ASIL-D workload projection and optimization using PA Ultra and Virtualizer

Matthias Glück*, Robert Evert **

* Volkswagen Group Components, matthias.glueck@volkswagen.de

** MicroNova AG, robert.evert@micronova.de

Abstract: Current automotive development projects often focus on autonomous driving and steer-by-wire and therefore require higher functional safety requirements. Besides these higher ASIL-levels, additional challenges arise from shorter development cycles and continuous pressure on cost. To meet these conflicting requirements the software and the hardware architecture need to be jointly analyzed and optimized early on in the development process.

This presentation shows the early architecture analysis of an ASIL-D electric power steering application. First a workload model is generated by combining software execution traces from a previous generation physical and a virtual hardware prototype. The workload model is then used to explore several optimizations of the next generation software and hardware architecture under consideration of the overall safety goals. Short feedback loops enable fast development progress and decision making during these studies.

1 Introduction

The recent chip hardware crisis shows a high need of optimization to be able to use current hardware platforms to their full potential. In the process of hardware-software co-partitioning, the hardware is modified to the specific needs of the software. Meanwhile, the software is tailored to use special features of the hardware to achieve higher throughput, power efficiency or reliability. Rising demands of requirements, such as increasing ASIL-levels or cost cutting of the hardware platform, create a need for advanced modeling capabilities. [1]

The following article shows a setup for creating an abstract software model, called workload, early in the design process of a system with high functional safety standards. The model contains runtime information and analysis of the most common bottlenecks in execution behavior, but is otherwise non-functional. This model is executed on a virtual hardware platform called a VPU. The second key component is a level 4[1] virtual ECU that is ISO 26262-8 qualified in order to be used for ISO26262-6 compliant software testing. A real hardware counterpart also exists and may be used to generate software execution trace data on a real platform. In terms of the V-model[2] , this approach resides on the side in the area of hardware and software design and analysis.

The expected and achieved benefit is a very fast what-if analysis for software architecture evaluation and optimization that can be backported to the real software structure. Practical questions can include the possibility of running the software on a two-core system instead of three cores. Additionally, the results may impact the next generation of the hardware specification.

2 Virtual ECU Level 4

A key part of the project is the use of a level 4 virtual ECU. It allows the execution of the unmodified and un-instrumented software binary in a complete representation of the software environment including a bus simulation, sensors and actors. Timing information during instruction execution is implemented as thoroughly as possible. The system is used for gathering software execution data such as program flow traces or function traces, memory access data for all CPU cores and DMA transfers. Due to the structure of the distributed memory hardware, the access information should also include the memory regions that are being accessed. [2, 3]

Usage of debug hardware allows the creation of software execution traces as well, although the amount of data is limited due to limited bandwidth . By using standard file formats, the workload model can be created from either real hardware or virtual platforms. Table 1 gives an overview of data sources.

Table 1: Data acquisition sources for generation of the VPU and workload model.

	Real hardware	**Virtual hardware**	**Format**
Function trace	X	X	Binary or text file
Instruction trace	Limited time	Unlimited time	Binary or text file
Access to Data RAM	-	X	Text file
Access to Data Cache	Limited via performance counters	X	Text file

[1] https://www.prostep.org/fileadmin/downloads/WhitePaper_V-ECU_2020_05_04-EN.pdf

[2] V-model for mechatronic systems development (after Bender*, 2004), for example "PSI 11 - Smart Systems Engineering - Simulation Model Exchange", version 2.0, prostep ivip, https://www.prostep.org/fileadmin/downloads/prostep-ivip-Recommendation_PSI11_SmartSE_V2-0.zip

The device under test (DUT) is a AutoSAR compatible real-time software. It consists of approximately 50 tasks and runs on three cores. It makes heavy use of inter-core communication and peripheral access. Hardware interruptions cause a non-deterministic execution profile in reality through shifting task execution. Task priorities and several task periods in the range from 200 µs to 10 ms shape a complex execution pattern that needs to be modeled in the software model as thoroughly as possible.

3 Platform overview

In general, the abstract platform, like it is shown in Figure 1, consists of the following components:

- VPU - the abstract hardware platform
- Workload - represents a non-functional model of the executed software
- Mapping - combines the workload with execution units of the VPU
- Simulation scenario - runs a specific mapping and executes the workload model on the VPU

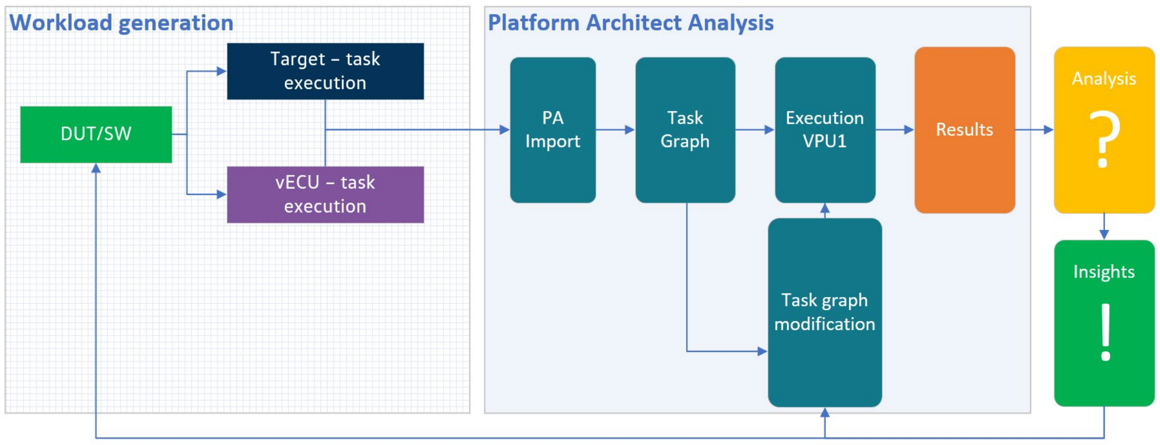

Figure 1: Optimization flow with target and virtual hardware data sources.

The platform consists of static parts that can be derived from the target hardware specifications. Statistical data parametrizes complex components such as busses and hardware such as Flexray or CAN controllers. Test data acquired from real measurements serves as a basis for re-creation of realistic behaviour in the model where applicable, mostly used for memory accesses from CPU cores and DMA transfers. Figure 2 shows an overview of the VPU structure. [5]

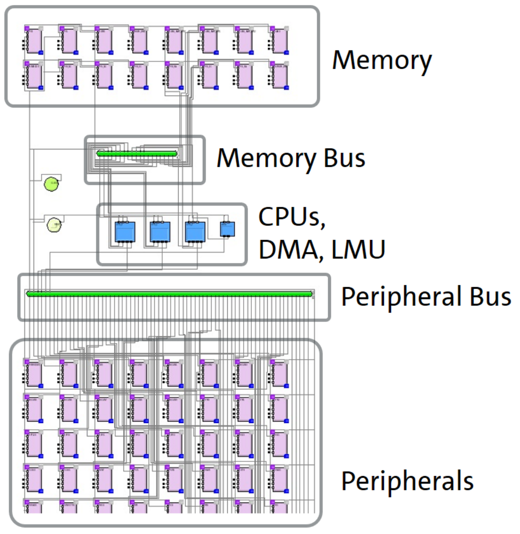

Figure 2: VPU implementation on Platform Architect.

The current implementation creates the abstract software model in the form of AutoSAR tasks. An implementation of a task scheduler is able to interrupt running tasks as they happen in real-time with software or hardware interruptions. Memory accesses are modelled in detail through files that include the individual access ports and the memory regions accessed. The delay resulting from these accesses is also modelled, so that bus congestion on the memory bus can be analysed later on. All steps are highly automated and can further be customized. The resulting task structure is called a task graph or workload model.

The task graph can be executed as it is as a standalone, although it makes more sense to run it on an abstract hardware platform. It is matched to this platform via a mapping description, that defines which task runs on which resource, either on a CPU core or a DMA controller. The results from this step is a software execution trace, similar to a trace that can be generated from the real hardware, for example in a BTF file format. Comparable to the vECU, all models inside the VPU can be traced as well. This allows the identification of bus-load bottlenecks due to memory or peripheral access limitations.

4 Possible Optimizations

The level of abstraction of the setup allows many different optimization procedures. Typical changes include changing the VPU or the model, where VPU changes resemble modifications to the hardware. Simple changes can be carried out if the hardware supports these changes like changing the clock or bus frequencies while other changes, such as a higher IPC, optimized instructions or the implementation of co-processors, have to be implemented by the hardware vendor. Changes to the workload include remapping of memory access across different cores, memory alignment or the task distribution across the different cores. The dynamic behavior of the model allows for examination of these changes in greater detail than a static analysis normally does. Typically, linear scaling of one parameter does not scale the performance of the rest of the system in a linear way due to additional constraints.

5 Examples

Workload characterization

Some optimizations can be identified as early as in the acquisition of the execution trace data stage. Identifying processing and memory access ratios for individual tasks provides an overview of the execution profile of the software. In this step optimization opportunities, such as alleviating bottlenecks in the memory access, can be determined. Furthermore, memory alignment can be checked according to the hardware platform as every relevant memory access is profiled.

Task restructuring analysis

An example for a simple what-if analysis is the remapping of a task from one core to another. The goal is to reach a better load distribution between the individual cores or optimize execution speed in the critical path. Additional constraints are task priorities and interruptions during software execution. Depending on the complexity of the task and modelling depth, remapping of memory accesses may be necessary in order to further optimize during this stage of the process. To analyze and solve these issues procedures such as a deadline violation analysis for real time systems can be performed.

Bus access optimization

Assuming a core (Core 0) performs an access memory read operation from another core (Core 1). This access is executed over the memory bus with a certain delay. Each signal goes through a decision process for placement in a memory region. This is done by a linker while compiling via configuration file or automatically without the knowledge of runtime information. In a typical use case, Core 1 additionally performs read or write accesses to its memory region. If the amount of accesses from Core 0 exceeds that from Core 1 it may make sense to move this memory region to a memory that is local to Core 0 as shown in Figure 3.

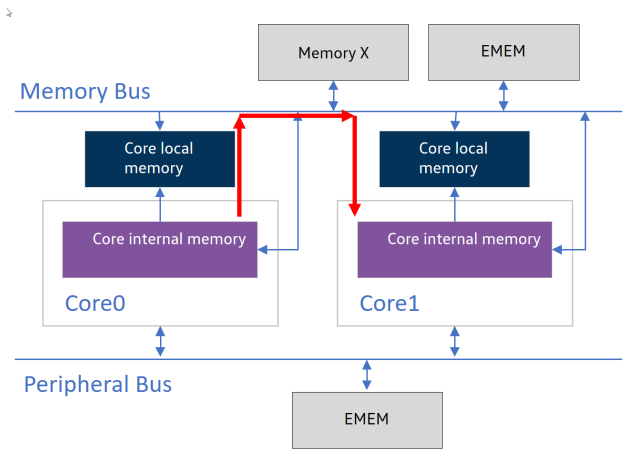

Figure 3: Example structure for connections of the CPUs and the memory- and peripheral bus.

These accesses are counted during the characterization phase of the workload. In a second step, a preliminary approximation of a possible gain is calculated to find better potential locations for memory region placement. Usually, the memory regions are not isolated but consist of inseparable complex data structures and a projection of the entire structure should therefore be performed. The platform allows to modify the model according to these results and make a projection of the achievable improvement. In the final step, this optimization can be ported to the real hardware, for example with a linker file. The software has to be recompiled and can be executed on the virtual prototype or flashed to the hardware. The improvement can be as significant as several percent, as has been shown for other link time optimization approaches too [4].

6 Testing impact

It is vital to note the discrepancy between prediction and reality in the analysis process. Changes in the software model may be relatively simple to implement but carrying the same changes on the hardware counterpart may prove to be nearly impossible. The shown analysis steps give hints for possible optimizations, but all changes still need to be reviewed by a formal validation process for architectural changes. In the end a full functional test needs to be performed on the modified software on traditional teststands like HiL-systems or virtual ECUs in order to make sure that the changes did not cause any safety impact nor violation.

7 Conclusion and outlook

Software abstraction is a way to use an abstract design process on the left side of the V-model. When performed in addition to the virtual and real hardware test stands, this procedure serves to close gaps in the testing process. Fast what-if analyses give a benefit to the user when the software design does not match the used hardware or quickly made changes of the requirements. The model is able to vary over several degrees of abstraction. Our current focus lies on optimizing memory transactions due to the fixed hardware implementation. Naturally, the model could be expanded to include more complex hardware for measurement acquisition or communications that are included in the workload. Initial optimization results show improvements of several percent CPU load with minimal changes to the software by using a static analysis of the memory alignment and a linker step optimization process. The current status does not yet enable optimization of the software to fulfill the 2 core lock-step requirement, although every optimization gives more headroom for higher safety margins or new features. The next steps include analysis of the instruction stream in order to gain deeper insights into the hardware platform.

8 Literaturverzeichnis

[1] TEICH, Jürgen. Hardware/software codesign: The past, the present, and predicting the future. Proceedings of the IEEE, 2012, 100. Jg., Nr. Special Centennial Issue, S. 1411-1430.

[2] Robert Evert, Andreas Drews und Stephan Schmidt , Effizient Absichern, Virtueller Prüfstand zum Verifizieren eines Lenkungssteuergeräts, https://www.elektroniknet.de/automotive/assistenzsysteme/virtueller-pruefstand-zum-verifizieren-eines-lenkungssteuergeraets.186126.html, abgerufen am 14.05.2021.

[3] KANG, Kyungsu, et al. Seamless SoC Verification Using Virtual Platforms: An Industrial Case Study. In: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2019. S. 1204-1205.

[4] PANCHENKO, Maksim, et al. Bolt: a practical binary optimizer for data centers and beyond. In: 2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). IEEE, 2019. S. 2-14.

[5] LU, Kuen-Long; CHEN, Yung-Yuan. SoC-Level Safety-Oriented Design Process in Electronic System Level Development Environment. Journal of Circuits, Systems and Computers, 2021, 30. Jg., Nr. 14, S. 2150254.