Techniques and Tools for Fast Fault Injection Simulations of RISC-V Processors at RTL

Johannes Geier[1], Daniel Mueller-Gritschneder[2], and Ulf Schlichtmann[1] *

[1]Chair of Electronic Design Automation, Technical University of Munich
[2]Institute of Computer Engineering, TU Wien

Abstract

With an ever-increasing utilization of open-source hardware, the demand for improved productivity in its various development phases rises. This demand can only be met by open-source tools accompanying these open-hardware projects to cover their crucial design, testing, and verification tasks. One of these required tasks is fault injection analysis. At the early stages of development, it can help to identify vulnerabilities in the hardware designs. However, simulating hardware can be cumbersome, which makes minimizing the effort needed to conduct a large-scale fault injection campaign a significant goal. This work leverages the features of an existing open-source fault injection tool to automatically add masking check support to the simulation. With these checks, a simulation framework can detect experiments where an injected transient fault would not have any impact, i.e., masked.

Introduction

The authors of [1] have shown that low-level Fault Injection (FI) simulation can be utilized to evaluate the soft error resiliency of digital systems. Furthermore, they showed that w.r.t. to accuracy, these simulations perform better than ones conducted on higher levels of abstraction, such as faults in memory or instruction set architecture. The high cost of low-level simulations due to computational effort and time can be mitigated by concurrently deploying individual experiments of a FI campaign. Works such as [2] leverage Checkpoint Restore Boot (CRB), where booting the simulation from pre-recorded states (checkpoints) of a fault-free run removes the simulation period before a particular point of interest.In statistical FI [3] with uniformly distributed samples, CRB can reduce the overall simulation effort by about 50%. However, besides this *warmup* acceleration method, the remainder of the simulation time has to be evaluated to observe the fault's impact (*cooldown*).

Techniques and Tools

To implement the CRB, we use Verilator's built-in *save/restore* feature. This enables serialization (checkpoint store) and de-serialization (restore) of a current simulation state of *vRTL* to a binary file. Another effective acceleration method is *masking checks*, which identifies and terminates experiments during *cooldown* that would not impact a faulted system's behavior. This work introduces an extension to our open-source tool vRTLmod [4] that allows *masking checks*. The tool vRTLmod takes Verilator [5] output and automatically adds FI capability at *flip-flop*-level comparable

*This work has been developed in the project MANNHEIM-FlexKI funded by the German Federal Ministry of Education and Research (BMBF) under contract no.01IS22086L.

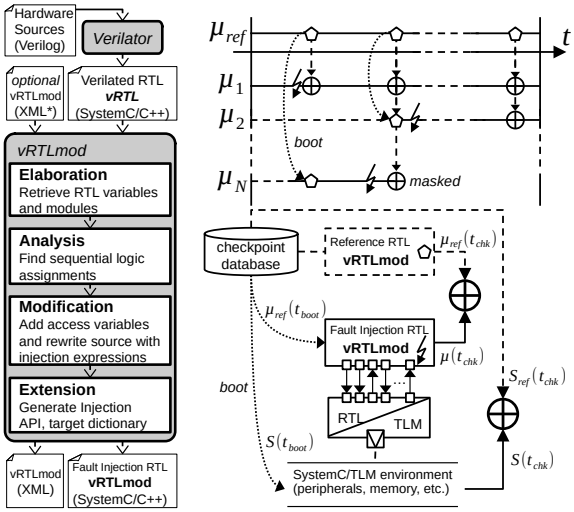

Figure 1: *vRTLmod transformation (left) and usage in a SystemC-based Virtual Prototype (right).*

to the classification presented in [1]. This is achieved via source-code transformation with LLVM's Clang [6]. Fig. 1 shows the [4] transformation (left), which can be divided into four steps: Elaboration, Analysis, Modification, and Extension. Elaboration and Analysis identify the input's structure and sequential elements (e.g., flip-flops). Modification alters the C++ code base for injection statements and adds accessor members in headers. *Masking check* support is added to the last stage (4), Wrapping and Extension. It generates a differential module (DIFF-API) that takes two references to the original fault injectable Verilated Register Tranfer Level (RTL) (vRTL). DIFF-API provides a bit-wise XOR ($\oplus$) between each module's sequential states μ. This can generate the set of differing states δ at a point in time t per Eq. (1). An empty set δ means the fault is masked in micro-architecture:

$$\delta(t) = \mu(t) \oplus \mu_{ref}(t) \stackrel{?}{=} \emptyset \quad (1)$$

Table 1: *Average simulation time $\overline{t_s}$ and classification of experiments conducted on differently configured RTL FI. Checkpoint interval of 10,000 clock cycles (CCs).* 11,000 *equal experiments per benchmark and configuration combination.*

benchmark	CC[K]	config	$\overline{t_s}$	save[%]	MSK	[%]	DUE	[%]	SDC	[%]	pLAT	[%]
aha-mont64	18.5	RTL	0.75	-	0	0.00	199	1.81	567	5.15	10,234	93.0
aha-mont64	18.5	CRB	0.56	25.3	0	0.00	199	1.81	567	5.15	10,234	93.0
aha-mont64	18.5	CMSK	0.48	36.0	2,821	25.65	199	1.81	567	5.15	7,413	67.4
huffbench	276	RTL	10.9	-	0	0.00	478	4.35	223	2.03	10,299	93.6
huffbench	276	CRB	5.91	45.8	0	0.00	478	4.35	223	2.03	10,299	93.6
huffbench	276	CMSK	3.62	66.8	4,875	44.32	478	4.35	223	2.03	5,424	49.3
picojpeg	720	RTL	28.1	-	0	0.00	248	2.25	71	0.65	10,681	97.1
picojpeg	720	CRB	13.8	50.9	0	0.00	248	2.25	71	0.65	10,681	97.1
picojpeg	720	CMSK	7.25	74.2	5,592	50.84	248	2.25	71	0.65	5,089	46.3

As per Fig. 1, halting the simulation at a checkpoint post-FI (t_{chk}) allows us to restore a simulation state from a checkpoint database via CRB to an inactive reference core, which gives us $\mu_{ref}(t_{chk})$.

Evaluation

We evaluate the performance and accuracy against a plain RTL simulation, a CRB version, and a version that adds *masking checks* at checkpoints in the *cooldown* phase to the CRB (CMSK). As Device Under Test (DUT), we choose the open-source RISC-V processor cv32e40p [7] attached to a minimal System on Chip (SoC) modeled in SystemC, which contains memory and peripherals. The right-hand side of Fig. 1 depicts this Virtual Prototype (VP). Note that the system part also has a differential check since between diffs of the DUT, a fault could propagate out of the observable RTL. This would not be necessary if the *verilated* hardware contained the testbench. The checkpoints are generated for an interval of 10,000 CCs starting at $c = 10,000$; e.g., aha-mont64 has only one checkpoint. The DUT executes a set of Embench™ benchmarks [8]. All three simulator configurations conduct the same set of unique single-bit, single-cycle fault experiments per benchmark program. Tab. 1 reports the average simulation time per experiment, as well as the rates of Silent Data Corruptions (SDCs), Detectable Unrecoverable Errors (dues), and possibly Latent Faults (pLATs). We classify an experiment as due when the RISC-V or a watchdog-timer (after 2x expected CC) trips an exception, as an SDC when the benchmark has erroneous output, and as *possibly latent* if neither SDC or due was classified. pLAT contains all experiments that could not be terminated early or did not affect the benchmark's output. However, this does not mean that DUT has not been corrupted.

Results and Conclusion

For pure Register Tranfer Level (RTL) and Checkpoint Restore Boot (CRB), the number of experiments classified as possibly Latent Fault (pLAT) is higher than for the ones featuring *masking checks* simply because there is no such feature; thus, there is no way to determine a corrupted state. For longer benchmarks, such as *picojpeg* and *huffbench* with around 720K and 276K CCs, respectively, CRB reduces the average simulation time per experiment by around 50%. Adding our *masking checks* can save an additional 23.3%. All simulation results match for equal sets of experiments, i.e., injection bit and time: The techniques are RTL accurate. As we have shown in this work, *masking checks* are complementary to existing techniques such that we have added this feature to vRTLmod on GitHub[1].

References

[1] H. Cho et al. "Quantitative Evaluation of Soft Error Injection Techniques for Robust System Design". In: *Proceedings of the 50th Annual Design Automation Conference*. DAC '13. Austin, Texas: Association for Computing Machinery, 2013. DOI: 10.1145/2463209.2488859.

[2] B. Tabacaru et al. "Efficient Checkpointing-Based Safety-Verification Flow Using Compiled-Code Simulation". In: *2016 Euromicro Conference on Digital System Design (DSD)*, pp. 364–371. DOI: 10.1109/DSD.2016.14.

[3] R. Leveugle et al. "Statistical Fault Injection: Quantified Error and Confidence". In: *2009 Design, Automation & Test in Europe Conference & Exhibition*. 2009, pp. 502–506. DOI: 10.1109/DATE.2009.5090716.

[4] J. Geier and D. Mueller-Gritschneder. "vRTLmod: An LLVM based Open-source Tool to Enable Fault Injection in Verilator RTL Simulations". In: *Proceedings of the 20th ACM International Conference on Computing Frontiers*. 2023, pp. 387–388. DOI: 10.1145/3587135.3591435.

[5] W. Snyder et al. *Verilator*. URL: https://github.com/verilator/verilator.

[6] C. Lattner and V. Adve. "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation". In: *Intern. Symposium on Code Generation and Optimization, 2004.* 2004, pp. 75–86. DOI: 10.1109/CGO.2004.1281665.

[7] M. Gautschi et al. "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.10 (2017). DOI: 10.1109/TVLSI.2017.2654506.

[8] Embench™. 2024. URL: https://github.com/embench.

[1] https://github.com/tum-ei-eda/vrtlmod