Contribution ID: **171**

Type: **Oral**

Towards Single-Event Upset detection in Hardware Secure RISC-V processors

Tuesday 3 October 2023 15:20 (20 minutes)

An increasing interest is growing towards reconfigurable processing systems embedded on the detector ASICs. Explorative work has been carried out to investigate Single-Event Upset (SEU) rates in open source RISC-V processors. The Ibex RISC-V core includes hardware security features that could detect SEUs in the core and alert the System-on-Chip (SoC) for possible malfunctions. This research addresses the possibility to rely on such hardware secure extensions for radiation hardness assurance.

Summary (500 words)

Single-Event Effects and hardware security show close similarities in terms of vulnerabilities and mitigation techniques. Secure processors address external physical attacks such as external laser stimulation to compromise the program and extract sensitive information from the systems. To overcome this vulnerability, hardware secure architecture extensions are often included in modern processor cores. These include dual program counters, EDAC on the register files and memories and even lock stepping in the CPU pipeline. Such features are highly similar to SEU protection done in spacecraft processing systems such as the LEON-FT and NOEL-V processors, commonly used in satellites. Provided by the limited design resources often found in space or high-energy physics experiment design teams, this paper addresses to which extend hardware secure architectures can be a reliable source to detect SEUs in the processor.

The Ibex open source RISC-V processor has been evaluated in simulation with extensive fault injection. The Ibex core has a hardware security option which can be enabled. This extensions enables a dual CPU pipeline, HSIAO code in the register file and the main instruction and data memories. Although the core does not intend to perform error correction, it provides 3 alert signals that flag possible security issues due to external attacks, henceforth compromising the program flow.

A simulation environment was developed using CoCoTB that includes the Ibex RISC-V instance along with several python models to model the SoC, such as memories, monitors, IO, etc. Fault injection was performed using an initial synthesis pass with Cadence Genus to list all flip-flops in the design. Random bit flips are stimulated from a CoCoTB coroutine. A monitor coroutine monitors critical CPU signals such as memory ports, register file, program counter, status registers, etc. Each cycle, a CRC is calculated on all these ports, providing a signature. Before SEU injection, a golden reference simulation is performed to calculate the correct CRCs on a cycle basis. Afterwards, during SEU injection, the CRCs are continuously compared.

The testbench checks if the Ibex alert signals are asserted if an error is encountered in the CRC check. If the core can successfully detect malfunction, it can be mitigated at system level later on. However, undetected errors can lead to problematic operation. During the tests, the Dhrystone benchmark was ran on the core.

Fault injection results showed that 10 % of all injected SEUs led to an error on the CPU (note that the memories were not stimulated). For SEUs that didn't lead to an error only 4% showed a false positive alert signal. For SEUs that did lead to an error, 88 % was detected by the CPU alert signals. The register file as well as the lockstep core are all detectable. Only the instruction fetch and load store unit can only detect 54% and 36% of all errors respectively.

Based on these results, we can conclude that relying on existing hardware security architectures could be a viable alternative to developing a custom SEU tolerant architectures.

Primary authors: Mr ENGELEN, Boris (KU Leuven); Prof. PRINZIE, Jeffrey (KU Leuven); APPELS, Karel (KU Leuven); Mr MARIEN, Levi (KU Leuven); JONCKERS, Naïn

Presenter: Prof. PRINZIE, Jeffrey (KU Leuven)

Session Classification: Programmable Logic, Design and Verification Tools and Methods

Track Classification: Programmable Logic, Design and Verification Tools and Methods