Microprocessors and Microsystems 38 (2014) 845–859

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Scalability evaluation of an FPGA-based multi-core architecture with hardware-enforced domain partitioning

Daniel Kliem *, Sven-Ole Voigt

Institute for Reliable Computing, Hamburg University of Technology (TUHH), Schwarzenbergstraße 95, 21073 Hamburg, Germany

ARTICLE INFO

Article history:
Received 29 March 2013
Revised 26 January 2014
Accepted 13 February 2014
Available online 22 February 2014

Keywords:
FPGA
MPSoC
Domain partitioning
Shared memory
Performance evaluation
Bus-centric architecture

ABSTRACT

There is a trend towards to dense integration of embedded systems for cost, weight, and power savings. Integration of multiple critical software functions in a single embedded platform requires domain partitioning. Groups of independent software functions exist in isolated domains to maintain individual functional correctness, even in presence of errors. Software solutions such as *Real-Time Operating Systems* (*RTOS*) with time and space partitioning are state-of-the-art segregation approaches. As an alternative to these existing solutions, we present a robust, reliable, and efficient architecture with segregation support for safety- and security-critical embedded systems. Our solution hosts different software functions on a platform with as few hardware components as possible: the *System-on-a-Chip* (*SoC*) approach.

The proposed architecture instantiates multiple self-contained soft processor systems on a single chip. The architecture offers hardware-enforced segregation and is completely transparent to software applications. We demonstrate this aspect by running multiple segregated instances of unmodified off-the-shelf Linux systems from a shared memory device. Since our architecture targets reconfigurable platforms, it is also flexible and can be tailored to application-specific needs at design time.

Segregation is achieved with a hierarchical connection of memory busses by secure bus bridges. The bridges perform caching, prefetching, and burst accesses to efficiently avoid temporal conflicts on shared resources. Hence, our secure bridges allow to use soft processors for critical designs.

We implement several prototypes and evaluate them by using novel bus observers for characterization of bus-centric architectures. Finally, we show the effectiveness of our implemented optimizations.

1. Introduction

Embedded systems are ubiquitous today. Cost-, weight-, and power-savings are key drivers for dense integration of embedded systems. At the same time there is also a demand for steadily increasing performance.

Consumer electronics, such as personal mobile devices, follow this trend. They show dense integration of multimedia and communication functions in battery operated devices.

In contrast to that, safety- and security-critical embedded systems traditionally follow the one-computer-per-function approach. With increasingly demanding functional requirements this approach leads to a multitude of networked *Electronic Control Units* (*ECUs*). For instance, state-of-the-art passenger cars are equipped with 20–70 ECUs [1].

So-called *domain partitioning* or *domain segregation* is an abstract concept at the system design level. Software functions are first grouped according to their safety and security classification. These groups of equally ranked functions are then treated as isolated domains. Domain partitioning preserves individual integrity of software functions and avoids error propagation across domain boundaries. Critical systems that are developed according to demanding safety requirements, such as avionics software, adhere to domain partitioning principles.

Domain partitioning implemented in a computing platform can integrate software functions of different domains on the same hardware. This reduces the number of necessary ECUs.

Typically, domain segregation of software functions is implemented as software solution as well. Such pure-software segregation solutions run on general-purpose processors and *System-on-a-Chip* (*SoC*) platforms. Software solutions are flexible. For instance the number of domains can be configured at the design time and is not directly determined by the processing hardware. However, since these software solutions use general-purpose devices, they lack an important advantage of embedded system design: co-design of hard- and software.

* Corresponding author. Tel.: +49 40 42878 3849.
E-mail address: d.kliem@tuhh.de (D. Kliem).

http://dx.doi.org/10.1016/j.micpro.2014.02.006
0141-9331/

Configurable platforms such as FPGAs offer benefits. Hardware resources can be tailored for each segregated domain on configurable devices. For instance, domains can use interface-combinations that are not covered by hard-wired general-purpose devices. Moreover, domain-specific accelerators are available. As with software solutions, the number of segregated domains is chosen by the system designer and is not preselected by the device manufacturer.

Configurable devices are flexible at the design-time of the system but are static at the run-time of the actual software (at least if dynamic reconfiguration is not used). This is a most wanted quality, since the design-time flexibility of the segregation solution is not traded for its robustness.

Configurable devices also allow to use soft processors. Different domain-specific architectures can be combined within the same device. However, despite their obvious advantages, soft processors on FPGAs are rarely used to host critical software functions. Clock frequencies of these soft processors are small in comparison to those of hard-wired devices. It is not expected that clock frequencies of synthesized logic on FPGAs will grow by an order of magnitude in the near future.

FPGA technology follows Moore's Law [7] and we can expect FPGAs to grow significantly in logic capacity. Devices with over one million *Lookup Tables* (*LUTs*) are available [8]. Luckily, current FPGA families provide wide and fast memory attachments, mostly implemented as hard macros that are faster than configurable logic (Table 1).

To overcome this performance gap, we propose and evaluate an architecture that combines the specific needs of partitioned software with the flexibility of reconfigurable hardware. Recent FPGAs can accommodate many processing cores in parallel, which operate from a single shared memory [9,10].

Our architecture instantiates multiple segregated and self-contained systems. Furthermore, it shares available memory bandwidth among the systems in a predictable and scalable way. Secure bus bridges are used to form a segregated hierarchy of memory busses. The architecture uses soft processors for safety- and security-critical functions and reaches high assurance levels with less effort.

Segregated systems must consider disjunctive resource usage in both the temporal and the spatial dimension. Given the flexibility of FPGAs, it is trivial to achieve spatial separation: Processors and interfaces are implemented as truly separate SoCs on a single programmable chip and memory overlaps are eliminated by address translation. Hence, we focus on mitigation of temporal conflicts and present several optimization techniques that deepen and extend our previous work on this topic [11].

The following sections provide an overview of current research (Section 2) and established segregation solutions (Section 3). As an alternative to these solutions, Section 4 presents our proposed architecture and Section 5 introduces our prototyping environment. Two secure bridge designs, our main contributions, are then presented in Section 6. The next sections introduce a dedicated bus observer (Section 7) for scalability evaluation of the bridge designs (Section 8) and present an outlook on further improvements (Section 9). Finally, Section 10 provides a summary of our contributions.

Table 1
Clock speed discrepancy (Vendor Data).

CPU/Memory attachment	FPGA family	Max data width (bits)	Clock rate (MHz)
Altera Nios II/s [2]	Stratix IV	32	240
Gaisler LEON3 [3]	Virtex 5	32	125
Xilinx Microblaze [4]	Virtex6(-3)	32	303
Altera DDR3 [5]	Stratix IV	144	533
Xilinx DDR3 [6]	Virtex6(-3)	144	533

2. Related work

We are aware of one architecture with similar objectives. Nojiri et al. present an ASIC implementation that aims at domain partitioning for embedded multi-core processors in the automotive industry [12].

Their architecture prototype comprises a dual-core system with a shared L2 cache. They propose a central bus structure with distributed *Physical Partition Controllers* (*PPCs*). The PPCs serve as a hardware configurable MMU, which restricts the access to peripheral devices and partitions the available memory. In case of access violations, a special exception handler is invoked.

Currently their approach is limited to two different domains – a real-time and an IT-domain. Even with larger PPC-enabled devices, the number of domains would still be preselected during expensive design of the ASIC. The use of configurable devices, as presented here, allows the system designer to determined the number of domains.

We target applications with multiple different domains and assume that fine-grained segregated systems are more easily developed, verified, and maintained. Typically, multiple operational functions shall reside in segregated domains alongside with maintenance and health monitoring functions. That is, we distinguish finer than real-time versus non-real-time.

With our concept, it is even possible to change the design during the system life-cycle. This offers the system designer the opportunity to cost-effectively deal with late requirement changes. An ASIC design process does not provide such a flexibility.

Their architecture fulfills the requirements of dense integration and provides hardware enforced spatial segregation. They do not focus on temporal segregation of the central bus structure, which means that they only deal with one aspect of segregation. Especially the shared L2 cache of the CPUs is a potential source of non-determinism at run-time.

Our architecture provides segregation at system level instead of CPU level. These segregated systems have exclusive access to their own local resources. Moreover, we especially target configurable platforms that offer more degrees of freedom during design. System designers can tailor local systems to their individual applications and they can include robust dedicated communication channels across the domains.

We conclude that both approaches provide robust domain segregation, at least in the spatial dimension, but follow a different strategy. Nojiri et al. aim at a mass-market general-purpose partitioning solution as required by automotive designs. In contrast to that, our approach targets low volume but long life-time systems that are typically found in avionics.

3. An overview of existing segregation solutions

A typical scenario for segregation is to host different software components on the same platform. The involved software components have to meet their individual functional and real-time requirements although they share resources, e.g., the same CPU. A shared platform is beneficial to reduce recurring hardware costs in mass-production markets (e.g., in automotive industry) or to reduce weight (e.g., in avionics applications).

There are different methods to establish segregation in a control unit. The obvious approach uses one processor per function at board-level. Board-level duplication offers separation but needs multiple devices and is inflexible in terms of future design modifications.