Safety-Verification Flow Sporting Gate-Level Accuracy and Near Virtual-Prototype Speed

Bogdan-Andrei Tabacaru*†, Moomen Chaari*†, Wolfgang Ecker*†,
Thomas Kruse*, and Cristiano Novello*
*Infineon Technologies AG - 85579 Neubiberg, Germany
†Technische Universität München
Firstname.Lastname@infineon.com

***Abstract*—Efficient fault-effect analysis on gate-level (GL) and RTL models has become increasingly costly due to rising complexity in today's SoCs. Although transaction-level (TL) models speed-up simulations, faults injected therein correlate poorly to faults from GL models. This leads to injection of false faults and over-engineering of safety mechanisms. In this paper, we propose a fault-injection methodology for TL models maintaining 100% correlation to GL models. Our approach's key aspects are: matching points across abstraction levels and selective abstraction of GL functionality using compiled-code simulation techniques. Measurements show an average 65-70x speed-up over RTL models and three orders of magnitude speed-up over GL models. Moreover, speed-up increases with the size of analyzed designs.**

Keywords—Automotive; Fault Injection; ISO 26262; Safety Verification; SystemC; TLM; Virtual Prototyping

I. Introduction

Fault-effect simulation of digital circuits is a lasting problem in areas such as safety verification. The current trend of increasing complexity affects not only the development of systems on chip (SoCs) but also slows down fault injection and simulation campaigns as part of safety verification.

Presently, fault injection is mainly performed on register-transfer level (RTL) models (mostly transient faults) and gate-level (GL) net-lists (stuck-at faults). Parts 5 and 10 of ISO 26262, the standard for safety-critical automotive applications, refer to GL net-lists as "appropriate for fault injection" since they contain sufficient information about a system's structure [1]. Furthermore, RTL models are also "an acceptable approach for stuck-at faults, provided that the correlation with gate level is shown" [1]. Thus, fault injection into lower abstraction levels (e.g., transistor level) is not necessary for automotive-related SoCs.

Moreover, the interest in performing safety verification on virtual prototypes (VPs) using SystemC and TLM [2] is increasing since VPs allow safety-architecture exploration early in the design phase [3]. However, the insertion of a sufficient number of realistic faults into such models is an ongoing challenge.

Due to lack of access to implementation details, faults cannot be inserted into transaction-level (TL) models using GL granularity but must be inserted at the TL model's boundaries (i.e., interface) or into the TL model's limited number of internal variables.

As a consequence, not only single bit but also multi-bit errors must be injected into TL models to get a realistic error footprint. The need for multi-bit fault injection can be easily explained utilizing a simple multi-bit adder with carry-in. On the one hand, if both summands are 0, the effect of a stuck-at-1 fault at the carry-in affects only one bit of the sum. On the other hand, if all bits of one summand are set to 1 and the other summand is 0, a stuck-at-1 fault injected into the carry-in bit affects all bits of the sum. In other words, a realistic fault-injection campaign performed on an add-operation in TLM must be able to insert any number of faults in parallel. However, inserting any number of faults on high abstraction levels can result in analysis of wrong (i.e., unrealistic) failures and, as a consequence, can also result in the implementation of unnecessary safety mechanisms to correct the observed failures.

Our focus lies in the insertion of realistic faults into TL models. For this reason, we follow a structured approach of fault abstraction as depicted in Fig. 1. We start at the transistor level since every fault originates here. From there, we follow the classical abstraction via transistors and gates to RTL models. The stuck-at faults conceptually remain the same; however, they are applied to different kinds of connectivity specifications. It is important that some matching points remain visible over all abstraction levels. Therefore, we model our systems at the TL so that some of the matching points can still be found.

Our definition of *matching points* includes, but is not limited to, module interfaces, buses, memories, and important block states (e.g., special purpose registers). In our case, we have embedded behavioral components into TLM methods, components which are reflected in RTL and GL models as well.

To address the problem of realistic fault injection, we have replaced primitive combinational TL models (e.g., adders, barrel shifters) with refined implementations. Since the primitive combinational TL models contain neither internal states, nor variables and execute a single operation (e.g., add, shift), their refined implementations

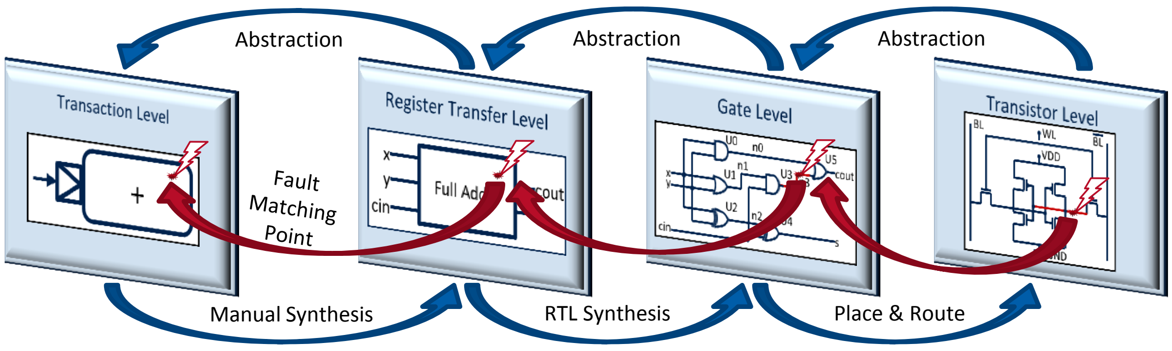

Figure 1. Fault matching-points on different abstraction levels

allow insertion of faults with GL granularity. Furthermore, we have co-simulated the new models utilizing a compiled-code method [4] and fault-injection variables. In doing so, the TL models have gained all injectable fault-locations (e.g., wires, ports) present in the GL net-list, while preserving their original functionality and while maintaining a sufficient amount of the original TLM simulation speed. Next, we have randomly injected faults into the refined implementation and, finally, observed the injected faults' propagation effects through the TL system. Thus, we have been able to replace the inaccurate multi-bit fault-injection approach at the boundaries of TL models with GL-accurate single-bit fault injection into refined TL models. This work was adapted from [5].

The remainder of this paper is structured as follows. Section II presents existing contributions related to the topics of fault injection and safety verification. Section III contains a description of our TLM-GL co-simulation and fault-injection frameworks. Simulation results and measurements made on several synthesizable circuits and a MIPS CPU are described in section IV. Finally, this paper's summary is presented in section V.

II. Related Work

GL fault-injection techniques have been reported as highly effective for safety verification of SoCs [1], [6]. However, large-scale safety-critical microcontrollers such as the STMicroelectronics SPC5 family [7] or the Infineon AURIX$^{\text{TM}}$family [8] are too large for complete system-level safety verification.

Currently, RTL-based fault-injection techniques are widely used in safety-verification flows [9]. However, such frameworks come with a series of drawbacks. First, many techniques inject faults only into RTL systems and exclude comparisons with the corresponding GL model [10], [11]. Therefore, potential matching points are not even analyzed. Second, it is difficult to find matching points to combinational GL net-lists even on the RTL abstraction level [12]. Finally, fault-injection campaigns on RTL models suffer from slower development, debugging, and simulation speeds compared to TLM.

Formal safety-verification approaches have been used to reduce the verification time of safety-critical registers [13]. However, formal checking is still too computationally expensive and has limited scalability on large designs.

FPGA (field-programmable gate arrays) emulation has been extensively researched in the last decades with respect to safety verification [14], [15]. During this time, much progress has been made in the area of simulation speed, improved debugging, and higher fault-injection controllability. However, FPGA emulation still presents three fundamental drawbacks: i) high development cost, ii) complex usability for large net-lists (e.g., AURIX), and iii) different timing properties between FPGA and target SoC technologies.

At the present time, SystemC-TLM-based VPs are created more frequently because they offer much faster simulation speeds than RTL and GL models [16].

Moreover, mixed-level fault-injection methodologies (e.g., TLM-RTL) are constantly emerging [17]–[19]. In [17], a wrapper has been used to co-simulate TL and RTL models. The wrapper was responsible for adding missing RTL signals to the TLM abstraction level (e.g., `read/write enable` signals). Furthermore, the wrapper handles the communication synchronization between TLM and RTL. Faults have been injected at the RTL model's boundary and their propagation has been observed at the TLM outputs. Moreover, a concurrent simulation approach has been used to perform the fault-injection campaigns. As mentioned in [20], the main drawback of mixed-level safety-verification methodologies is that faults injected into TLM or RTL systems provide sub-optimal results compared to fault injection into GL models. This inaccuracy is attributed: i) to the lack of injectable fault-locations (e.g., wires, ports) and ii) to different propagation effects of the same faults injected at different abstraction levels. This inaccuracy has a considerable negative impact on the efficient design of sufficient safety mechanisms early in the development phase. Therefore, an approach is needed to provide better matching points among multiple abstraction levels.

In this paper, we present a methodology to increase the number of injectable fault-locations on SystemC-TLM-based VPs because the number of faults that can be injected into any model at a higher level of abstraction is significantly smaller than in the equivalent GL implementation. Our work is complementary to that presented in [17]. We also perform mixed-level safety verification but we apply our approach on TL and GL models (not RTL). Moreover, we use saboteurs [21] instead of mutants to perform fault injection and a compiled-code approach [4] to co-simulated TL and GL models. Furthermore, we use a model-based approach to transform GL net-lists into C++ code, which we later integrate into TLM VPs. The generated C++ code replaces equivalent functionality modeled in the VPs. Thus, we are able to successfully inject the same realistic GL faults into TL models and still maintain a fast simulation speed slightly below TLM.

III. Mixed-Level Co-Simulation

We have applied a top-down development approach, in which functionally equivalent TL and RTL models have been simultaneously implemented from a common hardware specification. After the models have been sufficiently tested (Fig. 2), the RTL models have been synthesized using a commercial synthesis tool [22]. Next, the resulting Verilog net-list has been transformed into C++ code using VERITAS [4]. This code has been used to replace primitive combinational TL components (Fig. 3). Finally, we have injected faults into the GL-accurate TL models using Monte Carlo during a safety-verification co-simulation campaign.

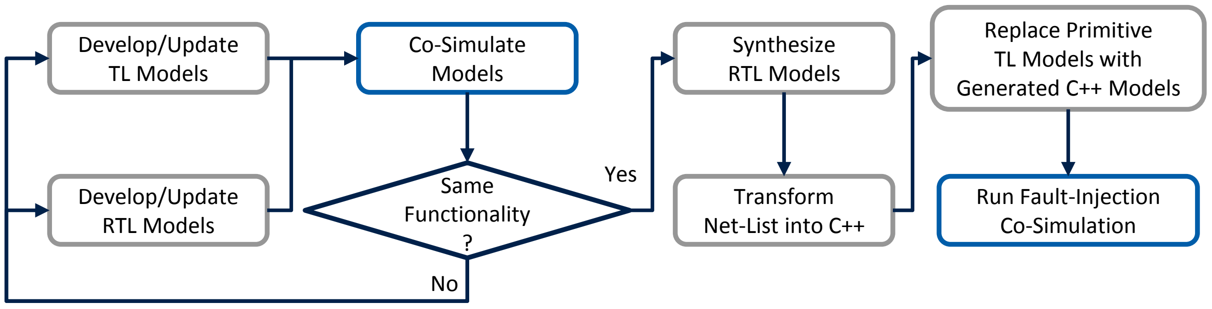
Develop/Update TL Models
Develop/Update RTL Models
Co-Simulate Models
Same Functionality ?
Yes
No
Synthesize RTL Models
Transform Net-List into C++
Replace Primitive TL Models with Generated C++ Models
Run Fault-Injection Co-Simulation

Figure 2. SoC development flow until application of fault injection

We define *GL-accurate TL models* as TLM target components with GL granularity. Thus, TL models which only represent one or more primitive operations (e.g., add, shift, multiply operations), become enriched with the GL structure and behavior. In section IV, we will show that this enrichment only adds an acceptable simulation overhead to the TL models.

For fault-injection purposes, we have extended VERITAS' C++ generators to use a special fault-injection object (FIO) class (section III-D).

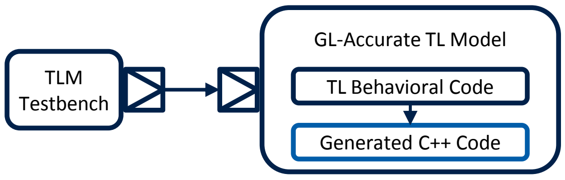

Figure 3. Integration of GL-accurate C++ code into TLM-based virtual prototypes

A. *Mixed-Level Fault Injection*

SaVer (SystemC safety-verification platform) is a fault-injection framework for SystemC-TLM-based models. Here, FIOs are used to inject faults into generated C++ code. Additionally, SaVer provides a means to create fault-injection stimulus executed in the context of fault-injection campaigns.

B. *Simulation Environment*

The C++ code generated from VERITAS is integrated into existing SystemC-TL models. Afterwards, the TL models are introduced into a fault-injection simulation environment (Fig. 4). The environment contains a stimulus generator, a transaction monitor, and a fault injector. The stimulus generator drives data either randomly or targeted (i.e., from a stimulus library) onto the SystemC model's input ports. Additionally, the fault injector accesses the model's internal signals and injects faults therein from a fault library. The monitor detects changes at the model's output ports. When the output values change, the monitor sends the observed data to an analyzer for processing. Finally, a controller regulates the driven stimulus, the injected faults, and the life time of a simulation.

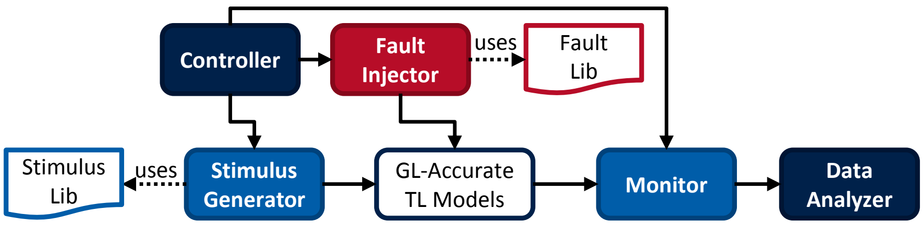

Figure 4. SaVer's verification environment with fault-injection features

C. Stimulus Generation

Hardware models require $\prod_{i=1}^{P} 2^{n_i}$ input stimuli to be exhaustively verified, where P is the number of input ports and n_i is the number of bits for input port i. Of course, this exponential amount of stimuli makes the exhaustive verification of even a simple 32-bit adder unfeasible.

To mitigate this computationally-expensive drawback, while still maintaining a high degree of accuracy, SaVer's stimulus generator creates only N input vectors for each TL model, where N is the TL model's number of input bits. Here, all input bits are '0' except for one bit which is '1' and shifted across the N input bits. In this case, this approach ensures 100% toggle coverage on all TLM inputs.

Additionally, the stimulus generator provides cross stimuli for all combinations of input ports. As a result, the total number of stimuli becomes $\prod_{i=1}^{P} n_i$.

Finally, we also apply random and real-world patterns whenever necessary.

The accuracy of our linearized stimulus generator is discussed in section IV-D.

D. Fault-Injection Objects

The fault-injection object (FIO) class (`fi_object`) is effectively an objected-oriented-based saboteur [21]. Apart from the classic read and write methods of a SystemC signal, the FIO contains extra methods for injecting permanent (`inject_sa0`, `inject_sa1`) as well as transient (`inject_seu`) faults. The method `inject_seu` simply flips a signal's value at a given bit location. This value can be overwritten by the simulator during the FIO's next write cycle. Conversely, the `inject_sa0` and `inject_sa1` methods effectively block the FIO's value update process during subsequent write cycles. The permanent fault's effect can be removed during a simulation by calling the FIO's `release` method.

E. Fault-Injection Simulation Management

To comprehensively cover the safety-verification space of our hardware systems, we have randomly injected one fault per co-simulation into the GL-accurate TL models. Additionally, the effects of the injected faults have been monitored at the outputs of the TLM verification environment.

To test our compiled-code simulation, we have injected only permanent faults (i.e., stuck-at-0 and stuck-at-1) since the effect of transient faults in combinational circuits is still relatively low [23]. Nevertheless, SaVer's FIO is able to inject transient faults as well.

We have injected each fault at the beginning of each simulation and maintained the fault's effect until the end of simulation.

We have chosen to only inject faults into internal signals and output ports (i.e., injectable fault-locations) of a hardware system. Input ports have been excluded since they are already accessible from the stimulus generator.

By only injecting one fault per simulation and only using two fault models for each internal signal, the contribution of the fault verification-space to the total number of fault-injection simulations becomes $2 \cdot \sum_{i=1}^{S} n_i$, where S and n_i are the number of internal signals and output ports and the number of bits for signal or port i, respectively.

F. Simulation Flow

All fault-injection co-simulations have been executed using SaVer's flow (Fig. 5).

First, one reference simulation (i.e., without fault injection) has been executed for each workload to serve as a golden run. Next, all fault-injection simulations calculated with the equation from Section III-E have been run. Finally, the fault-injection simulation results have been compared to the reference-simulation results. If no mismatch has been detected after a simulation, the fault has been considered masked. Otherwise, any

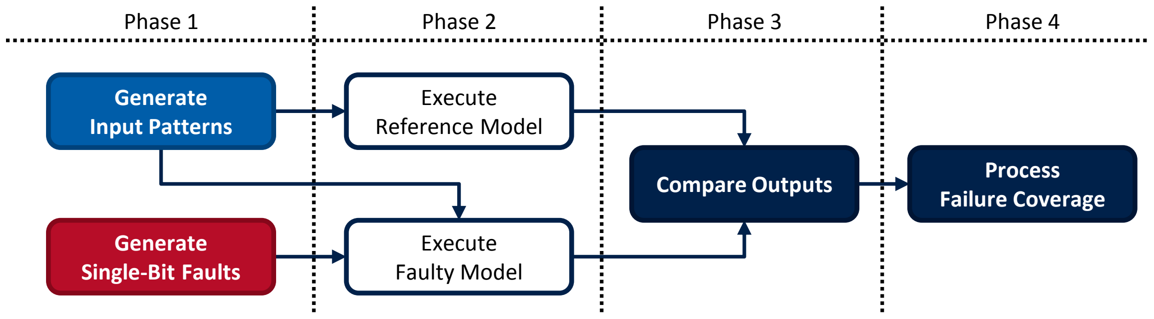

Figure 5. SaVer's fault-injection regression flow

mismatch between the compared simulation data has been considered to be the result of a failure detected at the TL model's outputs. In other words, the injected fault was propagated through the model until it reached the model's outputs.

IV. Case Studies and Results

All simulations have been performed on a 64-bit machine with an Intel® Xeon® E5 CPU @3.00 GHz, L3 cache 25600 kB, and 264 GB RAM.

We have measured the simulation speeds of all our case studies with the UNIX `time -v` command. We have excluded the measurement of compilation and elaboration times of the co-simulated models.

We have simulated all Verilog GL net-lists as well as the MIPS-RTL models with an off-the-shelf HDL simulator [24] and all (GL-accurate) TL models with the SystemC reference simulator [2].

A. Adder Architectures

We have chosen adders with overflow to analyze the relationship among different hardware architectures and almost identical equivalent TL models (i.e., the carry bit is modeled on different bit positions).

The primitive TL model of an n-bit adder with overflow has no internal states and only implements the add-operation. In this case, faults can only be injected into the model's outputs (i.e., `carry out` and `sum`).

The GL-accurate TL models generated with VERITAS as C++ code contain the models' complete GL structure visited in topological order. Hence, all injectable fault-locations present in the GL models are preserved within the GL-accurate TL models. As shown in Table I, it is clear that the original primitive TL models have significantly less fault-injection locations than the GL models. More precisely, there is on average less than 17% correlation between the primitive TL models and the GL net-list. Moreover, the number of injectable fault-locations increases notably with the adder's size (i.e., the numbers of internal signals and bits increase). Additionally, more complex adder implementations (e.g., carry look-ahead) provide a greater number of injectable fault-locations in comparison to the TL model.

Model	Injectable Fault-Locations (#)		Correlation Percentage	Simulation Time (s)		Slow-Down Factor
	Primitive TL Model	Gate-Level-Accurate TL Model		Gate-Level-Accurate TL Model	Primitive TL Model	
full_adder	2	6	33.33%	0.041	0.040	1.03x
adder_2bit	3	9	33.33%	0.097	0.087	1.11x
nibble_adder	5	31	16.13%	0.260	0.122	2.13x
adder_4bit	5	35	14.29%	0.310	0.122	2.54x
addsub_8bit	9	69	13.04%	0.690	0.194	3.57x
adder_8bit	9	77	11.69%	1.600	0.194	8.27x
adder_16bit	17	163	10.43%	3.470	0.405	8.58x
adder_32bit	33	339	9.73%	13.550	1.142	11.87x
carry_lookahead_32bit	33	523	6.31%	16.040	1.142	14.05x
		Average	16.48%		Average	5.91x

Table I. Comparison in number of fault-injection locations and simulation slow-down for several adder architectures

We have compared the co-simulation speed of our compiled-code approach against that of the original pure TL model and the pure GL simulation. We have observed that: i) the GL-accurate TL models are about 6x slower than the original primitive TL models (Table I) and ii) the GL-accurate TL models are on average about 2-5x faster than the original GL models (Fig. 6). Moreover, Fig. 6 indicates that the GL-accurate TLM simulations increase in performance with the size of the simulated model (i.e., the 32-bit adder is over 6x faster than the 2-bit adder).

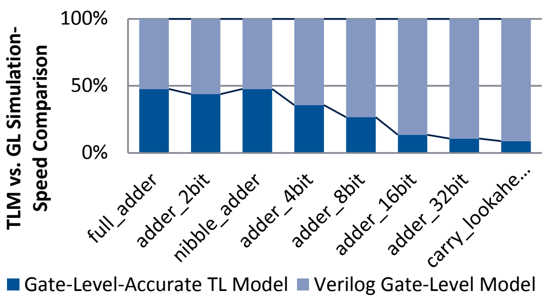

Figure 6. Simulation-speed comparison between each gate-level and gate-level-accurate TL model

Since we have enriched the primitive TL models with extra functionality and structure, the effects on simulation performance (i.e., faster than GL and slower than pure TLM) had already been expected. Nevertheless, the results show that a successful trade-off between simulation speed and fault-injection accuracy can be reached when co-simulating mixed-level hardware models using our compiled-code approach. Moreover, this trade-off can be further improved by optimizing the generated compiled-code approach (e.g., generating a reduced amount of fault-injection locations from the GL net-list).

B. MIPS CPU Architecture

The MIPS-RTL model contains an integer-based instruction set, five pipeline stages, hazard detection, and forwarding. The MIPS-TL model is a pipelined behavioral model functionally equivalent to the RTL implementation. We used components of a MIPS implementation to cover a wide range of design alternatives (e.g., simple shifters in the logic unit and look-up tables in the instruction-decode control unit). Moreover, we used the complete MIPS models to compare the performance of our GL-accurate TL models to the RTL and GL implementations to show speed-up to today's state-of-the-art technology.

Table II presents the impact of our method on the number of injectable faults added to the GL-accurate TL models and their execution speed. The results are similar to those in the previous case study: there is on average less than 17% correlation between TLM and the MIPS-GL net-list.

Model	Injectable Fault-Locations (#)		Correlation Percentage	Simulation Time (s)		Slow-Down Factor	Simulation Time RTL (s)	Speed-Up Factor
	TL Model	Gate-Level-Accurate TL Model		Gate-Level-Accurate TLM Model	TL Model			
adder	33	209	15.79%	2.593	0.24	10.80x	70.831	27.32x
ex_forwarding	4	38	10.53%	1.192	0.24	4.97x	70.831	59.40x
hazard_detection	4	73	5.48%	1.610	0.24	6.71x	70.831	43.99x
id_control	11	116	9.48%	1.914	0.24	7.98x	70.831	37.01x
jump	3	10	30.00%	0.408	0.24	1.70x	70.831	173.56x
logic_unit	32	266	12.03%	2.761	0.24	11.50x	70.831	25.65x
shift	32	931	3.44%	5.564	0.24	23.18x	70.831	12.73x
subtractor	33	174	18.97%	2.164	0.24	9.02x	70.831	32.73x
write_back	38	82	46.34%	0.380	0.24	1.58x	70.831	186.40x
		Average	16.89%		Average	8.60x	Average	66.53x

Table II. COMPARISON IN NUMBER OF FAULT-INJECTION LOCATIONS AND SIMULATION SLOW-DOWN FOR A MIPS CPU

However, the main differences have been observed when comparing simulation speeds. The original MIPS TL and RTL models have been simulated without fault injection and recorded into Table II. The GL-accurate MIPS-TL models have been simulated by successively replacing each primitive TL mode with the enriched C++ models generated from VERITAS.

We have observed an average slow-down of the GL-accurate TL models compared to the pure TL of less than 10x but a surprising average 65-70x speed-up against the RTL simulation. Moreover, when compared to the synthesized Verilog net-list simulation, the GL-accurate TL models are over three orders of magnitude faster to simulate. These results offer a conclusive benefit for early and comprehensive fault-effect analysis and safety-architecture exploration.

C. Fault-Injection Object's Performance

We have benchmarked our fault-injection class against C++ Boolean variables using 2^{32} read and write accesses over 10 simulations. Our measurements have shown that after applying the GNU C++ compiler's optimizations, a simulation with our fault-injection class took an average of 253.706 ms, compared to the simulation with C++ Boolean variables which needed an average of 248.639 ms. Thus, our fault-injection class is only approximately 2% slower.

D. Fault-Effect Analysis

To calculate the accuracy of our linearized stimulus generator with respect to fault propagation, we have measured how many of the injected faults have propagated to the outputs of our TL models. We have achieved this by injecting faults into all fault-injection objects of a TL model and applying all input patterns per injected fault. As illustrated in Fig. 7, although we have drastically reduced the number of input stimuli, only about 10% of the injected faults have been ultimately masked. In other words, although we did not exercise all input vectors, around 90% of the faults injected into combinational circuits have propagated to the system's output and have become failures. We argue that the missing failures either can be easily triggered by extending the current library of input vectors (i.e., workload library) with directed vectors instead of Monte Carlo simulation, or cannot be reached with any input vector.

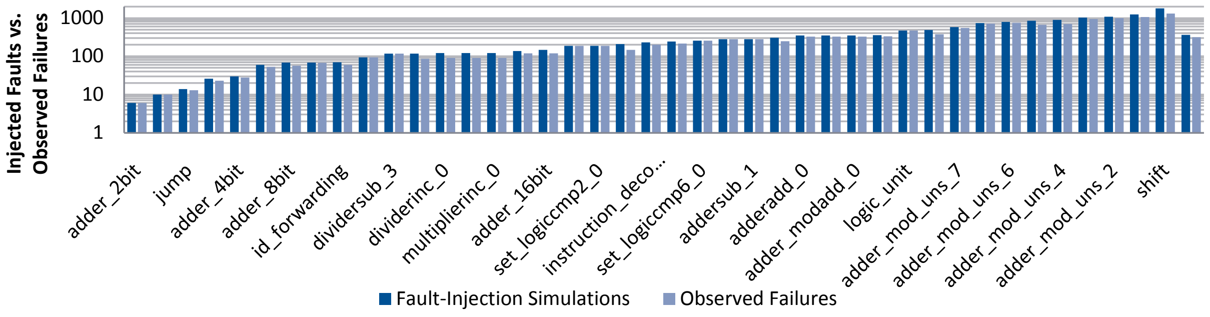

Figure 7. Accuracy of our linearized stimulus generator based on the ratio of injected faults and observed failures

V. Summary

In this paper, we have presented a comprehensive methodology to completely correlate injectable fault-locations on mixed-level hardware models (i.e., SystemC-TL models and GL net-lists).

Additionally, we have presented a fault-injection framework to inject realistic faults into TL models. This framework offers much faster fault-injection campaign speeds than RTL and GL simulations and those of classic safety-verification flows as well (i.e., on average 65-70x).

After applying the concepts presented in this paper, TL models which originally contained less than 17% of the total fault-injection locations of their equivalent GL models have been enhanced with all GL-specific combinational fault-injection locations. Furthermore, our approach allows the simulation of all realistic single-bit faults on the TLM abstraction level as requested by the ISO 26262. Therefore, we have effectively bridged the safety-verification gap with respect to injecting faults on multiple abstraction levels.

In other words, the effectiveness of implemented safety mechanisms can be tested at GL accuracy and at a simulation speed slightly slower than that of VPs. Additionally, analyses may be conducted which detect if safety mechanisms are missing, ineffective, or redundant.

Acknowledgment

This work is partially supported by the German Federal Ministry of Education and Research (BMBF) in the project EffektiV (contract no. 01IS13022).

References

[1] ISO, CD, "26262, Road Vehicles–Functional Safety," *International Standard ISO/FDIS*, vol. 26262, 2011.

[2] O. S. Initiative *et al.*, "IEEE Standard SystemC Language Reference Manual," *IEEE Computer Society*, 2006.

[3] J.-H. Oetjens, O. Bringmann, M. Chaari, W. Ecker, B.-A. Tabacaru *et al.*, "Safety Evaluation of Automotive Electronics Using Virtual Prototypes: State of the Art and Research Challenges," in *Design Automation Conference (DAC), 51st ACM/EDAC/IEEE*. IEEE, 2014, pp. 1–6.

[4] B.-A. Tabacaru, M. Chaari, W. Ecker, T. Kruse, and C. Novello, "Fault-Effect Analysis on Multiple Abstraction Levels in Hardware Modeling," *DVCon USA*, pp. 1–12, 2016.

[5] ——, "Gate-Level-Accurate Fault-Effect Analysis at Virtual-Prototype Speed," *ERCIM/EWICS/ARTEMIS Workshop on "Dependable Embedded and Cyber-physical Systems and Systems-of-Systems" (DECSoS'16)* – accepted, to be published, pp. 1–13, 2016.

[6] J. Espinosa, C. Hernandez, and J. Abella, "Characterizing Fault Propagation in Safety-Critical Processor Designs," in *On-Line Testing Symposium (IOLTS), 2015 IEEE 21st International*. IEEE, 2015, pp. 144–149.

[7] STMicroelectronics, "32-bit Power Architecture® Microcontroller for Automotive SIL3/ASIL-D Chassis and Safety Applications," SPC56 Datasheet, Rev 11, 2014.

[8] I. T. AG, "AURIX–TriCore Datasheet." Accessed: 2016-2-22. [Online]. Available: https://www.infineon.com/dgdl?folderId=db3a304412b407950112b409ae660342&fileId=db3a30431f848401011fc664882a7648

[9] R. Leveugle, D. Cimonnet, and A. Ammari, "System-Level Dependability Analysis with RT-Level Fault Injection Accuracy," in *Defect and Fault Tolerance in VLSI Systems, 2004. DFT 2004. Proceedings. 19th IEEE International Symposium on*. IEEE, 2004, pp. 451–458.

[10] M. Schwarz, M. Chaari, B.-A. Tabacaru, and W. Ecker, "A Meta-Model-Based Approach for Semantic Fault Modeling on Multiple Abstraction Levels," *DVCon Europe*, pp. 1–6, 2015.

[11] I.-D. Vidrascu, "Implementation of a Safety Verification Environment (SVE) Based on Fault Injection," Master's thesis, Fachhochschule Kärnten, Klagenfurt am Wörthersee, Austria, 2015.

[12] H. R. Zarandi, S. G. Miremadi, and A. Ejlali, "Dependability Analysis Using a Fault Injection Tool Based on Synthesizability of HDL Models," in *Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on*. IEEE, 2003, pp. 485–492.

[13] H. Busch, "An Automated Formal Verification Flow for Safety Registers," *DVCon Europe*, pp. 1–8, 2015.

[14] F. Kastensmidt and P. Rech, *FPGAs and Parallel Architectures for Aerospace Applications: Soft Errors and Fault-Tolerant Design*. Springer, 2015.

[15] C. Bernardeschi, L. Cassano, and A. Domenici, "SRAM-Based FPGA Systems for Safety-Critical Applications: A Survey on Design Standards and Proposed Methodologies," *Journal of Computer Science and Technology*, vol. 30, no. 2, pp. 373–390, 2015.

[16] K.-J. Chang and Y.-Y. Chen, "System-Level Fault Injection in SystemC Design Platform," in *Proceedings of 8th International Symposium on Advanced Intelligent Systems (ISIS)*. Citeseer, 2007.

[17] R. Baranowski, S. Di Carlo, N. Hatami, M. E. Imhof, M. A. Kochte, P. Prinetto *et al.*, "Efficient Multi-Level Fault Simulation of HW/SW Systems for Structural Faults," *Science China Information Sciences*, vol. 54, no. 9, pp. 1784–1796, 2011.

[18] M. Kochte, C. G. Zoellin, R. Baranowski, M. E. Imhof, H.-J. Wunderlich, N. Hatami *et al.*, "Efficient Simulation of Structural Faults for the Reliability Evaluation at System-Level," in *Test Symposium (ATS), 2010 19th IEEE Asian*. IEEE, 2010, pp. 3–8.

[19] M. B. Santos and J. P. Teixeira, "Defect-Oriented Mixed-Level Fault Simulation of Digital Systems-on-a-Chip Using HDL," in *Design, Automation and Test in Europe Conference and Exhibition 1999. Proceedings*. IEEE, 1999, pp. 549–553.

[20] H. Cho, S. Mirkhani, C.-Y. Cher, J. A. Abraham, and S. Mitra, "Quantitative Evaluation of Soft Error Injection Techniques for Robust System Design," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. IEEE, 2013, pp. 1–10.

[21] H. Ziade, R. A. Ayoubi, R. Velazco *et al.*, "A Survey on Fault Injection Techniques," *Int. Arab J. Inf. Technol.*, vol. 1, no. 2, pp. 171–186, 2004.

[22] Synopsys®, "Design Compiler® User Guide," Accessed: 2016-2-23. [Online]. Available: http://acsweb.ucsd.edu/~coz004/DC_user_guide.pdf

[23] P. E. Dodd, M. R. Shaneyfelt, J. A. Felix, and J. R. Schwank, "Production and Propagation of Single-Event Transients in High-Speed Digital Logic ICs," *Nuclear Science, IEEE Transactions on*, vol. 51, no. 6, pp. 3278–3284, 2004.

[24] M. Graphics®, "Questa® SIM User's Manual," Accessed: 2016-2-23. [Online]. Available: http://rise.cse.iitm.ac.in/people/faculty/kama/prof/questa_sim_user_manual.pdf