Trustworthy System-on-Chip by monitoring system behavior at runtime

Martin Flasskamp, Christian Klarhorst, Jens Hagemeyer
Cognitronics & Sensor Systems
Bielefeld University
Bielefeld, Germany
{mflasskamp,cklarhor,jhagemey}@techfak.uni-bielefeld.de

***Abstract*—Electronic systems form the backbone of the digital economy and therefore have an influence on many areas of people's work and lives. Exploring novel self-monitoring components by combining known methods and machine learning, so that they can assess their own trustworthiness is important. In this work, a Trust Monitoring Unit for detecting abnormal SoC behavior is proposed. The first two components to capture the SoC behavior at runtime were implemented. The components were evaluated on two FPGA based platforms and in software. The preliminary results show that bus transactions characterize the SoC behavior. In ongoing work captured behavior will be used by hand-crafted and AI-based methods to generate rules that identify abnormal behavior.**

***Index Terms*—SoC, runtime monitoring, behavior-based trust**

I. Introduction

Current microchips are complex systems composed of various interacting components. Ensuring that a microchip is trustworthy requires guaranteeing that it operates according to its specifications and is not compromised by hostile attacks or unexpected errors. Trustworthiness can be strengthened through monitoring, whereby the chip's behavior is continuously monitored to detect and respond to deviations. Complex systems can be monitored through a variety of techniques, such as by verifying behavioral patterns or by comparing to valid system behavioral specifications. The precision of specifying valid system behavior is a crucial factor in monitoring microchips, as it can ensure that deviations from a known pattern can be detected. It is also important that methods for detecting deviations provide appropriate measures for reacting to these deviations.

In this work, we address the following attack scenarios: 1) components compromised by a security vulnerability in their implementation, 2) changes in the behavior of a (purchased) IP component after production and 3) abnormal behavior due to counterfeiting (e.g. less than the required performance). The fundamental idea is that all of these attack scenarios cause a change in the communication behavior on the bus. Therefore, supervising the bus and checking the different aspects of the communication is ideal for detecting a broad range of attack scenarios. Those aspects include the communication pattern (size of the accesses, frequency), the order of the communicating bus participants and the current status of the system (e.g. booting) during the bus activity. We address this challenge by proposing a *Trusted Monitoring Unit* (TMU) that monitors and secures the system state of an SoC at runtime. In addition, a deeper understanding of the system can be derived from the interaction of the IP components.

The project on which this report is based was funded by the German Federal Ministry of Education and Research (BMBF) under the sponsor number 16ME0284. The responsibility for the content of this publication lies with the author.

II. Related Work

In recent years, several research efforts have focused on developing techniques to improve the security and trustworthiness of SoCs. The concept of a behaviour-based trust model aims to determine the trustworthiness of an entity in a system of cooperating entities by knowledge of previous interactions. In [WB06] Weth and Böhm present a framework for behavior-based trust models. They describe how to formulate behavior-specific knowledge about an entity from previous interactions and propose an algebra to formulate trust-policies. This algebra can then be queried to evaluate the trustworthiness. In our work, we plan to adapt this approach to SoCs. Entities in SoCs like CPUs, memories or accelerators interact with each other via communication infrastructure. Besides hand-crafted trust-policies we're working on methods to create AI-based policies. Different approaches exist to improve the trustworthiness inside an SoC. These include hardware-based monitoring of the systems and software-based techniques such as run-time verification. A noteworthy example for hardware-based monitoring are IP components developed by UltraSoC (acquired by Siemens in 2020) included in their embedded analytics suite. Another hardware-based approach are the extensions to the bus fabric by the ARM TrustZone architecture[Nga+16]. ARM provides the hardware logic to distinguish whether access to a resource is coming from the "secure" or "normal" world.

III. Implementation of the TMU

The TMU consists of the components *Probe*, *Monitor*, *Detect* and *Respond*. Its implementation is done in the migen framework [Mig], a Python toolbox for building complex digital hardware.

The *Probe* component captures the signals of interest from the SoC. Possible sources are among others the SoC interconnect bus, built-in self-tests (BIST) or status signals from

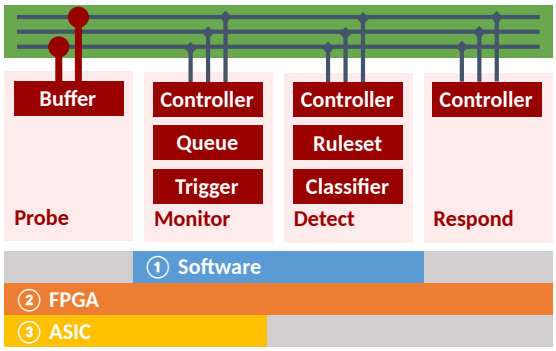
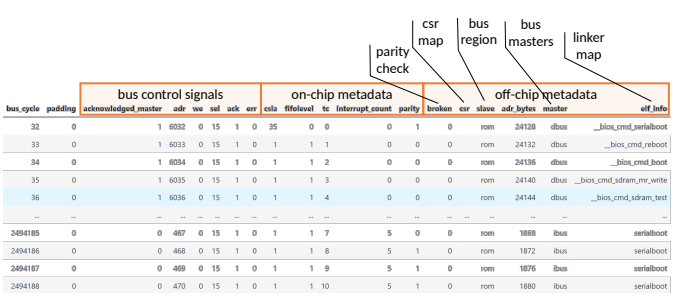


Fig. 1. Architecture of the Trusted Monitoring Unit and coverage of its components by the three planned demonstrator platforms.

IP components. To probe the SoC interconnect, various bus control signals are captured depending on the particular bus implementation. Our hardware demonstrator platforms (see IV-A) support AHB and Wishbone buses. Internally the probe component buffers the captured data.

The *Monitor* component combines the captured signals and merges them to the SoC's system behavior. Via a control input, the component can be activated to start triggering on input from the probe buffer. Incoming data is stored in a queue for further processing on-chip or to be transferred off-chip. To minimize resource usage, lossless compression methods of bus transactions are implemented: Accesses to the same or consecutive addresses are combined to an access group, either for rising or falling access patterns. In addition, it is planned to integrate sophisticated methods to identify more complex access patterns. For bus transactions supplementary metadata in addition to the bus control signals is added on-chip: a transaction counter or parity information to ensure data integrity (cf. Fig. 2 for wishbone bus). The recorded bus communication can be transferred off-chip for further analysis. That allows to extend the data-set by additional information from the software build environment, e.g. the linker or memory maps.



bus_cycle	padding	acknowledged_master	adr	we	sel	ack	err	csla	fifolevel	tc	interrupt_count	parity	broken	csr	slave	adr_bytes	master	elf_info
32	0	1	6032	0	15	1	0	35	0	0	0	1	0	rom	24128	dbus		__bios_cmd_serialboot
33	0	1	6033	0	15	1	0	1	1	1	0	0	0	rom	24132	dbus		__bios_cmd_reboot
34	0	1	6034	0	15	1	0	1	1	2	0	0	0	rom	24136	dbus		__bios_cmd_boot
35	0	1	6035	0	15	1	0	1	1	3	0	0	0	rom	24140	dbus		__bios_cmd_sdram_mr_write
36	0	1	6036	0	15	1	0	1	1	4	0	0	0	rom	24144	dbus		__bios_cmd_sdram_test
..	...	..	..	..	..	..	..	..	..	..	..	..	...	..	..	..		..
2494185	0	0	467	0	15	1	0	1	1	7	5	0	0	rom	1868	ibus		serialboot
2494186	0	0	468	0	15	1	0	1	1	8	5	1	0	rom	1872	ibus		serialboot
2494187	0	0	469	0	15	1	0	1	1	9	5	1	0	rom	1876	ibus		serialboot
2494188	0	0	470	0	15	1	0	1	1	10	5	1	0	rom	1880	ibus		serialboot

Fig. 2. Additional metadata added to bus control signals on-chip and off-chip.

The *Detect* component is ongoing work and will be responsible for observing the system state. A classifier processes the system behavior from the monitor queue and outputs its result. The decision from the classifier is either derived from a predefined rule-set or the whole classifier is implemented by machine learning methods. This rule-set has to be generated in advance, hand-crafted using expert knowledge about the hardware platform and the running application. Otherwise, the rules can be determined by machine learning methods running on training data created via simulation of a prototype.

The *Respond* component is planned to react on the classification done by the detect classifier. Depending on the target platform, a valid reaction in case of a detected loss of trust in the system state might be to enter a platform specific safe state (for safety critical systems) or a reset of the whole system.

IV. Analysis

Several demonstrator platforms of the TMU are implemented in software and on an FPGA. An ASIC implementation for specific TMU components is planned. Figure 1 depicts the coverage of the TMU components in each demonstrator platform. The platforms are used to capture data for the upcoming development of the detect component.

A. Hardware Demonstrator Platform

The probe and monitor components are compatible with the AIRISC Core Complex[Fra22] and the LiteX framework[Lit]. The current demonstrator platforms include a virtual prototype using LiteX and Verilator for training data and rule-set generation, and two FPGA based platforms for evaluation and demonstration purposes. One uses AIRISC with an AHB bus on a Xilinx basys3 FPGA board, and the other one LiteX with a wishbone bus on an AMiRo mini robot platform[Her+16] with a Spartan6 FPGA shown in Figure 3. The LiteX SoC framework allows the use of different RiscV core implementations, as well as the integration of various IP cores. The AMiRo SoC currently consists of a VexRiscv CPU with 256 MByte RAM, a CAN controller, SD-Card, a camera interface and the TMU monitor component. Furthermore, it is planned to have a silicon-proven version of the monitor component as part of an AIRISC tape out.

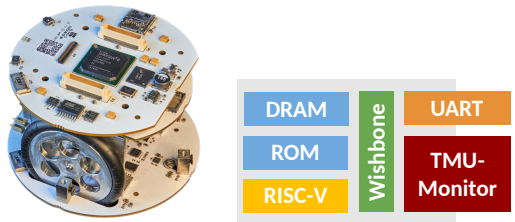


Fig. 3. FPGA demonstrator AMiRo.

B. Preliminary Results

For the preliminary results, we focus on two example programs, a small self-written example and CoreMark. Both applications were executed on AIRISC while the data bus was recorded with the help of the monitor component. The recorded transactions were then transferred to the host computer for further analysis. For each transaction, a total of 64 bit metadata was generated consisting of a 32 bit address, 1 bit indicating whether the transaction was a read or write, 3 bit recording the size of the transaction, an error bit to indicate

failed transactions and two 8 bit counter to encode the timing behavior of the bus for the idle and busy states. Additionally, 11 bits were used for the compression algorithm.

The first test consists of a bubble sort algorithm, with a uniform continuous access pattern, and an algorithm that sums equally spaced elements, with a non-continuous accesses pattern. The code listing of the second algorithm is shown in Figure 4. The recorded transactions on the data bus for the execution of both algorithms are shown in Figure 5. The monitor component recorded a total of 30399 data bus

```
for (int i = 0; i < 100; i++) {
    counter = (counter + 43) % 25;
    sum += array[counter];
}
```

Fig. 4. Summing example that produces a non-continuous access pattern.

transactions (3 % of the accesses were writes). The transactions were compressed inside the monitor by a factor of 13. Thus, the total compressed record size is 4.7 kByte. Both algorithms show different bus behavior and can therefore be distinguished from each other.

The second program is EEMBC CoreMark[EEM] because it consists of common algorithms found in other applications, e.g. list processing and matrix manipulation. Figure 6 shows the bus transactions while the CoreMark program was executed. The monitor component recorded a total of 52394 data bus transactions (5 % of the accesses were writes). The compression factor was 9, with a total compressed record size of 11.6 kByte. The different phases of the CoreMark program can be seen in the figure because of their different access behavior. The figure only shows the accesses to the memory region of the SoC, although, there were also accesses to a timer and the UART component.

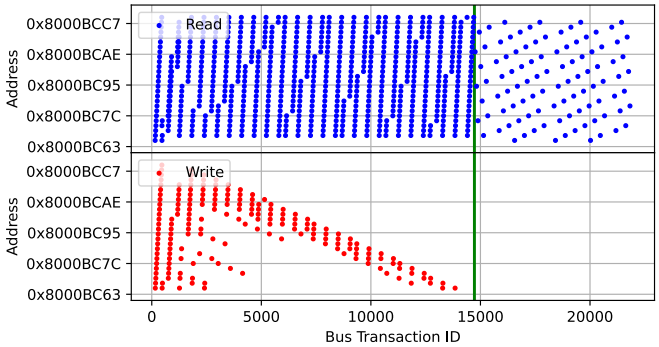


Fig. 5. Bus transactions to the memory region on AIRISC running bubble sort (left of the green line) and the summing example (right of the green line).

V. Conclusion

We implemented the probe and monitor components of the TMU in Software and on FPGA. We're able to capture the SoC behavior while running various programs. The preliminary results in section IV-B show that bus transactions form a

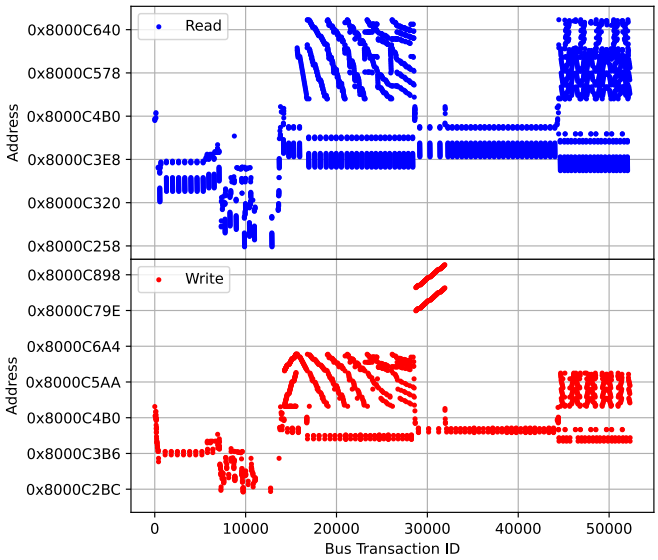


Fig. 6. Bus transactions to the memory region on AIRISC running CoreMark.

characteristic pattern in the address space depending on which algorithm is executed on the CPU. In ongoing work these patterns are used to derive a rule-set for detecting abnormal behavior of a SoC in the detect component. Hand-crafted and AI-based methods to generate the rule-set will be evaluated.

Acknowledgment

We thank Bjarne Wintermann for his support on implementing the TMU component within his work as a student assistant.

References

[WB06] Christian von der Weth and Klemens Böhm. "A Unifying Framework for Behavior-Based Trust Models". In: Oct. 2006.

[Her+16] Stefan Herbrechtsmeier et al. "AMiRo: A modular & customizable open-source mini robot platform". In: *ICSTCC 2016*. 2016.

[Nga+16] Bernard Ngabonziza et al. "TrustZone Explained: Architectural Features and Use Cases". In: *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. 2016.

[Fra22] Fraunhofer IMS. *The Fraunhofer IMS AIRISC RISC-V Processor*. Version 1.1.0. Dec. 2022. URL: https://github.com/Fraunhofer-IMS/airisc_core_complex.

[EEM] EEMBC. *Embedded Microprocessor Benchmark Consortium*. URL: https://www.eembc.org/.

[Lit] LiteX. *SoC builder framework*. URL: https://github.com/enjoy-digital/litex.

[Mig] Migen. *A Python-based tool that automates further the VLSI design process*. URL: https://m-labs.hk/gateware/migen/.