Accepted manuscript, the final version can be found at: https://www.computer.org/csdl/proceedings-article/dft/2023/10313565/1S4hTTCnmzC
DOI Bookmark: 10.1109/DFT59622.2023.10313565

Black-Box IP Validation with the SafeTI Traffic Injector: A Success Story

Francisco Fuentes[†,‡], Sergi Alcaide[†], Raimon Casanova[‡], Jaume Abella[†]

[†]Barcelona Supercomputing Center (BSC)
Barcelona, Spain

[‡]Microelectronic and Electronic Systems Department
Universitat Autònoma de Barcelona (UAB), Bellaterra, Spain

***Abstract*—Functional and performance validation of high-performance safety-related hardware platforms require generating specific traffic patterns in the network-on-chip (NoC) to test IP components and their integration. Software-only tests offer indirect control on the NoC traffic and are subject to numerous constraints (e.g., the inability to inject bursty traffic synchronously since cores may not be able to generate it explicitly). Instead, hardware traffic injectors, such as the SafeTI are much more versatile and controllable.**

This paper presents how our SafeTI hardware traffic injector can be used for the validation of an L2 cache in a safety-relevant space MPSoC by Frontgrade Gaisler, and how such validation process allowed identifying a hard-to-detect design flaw that led to decreased available cache space. Overall, our experience shows a specific application of the SafeTI for black-box IP validation and a success story for that application.

***Index Terms*—design for testability, fault diagnosis, multiprocessor SoC, embedded systems, cache validation**

I. Introduction

The increasing number and complexity of safety-related functionalities, including those related to autonomous operation, impose the adoption of higher performance Multi-Processor Systems-on-Chip (MPSoCs) in domains such as automotive, avionics, robotics, and space, to name a few [18]. For instance, multicores, in many cases with some form of acceleration support, are in the process of being adopted in automotive [17], avionics [20], and space [12].

A common trait in all those MPSoCs is the use of a shared second-level (L2) cache across all or a subset of cores, which implement private L1 caches. Those L2 caches bring increased on-chip storage for data, hence reducing memory access latencies if data locality is properly exploited, as well as other non-functional properties such as increased reliability (e.g., by smartly managing write policies and error correction codes across L1 and L2 caches [12]).

Safety-related systems must undergo strict development processes in line with domain-specific safety guidelines, such as ISO26262 [15] for automotive and ECSS standards for space [9]. One key step of such development processes is the use of tests for component and system validation. However, the increasing complexity of the MPSoCs deployed in safety-related systems challenges the ability to test those MPSoCs themselves due to the lower controllability that software running on the cores can exercise on processor resources without explicit or direct control, such as L2 caches.

Recently, we presented the SafeTI traffic injector [21], and released it as open source [6]. Such a programmable traffic injector is a flexible tool for platform co-testing and offers a large variety of applications.

This paper presents a success story of a smart use of the SafeTI to test a commercial open-source L2 cache by Frontgrade Gaisler, part of the IPs used to build LEON5 and NOEL-V space multicores [2]. Our work has allowed spotting a hard-to-find design flaw by means of the SafeTI, narrowing down its cause, and sharing the information with Gaisler to get it fixed in the near future. In particular, the contributions of this work are as follows:

1) We instantiate the SafeTI in a space SoC based on Gaisler's technology and program traffic patterns to test the L2 cache's behavior.
2) We identify a design flaw affecting only the synthesized version of the component, hence not findable in the pre-synthesis simulation model, and show how it leads to a decrease in the effective L2 cache size available.
3) By means of a series of experiments building on the SafeTI, we narrow down the problem and patch it. In doing so, the flaw and the solution, along with detailed experiment data have been packed and sent back to Gaisler to help them fix the issue, which should be no longer in place soon when the new release of the L2 cache component is published.

The rest of the paper is organized as follows. Section II provides some background on the SafeTI and the platform used for our work. Section III presents our instantiation of the SafeTI and explains the methodology applied to test the L2 cache component mentioned before. Section IV brings quantitative evidence of the flaw spotted with our work, including the solutions we propose, and shows the effectiveness of the correction. Section V provides some related work. Finally, Section VI concludes this paper.

This work is part of the project (ISOLDE), funded by MCIN/AEI/10.13039/501100011033 and the European Union NextGenerationEU/PRTR under grant PCI2023-143372, and the European Union's Horizon Europe Programme under project KDT Joint Undertaking (JU) under grant agreement No 101112274. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant PID2019-107255GB-C21 funded by MCIN/AEI/10.13039/501100011033.

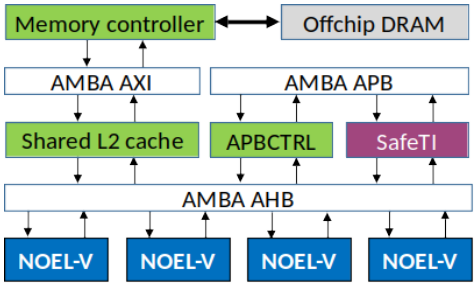

Fig. 1. SELENE platform block diagram of related components.

II. Background

This section provides some background on the SELENE platform, and on the SafeTI, on which we build for our work.

A. SELENE MPSoC platform for safety-critical applications

SELENE is a MPSoC platform [8] tailored for safety-critical applications based on Frontgrade Gaisler 64-bit RISC-V processor NOEL-V [12]. The platform infrastructure and core components (e.g., caches) are integrated from Frontgrade Gaisler GRLIB IP library [2] with General Public License (GPL), allowing us to perform research and test operations with the SafeTI.

For this case study and as Fig. 1 shows, we have integrated a SafeTI module targeting the L2 cache level in a SELENE instance of 4 NOEL-V cores. The implemented L2 cache component specifications include a pseudo-Least Recently Used (pLRU) replacement policy, with an associative level of 4 ways and a total capacity of 512 KB.

B. SafeTI traffic injector

The SafeTI has been conceived as a flexible and programmable hardware traffic injector, using descriptors to encode the traffic patterns to be injected at the target interconnect. The programming and control of a SafeTI instance are executed through the integrated Advanced Microcontroller Bus Architecture (AMBA) Advanced Peripheral Bus (APB) Subordinate interface. Meanwhile, the Manager interface for targeting the interconnect is interchangeable, allowing future extensions for targeting virtually any communication protocol, which at the present can target AMBA AHB or AXI4 interconnects. The SafeTI design is open source [6], [21] and can be both simulated and synthesized for FPGA prototyping.

Descriptor characteristics include options such as access size, initial address of the access, and type of access, among others. The SafeTI component also integrates options for automatic halt at network error response, injection pattern completion, and other utilities for expanded study or usage.

In terms of the resource usage for integrating the SafeTI in the platform, the module requires 3.2 % LUT's and 12.6 % registers with respect to one of the platform cores, though this resource data may differ in future designs by applying specific FPGA resource optimizations and implementing extensions for additional features.

III. Contribution

The SafeTI can be used for memory and cache components testing by injecting specific traffic patterns while making other components in the SoC (e.g., computing cores) be in a controlled state (e.g., idle) as shown in a previous paper [21]. In the context of our work, all cores but one are disabled and, as discussed in this section, the SafeTI traffic injection along with a specific benchmark running in the active core allow testing the L2 cache in the SoC.

One of the SafeTI advantages over other solutions is that, once integrated within a system, it can perform generic traffic tasks without prior knowledge of the memory components characteristics, providing a black-box testing method applicable for virtually any platform. In detail, the SafeTI can be programmed to access a specific data size in loop during a controlled stressing-test execution by one of the cores, a benchmark which accesses preferably twice of the largest cache size and misses on every cache. Since the SafeTI access latency depends on the proximity of the data that it is accessing to, several runs sweeping on the SafeTI access size produces a profile of the system caches, correlating cacheability with benchmark duration due to the interference between both actors. However, the SafeTI is limited to study only those components which it has direct or indirect access, meaning the module must be integrated at or below the level in which the target component is located at.

Software-only tests are unable to perform the injection done with the SafeTI due to several reasons. For instance, the RISC-V Instruction Set Architecture (ISA), which the NOEL-V cores implement, does not include instructions for generating burst accesses. Therefore, the cores accesses are limited to be atomic. In addition, the cores are integrated with the aim of minimizing off-core accesses, so they include private L1 data and instruction caches, which imposes some filtering to reach the L2 cache. The SafeTI does not have these limitations due to being able to be integrated at any level of the memory hierarchy reachable from an interconnect, easing the traffic pattern planning and obtaining data with reduced noise.

IV. Evaluation

The evaluation has been executed at a bare-metal synthesis of SELENE on the Virtex UltraScale+ VCU118 FPGA-based evaluation kit [25], operating at a frequency of 100 MHz. Software programs have been compiled from C code with Frontgrade Gaisler's NCC GCC Bare-metal toolchain 1.0.4 release on a Linux system with an O2 optimization level for a RISC-V target. For the simulation and synthesis environments, we followed the platform recommendations: Siemens Questa 2019.4 and Xilinx Vivado 2020.2, respectively.

When performing the black-box testing method, we have found anomalous behavior not consistent with the L2 cache specifications, as we explain in Section IV-A. Further, Section IV-B focuses on the design fault found and its solution, and Section IV-C shows increased performance due to the fix.

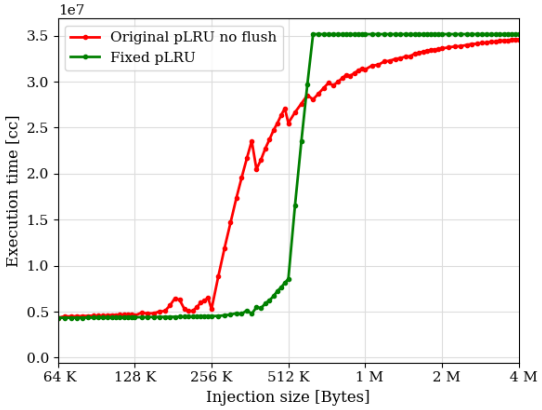

Fig. 2. Benchmark execution times, in clock cycles, for different SafeTI access sizes at the original L2-lite cache pLRU (red) and the revised pLRU (green) implementations.

```
procedure pLRU_update(
 variable pLRU_data : inout pLRU_data_t;
 ...

pLRU_update(v.pLRU_data, ... );
```

Fig. 3. VHDL design excerpts of the procedure declaration (top) and use (bottom) from the L2 cache pLRU logic, where the red text fixes the fault.

A. Black-box testing with SafeTI

The pLRU implementation of the L2 cache replacement policy provides an ordered line replacement behavior, which can be studied with relative ease using the SafeTI. For instance, Fig. 2 shows the execution time of several benchmark runs while the SafeTI is programmed to access between 64 KB and 4 MB on the FPGA platform. Note that the horizontal axis of the figure is in logarithmic scale for clarity. The graph red line shows the cache begins to be full around 256 KB instead of the expected 512 KB, while the curve does not correspond to a pLRU replacement policy, pointing to a design fault.

B. Design fault on the L2 cache pLRU logic

Thanks to GRLIB library being open-source, we have been able to further debug the issue. In essence, black-box testing on Register-Transfer Logic (RTL) simulator would provide data correspondent to a pLRU implementation of a 512 KB cache capacity, indicating that the design fault is related to non-deterministic behavior between simulation and synthesis.

Fig. 3 shows an excerpt of the VHDL procedure implementing the pLRU bit-tree update with a particular design flaw that results in the discrepant behavior of the L2 cache. In detail, the non-initialization of one of the outputs allowed the synthesis tool to perform some optimizations, usually encouraged, but in this specific case resulted in the problematic behavior. By initializing the output variable with the previous state of the bit-tree, as marked in red in the excerpt, the L2 cache would respond as expected following Fig. 2 green line, showing the cache capacity of 512 KB as specified.

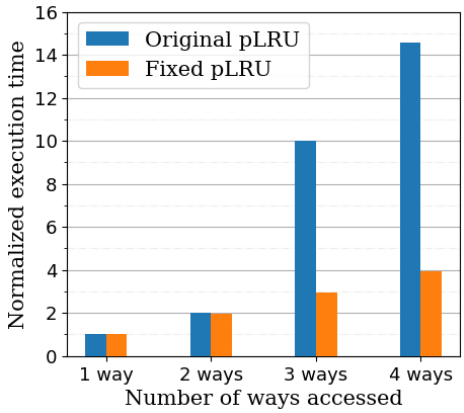

Fig. 4. Stressing test normalized execution time for accessing a data size equal to 1, 2, 3, and 4 ways (128, 256, 384, and 512 KB) of the L2 cache.

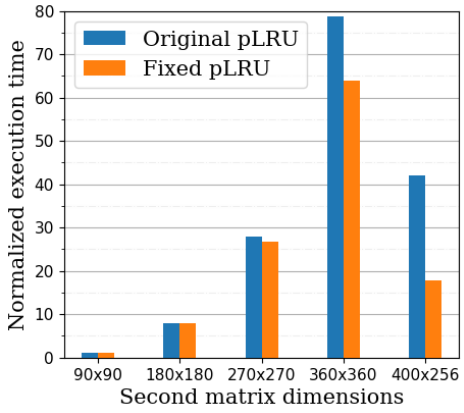

Fig. 5. Normalized execution time of matrix multiplications of square matrices of 90, 180, 270, and 360 row elements, analog to the accesses of 1 up to 4 ways of the L2 cache. The last column corresponds to the matrix multiplication of 100x400 by 400x256 elements. All bars have been normalized over the execution time of the 90 row element matrices multiplication.

C. Performance increase for processor programs

Benchmarks and other software programs benefit from the increased cache capacity with lower execution times. Fig. 4 shows that the maximum increase in performance achievable using benchmarks is 268 % when accessing to 512 KB, the total cache capacity. Meanwhile, in Fig. 5, matrix multiplication examples show a performance increase of 137 % on the matrix multiplication of 100x400 per 400x256 elements. Note that the matrix multiplication includes several noise sources (e.g., compiler), making it difficult to reach the maximum performance increase found by the benchmark case.

From this data, it could be argued if the bug could be found with core accesses only. However, note that there are several components between the core and the design fault (e.g. L1 cache), making difficult the isolation task which the SafeTI black-box study avoids. Along this argument, it may be possible to debug the faulty pLRU with Gate-Level Simulators (GLS), however, they carry high time cost when compared with FPGA debugging with a SafeTI module.

V. Related Work

Traffic injection has been used at different scopes and for different purposes. The remaining of this section reviews the main directions followed in the literature.

a) Performance simulators: . Performance simulators such as Gem5 [5] model Networks-on-Chip (NoCs), typically integrating other tools (e.g., Garnet [16] interconnect model in the case of Gem5). Other tools with a similar purpose to that of Garnet are GNoCSim [7] and Synfull [4]. While those tools are useful to perform software simulations of NoC models, they cannot be integrated with VHDL or Verilog designs and be deployed on an FPGA. Moreover, their simulation speed is orders of magnitude slower than SafeTI injection, which occurs at speed. Hence, those other tools cannot be used for IP validation as we do in our work with SafeTI.

b) Circuit simulators: . Questasim [22] and Verilator [23] are examples of circuit simulators enabling direct traffic injection on the IPs. While they offer even higher flexibility than SafeTI, they can only afford to inject very limited traffic patterns owing to their several orders of magnitude slower speed than ASIC designs or FPGA models. Moreover, issues appearing only after synthesis, like the one spotted in our work, could not be reproduced with circuit simulators.

c) Hardware Emulated Injectors: . Within this category, we can find SafeTI, Xilinx's Advanced eXtensible Interface (AXI) Traffic Generator (ATG) [24], and a trace injector part of the H2020 Mont-Blanc 2020 project [3]. Those IPs allow injecting traffic at speed with varying degrees of flexibility. To our knowledge, SafeTI is the only open source among those, and its high flexibility for programming could be, at most, matched by those other tools.

d) Software-based Traffic Injection: . Software tests have also been developed for traffic injection. A strand of works aims at developing stress benchmarks for performance validation [10], [11], [13]. While those tests can be run at speed, the limited controllability they have of some components, such as L2 caches, whose accesses are filtered by the L1 caches, and the inability to inject specific traffic patterns synchronously (e.g., long bursts such as those coming from DMA or Ethernet interfaces) make software stressing tests less powerful than SafeTI.

In this category, we can also find the so-called Software Based Self Tests (SBST) [14], [19]. Those software tests are intended to validate the functional behavior of some MPSoC components, but again, they suffer from the same limitations as stress benchmarks when compared against SafeTI.

VI. Conclusions and Future Work

Hardware component and integration validation in high-performance MPSoCs is a difficult challenge owing to the limited controllability that can be exercised with software-only means. Hardware traffic injectors, such as the SafeTI, offer new opportunities to perform such testing at speed and with high degrees of controllability.

We have presented the application of the SafeTI for the validation of the L2 cache of a safety-critical space MPSoC. Our black-box testing approach exploits the flexibility and programmability of the SafeTI to inject varying data patterns and collect detailed troubleshooting data. Our validation methodology using the SafeTI has identified a challenging non-determinisitic design flaw. Information has been shared with the IP vendor, which ultimately will allow fixing the design in a future release bringing increased value to the community since both, our SafeTI and the L2 cache are open-source components [1], [6].

References

[1] Frontgrade Gaisler AB. Frontgrade Gaisler AB GRLIB public GPL releases. https://discourse.grlib.community/c/general-grlib-soc/. Last accessed:: 17 May, 2023.

[2] Frontgrade Gaisler AB. Frontgrade Gaisler AB SoC GRLIB IP library. https://www.gaisler.com/index.php/products/ipcores/soclibrary. Last accessed:: 17 May, 2023.

[3] A. Armejach et al. Mont-blanc 2020: Towards scalable and power efficient european hpc processors. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 136–141, 2021.

[4] M. Badr and N. E. Jerger. Synfull: Synthetic traffic models capturing cache coherent behaviour. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 109–120, 2014.

[5] N. Binkert et al. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[6] Barcelona Supercomputing Center. Traffic injector SafeTI open IP repository. https://gitlab.bsc.es/caos_hw/hdl_ip/bsc_safeti. Last accessed:: 18 January, 2023.

[7] Universitat Politècnica de València. Gnocsim. a cycle-accurate wormhole-based network simulator. http://www.gap.upv.es/index.php?option=com_content&view=article&id=72&Itemid=108.

[8] Universitat Politècnica de València. SELENE: Self-monitored Dependable platform for High-Performance Safety-Critical Systems. https://www.selene-project.eu/. Last accessed:: 22 November, 2022.

[9] European Cooperation for Space Standardization (ECSS). ECSS Active Standards. https://ecss.nl/standards/active-standards/.

[10] G. Fernández et al. Assessing time predictability features of Arm Big. LITTLE multicores. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 258–261, 2018.

[11] M. Fernández et al. Assessing the suitability of the ngmp multi-core processor in the space domain. In *Proceedings of the Tenth ACM International Conference on Embedded Software*, EMSOFT '12, page 175–184. Association for Computing Machinery, 2012.

[12] Frontgrade Gaisler. NOEL-V Processor webpage. https://www.gaisler.com/index.php/products/processors/noel-v.

[13] S. Girbal et al. METrICS: a Measurement Environment For Multi-Core Time Critical Systems. In *ERTS 2018*, 9th European Congress on Embedded Real Time Software and Systems (ERTS 2018), 2018.

[14] D. Gizopoulos et al. *Software-Based Processor Self-Testing*, pages 81–155. Springer US, Boston, MA, 2004.

[15] International Standards Organization. *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.

[16] Synergy Lab. Garnet. An on-chip Network Model for Diverse Interconnect Systems. https://synergy.ece.gatech.edu/tools/garnet/.

[17] NVIDIA Corporation. NVIDIA DRIVE PX. Scalable supercomputer for autonomous driving. http://www.nvidia.com/object/drive-px.html.

[18] J. Perez-Cerrolaza et al. Multi-core devices for safety-critical systems: A survey. *ACM Comput. Surv.*, 53(4), 2020.

[19] M. Psarakis et al. Microprocessor software-based self-testing. *IEEE Design & Test of Computers*, 27(3):4–19, 2010.

[20] D. Radack et al. (Rockwell Collins). Civil Certification of Multi-core Processing Systems in Commercial Avionics, 2018.

[21] O. Sala et al. Safeti: a hardware traffic injector for mpsoc functional and timing validation. In *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2021.

[22] Siemens. Questa Advanced Simulator, 2020.

[23] Veripool.org. Verilator, 2020.

[24] Xilinx. *AXI Traffic Generator v3.0. LogiCORE IP Product Guide*, 2019.

[25] Xilinx Virtex UltraScale+ FPGA. VCU117 evaluation kit. https://www.xilinx.com/products/boards-and-kits/vcu118.html.