Generating trustworthy hardware/software I^{2}C drivers for board management controllers

Daniel Schwyn, Zikai Liu, Timothy Roscoe
Systems Group, ETH Zurich

At the heart of every modern server platform sits an embedded system called a board management controller (BMC) that is responsible for the low-level functioning of the platform. Despite the critical nature of these systems, they are generally not built as trustworthy systems. In this talk, we will give an update on our efforts to cyber-retrofit BMCs. The focus will be on generating software/hardware I^{2}C drivers from model-checked specifications.

At the seL4 summit in 2022, we presented our vision for trustworthy BMCs with seL4 as a separation kernel. By employing the cyber-retrofit strategy, we can use an existing Linux-based BMC implementation (OpenBMC), and gradually migrate critical components into native seL4 tasks. One critical task of a BMC is to correctly configure the power and clock distribution on the platform. The formal model we have developed allows us to generate configurations and transition sequences for power and clock. Built on that, the BMC then needs to communicate the configurations to the regulators that are connected through chip-to-chip communication buses like I^{2}C. To preserve the guarantees of the generated configurations, we need to ensure that the communication is correct. We will give an update on the trustworthy BMC project overall but focus on our work to generate correct I^{2}C stacks from specifications.

Unlike memory-mapped devices which have one-to-one interfaces with the BMC, multiple I^{2}C devices may share the same bus, creating a group of devices interacting with each other. Quirks of one device can influence the correct communication of the whole assemblage. Therefore, to produce a correct driver stack for I^{2}C, we need to consider the specification of not only the I^{2}C controller but all devices on the bus.

In this talk, we will present Efeu, a framework that allows us to specify both the host-side driver and the peripherals. The entire system is then model-checked for interoperability using Spin[1]. This ensures that any quirks in the peripherals are correctly handled by the host-side communication stack and that the peripherals do not hazardously interfere with each other. The specifications are composed of layers, which enables swapping out a layer of the communication stack in which a device might have a quirk while reusing the specifications for the rest of the stack without code duplication. We envision building up a library of device specifications to amortize the specification effort across multiple systems.

From the specifications, we can generate implementations for the host-side stack. Efeu can generate both software and hardware implementations. The software implementations target seL4, but could also address other operating systems. The hardware implementations can be materialized on programmable hardware such as Field Programmable Gate Arrays (FPGAs). While BMCs used to be based on low-cost micro-controllers, the industry is increasingly adding computing power to them, including FPGAs[2].

Efeu can furthermore generate combined software/hardware stacks. The split point between hardware and software can be chosen at compile time. This allows the implementation to be adapted to system constraints. For example, if an existing hardware I^{2}C controller must be used (due to performance requirements for example), we can model it as a part of the specification and let the model checker assert its interoperability with the peripherals (possibly with quirks). If the checking shows that the system is not inter-operable, we may fall back on instantiating a bit-banging driver at the cost of performance. On the other hand, if there exists an FPGA on the BMC module, the hardware parts of the stack can be implemented on it. We can then choose a split point that optimizes for a given metric such as CPU usage, latency or FPGA utilization.

We evaluated Efeu-generated I^{2}C stacks on a Zynq MPSoC and show that generating the full communication stack from verified specifications is not only practical but that the resulting implementation can saturate the I^{2}C bus and achieve competitive performance with off-the-shelf solutions. We also show that by varying the split point between hardware and software, we can explore the trade-offs mentioned above and select the most appropriate split point for a given system.

Efeu closes the gap between the power manager and the hardware in our trustworthy BMC stack. We also believe that the methodology used in Efeu can be applied to other buses, the ultimate goal being PCIe.

[1] https://spinroot.com
[2] https://opensource.antmicro.com/projects/artix-dc-scm/