit combinations.
To compare this to RTL simulation, the number of outputs corrupted by the soft error at micro-architectural level is logged by comparing FP to TP outputs. As will be shown in the experimental results, faults in the micro-architectural FFs may lead to the corruption of various bits or various numbers of outputs. Hence, one should be aware that single-bit FI at ISS-level does not cover the impact of all soft errors in the micro-architecture, which frequently cause multi-bit corruptions.

IV. ETISS-ML Implementation

We build ETISS-ML upon an existing ISS named ETISS [8]. Its main use case is as processor model within a virtual prototype based on SystemC/TLM. Using ETISS-ML with an existing multi-level processor model is a simple task. Setting up the multi-level simulation for a new processor is a more laborious task, so we developed automation support. As ETISS is written in C++ for easy SystemC integration, one needs to obtain a C++ RTL model of the processor. We used the tool Verilator [11]. It translates Verilog designs to high performance C++ simulation code. In order to set up the RTL fault simulation with two pipelines, we wrote a CLANG-based tool that automatically translates the verilated C++ RTL (vRTL) to an C++ RTL with both FP and TP. It also adds FI capabilities to all FFs.

TABLE I
Impact of single soft error in pipeline FF on pipeline outputs

	Masked	No influence	single-bit	multi-bit 1 output bus	multi-bit 2-4 out. buses	multi-bit 5-10 out. buses	multi-bit >10 out. buses
AES	40508	5944	18291	869	1090	1196	2102
EDGE	39756	7859	17311	642	1021	1401	2010
FIR	41882	6135	16972	486	885	1498	2142
HEAP	41759	5795	17334	545	907	1535	2123
IIR	40831	7387	16681	648	984	1425	2044
JDCT	41457	5280	17976	652	1052	1648	1935
Covered by ISS FI	no	no	yes	no	no	no	no

V. Experimental Results

The multi-level simulation was implemented for the Open-RISC OR1200 processor [1]. The RTL implementation uses only the pipeline, programmable interrupt controller and timer but no caches in order to have a MCU-type processor similar to ARM's Cortex M family. We select 100MHz as frequency. The processor has 1440 micro-architectural FFs as part of the pipeline, exception unit, interrupt controller and timer, which we use as targets for FI. We do not include the register file of the processor for FI since this is considered an external state also available at ISS level. FI experiments were executed on a LINUX compute cluster comprised of different Xeon-based Intel servers from generations from 2009 to 2015 and Intel PCs from 2013 to 2017.

A. Error Impact at Pipeline Outputs

Table I shows the impact of a single soft error in a micro-architectural FF on the pipeline outputs during execution of six benchmark programs. Firstly, many faults are already masked within the pipeline. Some do not propagate to the outputs (no influence). Then a high number of soft errors lead to a single wrong bit. Yet, beyond this, there are many soft errors that lead to multiple bits of one bus or multiple output buses being corrupted, or the same output bus being corrupted multiple cycles (this is not differentiated in the table). Such error impacts would not be covered by a single-bit ISS FI.

B. ETISS-ML Accuracy and Speed-up

Table II shows the simulation performance averaged over 10k FI scenarios for RTL FI using Verilator as simulator (vRTL), for multi-level FI using a fixed cool down length (ML fixed CDL) as proposed in [12], and multi-level FI with dynamic cool-down length (ML dyn. CDL). For the five benchmark programs the fixed cool-down length (CDL) is set to 10k. For dyn. CDL, the maximal cool-down length is also set to 10k for same accuracy. Dyn. CDL gives speed-ups between 3.2x and 9.7x compared to the fixed CDL. One can observe that both ML FIs are slower compared to vRTL FI for the first three small benchmarks due to the additional overheads of the transformed verilated RTL code with error tracking used in RTL mode of ML. These overheads compensate the speed-up obtained from ISS-mode. However, for the two longer benchmarks, significant speed-up compared to vRTL is obtained. In terms of accuracy, no difference in

TABLE II
RTL VS. MULTI-LEVEL (ML) SIMULATION PERFORMANCE
AVERAGED OVER 10K FIS, *VALUE ESTIMATED

	Instructions	vRTL Avg. Dur. [s]	vRTL Avg. Perf.[MIPS]	ML Fixed CDL Avg. dur. [s]	ML Fixed CDL Avg perf. [MIPS]	ML dyn. CDL Avg. dur. [s]	ML dyn. CDL Avg perf. [MIPS]	Speedup vs. vRTL	Speedup vs. ML fixed CDL
Fixed CDL 10k cycles, Max dyn. CDL 10k cycles									
JDCT	22E3	0.1	0.18	4.41	0.005	1.34	0.016	0.09x	3.2x
AES	57E3	0.30	0.19	5.29	0.011	1.28	0.044	0.23x	4.0x
IIR	69E3	0.38	0.18	5.03	0.014	1.05	0.066	0.37x	4.7x
EDGE	305E3	1.71	0.18	5.18	0.059	1.09	0.28	1.6x	4.7x
ISORT	2.9E6	15.3	0.19	5.42	0.18	1.67	1.74	9.2x	9.7x
Fixed CDL 500k cycles, Max dyn. CDL 500k cycles									
ACC Sys	2E9	10k*	0.2*	360	5.54	97.5	**20.5**	100x	3.6x

the outcome of the FI simulation are observed. Both ML simulation setups obtained RTL-accuracy for all benchmarks. For the ACC System, fixed and max. dyn. CDL is set to 500k so that reset error handlers are able to trigger before max. CDL is reached. For the long FI scenarios of 20s, dyn. CDL ML obtains more than 20 MIPS. This is a factor of 100x faster than the vRTL simulation. Here we estimated the runtime for the vRTL simulation by assuming 0.2 MIPS performance. The estimate is slightly optimistic, because vRTL shows less than 0.2 MIPS for all other test-cases.

C. Case Study for ACC System

We study a SystemC VP that models an MCU used in an adaptive cruise control (ACC) system. Its goal is to maintain a constant distance between two moving vehicles by controlling the speed of the rear vehicle via the throttle value of the motor (actuator). The processor of the MCU periodically executes a PI control algorithm. The PI's inputs are sensor values measuring the distance to the front vehicle and speed of the rear vehicle. The sensor values are dynamically generated by a physics simulation of the two vehicles based on the commands sent to the actuator. We define a simple safety specification to demonstrate the evaluation. The desired distance between the vehicles is set to 40m. A fault is classified to cause a system-level failure when the distance leaves the corridor between 20m and 60m within a given driving scenario. For this scenario, both vehicles have same speed and a distance of 50m at time zero. After 10s the front vehicle brakes. The rear vehicle has to react to the braking front vehicle.

In order to test cross-layer resiliency, we apply the following error detection and handling mechanisms:

Memory Protection (MP) with Reset: Whenever an access on instruction or data bus to an illegal (unmapped) address is detected, the system is reset.

Watchdog Timer (WDT) with Reset: The control algorithm has to write a value to the actuator every 10ms. If no actuator write is detected within 20ms, the system is reset by the WDT.

Error Detection by Duplicated Instructions (EDDI) with Reset: Instruction duplication is applied by the compiler to detect silent data corruption [10]. When EDDI detects a fault, the SW triggers a reset.

Each method comes with a certain overhead. MP and WDT require additional area but are usually available on modern MCUs. EDDI increases the execution time of the task at the cost of idle time of the processor. Additionally, this increases the delay between reading the sensor values and writing the actuator value.

TABLE III
FAILURE RATE (72K FIS PER COMBINATION)

	Error detected			Error not detected			
	Fail	Dist.	No infl.	Fail	Dist.	No infl.	Σ Fail
No Protection	0	0	0	4428	842	66730	4428
	0.0%	0.0%	0.0%	6.2%	1.2%	92.7%	6.3%
MP	0	1149	1061	2032	821	66937	2032
	0.0%	1.6%	1.5%	2.8%	1.1%	93.0%	2.8%
WDT	3	3805	1803	90	594	66806	93
	0.0%	3.8%	2.5%	0.1%	0.8%	92.8%	0.1%
EDDI, MP, WDT	0	3601	1909	16	97	66377	16
	0.0%	5.0%	2.7%	0.0%	0.1%	92.2%	0.0%

Table III shows the number of simulation outcomes with failures, disturbances and no influence for detected and undetected errors. Disturbances are noticeable deviations form the fault-free reference run either caused by a fault effect or a error handler such as the system reset. Activating MP and WDT significantly reduces the failure rate, especially the WDT. Moving to a cross-layer approach, a combination of EDDI, MP and WDT again significantly improves the failure rate of the system. Overall, the case study shows that with system-level evaluation using VPs and ETISS-ML, one can gain important insights on how to apply cross-layer resilience methods early in the design flow.

VI. CONCLUSIONS

ETISS-ML allows almost RTL-accurate FI simulation of soft errors in RISC pipelines with almost ISS-level simulation performance with a speed up of about 100x compared to RTL.

REFERENCES

[1] Openrisc 1000 architecture manual v1.1. http://opencores.org, 2014.

[2] E. Cheng, S. Mirkhani, et al. CLEAR: cross-layer exploration for architecting resilience: Combining hardware and software techniques to tolerate soft errors in processor cores. In *DAC*, 2016.

[3] H. Cho, S. Mirkhani, et al. Quantitative evaluation of soft error injection techniques for robust system design. In *DAC*, 2013.

[4] M. Ebrahimi, N. Sayed, et al. Fault injection acceleration by architectural importance sampling. In *CODES/ISSS*, 2015.

[5] J. Espinosa, C. Hernandez, et al. Analysis and RTL correlation of instruction set simulators for automotive microcontroller robustness verification. In *DAC*, 2015.

[6] A. Höller, G. Macher, et al. A virtual fault injection framework for reliability-aware software development. In *Dependable Systems and Networks Workshops*, 2015.

[7] M. L. Li, P. Ramachandran, et al. Accurate microarchitecture-level fault modeling for studying hardware faults. In *HPCA*, 2009.

[8] D. Mueller-Gritschneder, M. Dittrich, et al. The extendable translating instruction set simulator (ETISS) interlinked with an MDA framework for fast RISC prototyping. In *RSP*, 2017.

[9] J.-H. Oetjens, N. Bannow, et al. Safety evaluation of automotive electronics using virtual prototypes: State of the art and research challenges. In *DAC*, 2014.

[10] N. Oh, P. P. Shirvani, and E. J. McCluskey. Error detection by duplicated instructions in super-scalar processors. *IEEE Trans. on Reliability*, 2002.

[11] W. Snyder. Introduction to verilator. veripool.org, 2017.

[12] N. J. Wang, A. Mahesri, et al. Examining ace analysis reliability estimates using fault-injection. In *ISCA*, 2007.

[13] N. Wehn, U. Schlichtmann, et al. A cross-layer technology-based study of how memory errors impact system resilience. *IEEE Micro*, 2013.