Noname manuscript No.
(will be inserted by the editor)

Application-Specific Tailoring of Multi-Core SoCs for Real-Time Systems with Diverse Predictability Demands

Steffen Vaas · Peter Ulbrich · Marc Reichenbach · Dietmar Fey

Received: date / Accepted: date

Abstract Embedded multi-core processors improve performance significantly and are desirable in many application-fields. This development, in particular, includes safety-critical real-time systems, which typically require a deterministic temporal behavior. However, even tasks without dependencies running on different cores can interfere due to, sometimes hidden, shared hardware resources, such as memory or communication buses. Consequently, only a pessimistic assumption of the *worst-case execution time* (WCET) that incorporates interference can be given. The desired performance gain therefore evaporates in the poor temporal analyzability.

Safety-critical real-time systems are typically composed of multiple tasks with varying criticality levels and requirements on predictability and performance, respectively. In this paper, we present an approach that generates an application-specific, deterministic multi-core architecture for such mix-critical systems, thus eliminating the aforementioned hardware-induced interferences in the first place. Safety-critical tasks with stringent temporal requirements are mapped to dedicated *Deterministic Execution Units* (DEUs) while the remaining soft real-time tasks co-reside on a general purpose multi-core processor that offers performance over determinism. Just as well, predictable interconnections between DEUs are generated to satisfy dependencies and precedence constraints. Consequently, timing analysis for hard real-time tasks is significantly simplified, since interferences caused by shared resources and scheduling are finally eliminated. To show the benefits of our approach, an application-specific architecture for a flight controller was generated and compared to an *ARM Cortex-A9* dual-core as a reference. Overall, we were able to significantly improve temporal properties of safety-critical tasks while preserving the overall performance for soft real-time tasks.

This work is supported by the German Research Foundation (DFG) under grants no. SCHR 603/9-2, the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR89, Project C1), the Bavarian Ministry of State for Economics under grant no. 0704/883 25 (EU EFRE funds) and by the Bavarian Research Foundation (BFS) as part of their research project "FORMUS^{3}IC".

Steffen Vaas
Friedrich-Alexander-University (FAU) Erlangen-Nürnberg, Martensstr 3., 91058 Erlangen, Germany
Chair of Computer Science 3 - Computer Architecture
Tel.: +49-9131-85-27028
Fax: +49-9131-85-27912
E-mail: steffen.vaas@fau.de

Peter Ulbrich
Friedrich-Alexander-University (FAU) Erlangen-Nürnberg
Chair of Computer Science 4 - Distributed Systems and Operating Systems
E-mail: peter.ulbrich@fau.de

Marc Reichenbach
Friedrich-Alexander-University (FAU) Erlangen-Nürnberg
Chair of Computer Science 3 - Computer Architecture
E-mail: marc.reichenbach@fau.de

Dietmar Fey
Friedrich-Alexander-University (FAU) Erlangen-Nürnberg
Chair of Computer Science 3 - Computer Architecture
E-mail: dietmar.fey@fau.de

1 Introduction

A characterizing feature of real-time systems is that they have to react to physical events within given deadlines. These can be more or less strict, for which

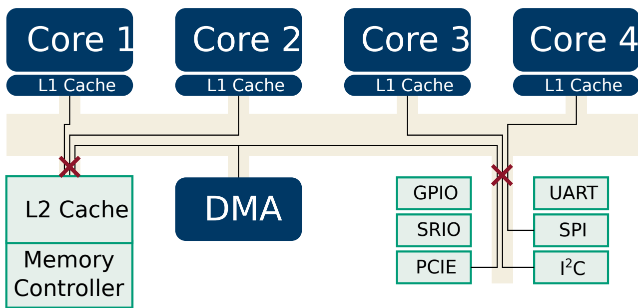


Fig. 1 The structure of an exemplary multi-core processor architecture: tasks running on different cores share hardware resources, as the common memory controller, which leads to contention. Even if tasks do not access to same peripherals, interferences occur caused by the common bus system.

reason real-time systems are typically classified into hard and soft real-time (i.e., high and low criticality). In a car, for example, the airbag always has to meet hard timing constraints, whereas missed deadlines in the navigation system are ultimately just annoying.

Therefore, a fundamental requirement in real-time system engineering is the sound estimation of *worst-case execution times* (WCETs) for safety-critical, hard real-time tasks. However, contemporary processors typically pack a whole series of performance-enhancing measures, such as pipelines, out-of-order execution, and tiered memory architectures jeopardizing this aim. To give an example, a simple addition may take between 3 (best) and 321 cycles (worst case) in a 2001 PowerPC 755 [6]. Consequently, any sound WCET analysis will be pessimistic to a certain extent and provide only over-approximations (upper bounds) of the actual WCET. In the past, real-time applications of different criticality (i.e., hard and soft) were either treated equally according to the higher criticality or partitioned on dedicated execution platforms. For example, in a car, each function is rendered by a single *electronic control unit* (ECU), which made it relatively easy to isolate and satisfy timing and safety requirements. Recently two important factors have changed: On the one hand, autonomous driving is just around the corner, making software much more complex as well as more parts safety-critical. On the other hand, multi-core embedded processors have hit the market, supplying an amount of processing power unthinkable in the past. Market pressure forces the industry to seize this opportunity by consolidating ECUs. Therefore, software of lower safety requirements now co-resides with applications of high safety demands, thus increasing interference, complexity and the certification hurdle occur at the same time.

Although considerable progress has been made in WCET and especially hardware analysis for single core real-time systems, multi and many-core systems still represent an unresolved challenge to be tackled. Main issues are interferences caused by inter-task dependencies as well as shared resources and, even worse, hidden yet inherent interdependencies within the actual hardware architecture, as illustrated in Fig. 1. Overall, these massively increased interferences typically bloat analysis estimates to the point where they become unfeasible in practice.

Consequently, the real-time community pursues various other ways to deal with the problem: with partitioning, novel scheduling techniques, and deterministic hardware being the most prominent representatives. The first, partitioning, indeed is the most intuitive and widespread approach. It is, for example, state-of-the-art in the automotive domain, where individual functions are tied to dedicated cores. This works well, as long as the problem fits the hardware granularity. Another approach is to pigeonhole the cumbersome WCET over-estimates by resorting to mixed-criticality scheduling. Here, optimistic WCET estimates are used as long as budgets are not exceeded. Otherwise, the system gradually switches to hard upper bounds for critical tasks with hard deadlines by omitting less critical tasks to mobilize resources. Although being a captive concept, it requires multiple WCET estimates per task, which can be tricky to acquire, and restricts the verifiability in practice due to the added scheduling complexity. Finally, deterministic architectures, such as PATMOS or PRET, can be used to simplify the WCET analysis again. However, this is also accompanied by a substantial loss in performance due to the absence of performance-enhancing hardware techniques as mentioned earlier.

1.1 Problem Statement

Recent developments in *field programmable gate arrays* (FPGAs) and *systems on a chip* (SoCs) allow for a further approach: Instead of relying on *commercial off-the-shelf* (COTS) hardware, the architecture can be specifically tailored to meet the system's needs. In earlier work [21,22], we have investigated the automated partitioning and tailoring of automotive real-time systems on dedicated, deterministic soft-cores within contemporary FPGAs to the benefit of determinism and analyzability. In principle, FPGAs permit various architectural approaches as illustrated in Fig. 2: in this example, a task set is composed of safety-critical and non-critical

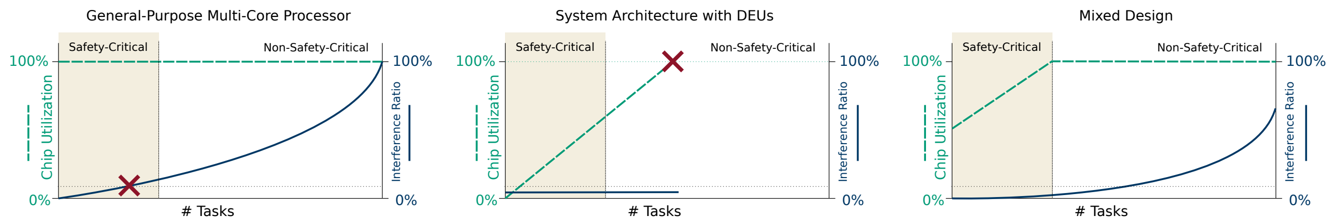


Fig. 2 Illustrative comparison of architectural approaches to mixed-criticality real-time systems. In this example, task priorities decrease from left to right, with subsets for critical and non-critical tasks. Relying on general-purpose (left) or deterministic cores (middle) only violates predictability and performance requirements, respectively. The proposed mixed design approach (right) combines both aspects.

tasks with high and low requirements on hardware determinism, respectively. By utilizing the entire FPGA to implement a general-purpose multi-core processor (right graph), a large number of tasks can be executed. However, tasks increasingly suffer from interference with decreasing criticality and priority. In our example, the predictability requirements are violated for a subset of safety-critical tasks. Resorting to predictable *Deterministic Execution Units* (DEUs) (middle graph) as proposed in our previous work solves this issue; however, it is accompanied by an inadequate utilization of the chip area. Moreover, tasks typically exhibit substantial dependencies and precedence constraints, for example between sensing, computing and actuating. Consequently, synthesis of dedicated soft cores for individual tasks quickly reaches its limits and is not generally applicable.

As mentioned earlier, real-time systems are heterogeneous in practice, with performance and safety requirements varying between individual tasks. We assume that the predominant part of real-time systems is of low criticality (e.g., comfort and diagnostic functionality). To leverage FPGAs in such heterogeneous settings a combination of performance-oriented general-purpose cores and task-exclusive real-time cores is missing that allows for interdependencies and precedence constraints between critical and non-critical tasks and, at the same time, prevents harmful interference by strong isolation concepts. Such an approach, as illustrated in Fig. 2 (right graph) would solve both the performance and the predictability issue, while applying to a broad range of application scenarios.

1.2 Our Contribution

In this paper, we present an extended work of the *reconfigurable, reliable, deterministic, distributed* (R2-D2) toolchain. Critical and uncritical tasks are now automatically mapped to an application-specific SoC including a combination of hardwired COTS multi-cores and deterministic soft-core processors. We claim the following key contributions to improve:

- Improved *scalability* by an automated mapping of soft and hard real-time tasks to COTS processors and *Deterministic Execution Units* (DEUs), respectively.
- Enhanced *performance* due to an optimized utilization of the available FPGA resources.
- Strong *isolation* between deterministic and non-deterministic execution units by tailored data exchange and synchronization mechanisms.

The remainder of this paper is structured as follows: First, we detail our R2-D2 approach, the automated architecture generation and shared resources in Sec. 3, followed by an scalability analysis in Sec. 4. Subsequently, we exemplify our approach by a flight control showcase in Sec. 5 and 6 before discussing related work in Sec. 7 and concluding the paper in Sec. 8.

2 System Model

Our approach demands three fundamental properties from the real-time system: (1) All allocation and scheduling-relevant system objects (threads, ISRs, resources, etc.) and their configuration are known ahead of time; either provided by some configuration file or statically extractable from the source code. (2) Precedence constraints and dependencies are explicitly implemented or modeled as dedicated tasks and resources. (3) A deterministic scheduling policy, such as fixed-priority preemptive scheduling.

Without loss of generality, we based our approach on the system model mandated by the AUTOSAR/OSEK-OS standard [15]. The AUTOSAR standard defines a widely used class of fixed-priority *real-time operating systems* (RTOSs) and has been the dominant industry standard for automotive

applications for the last two decades. For a specific application, the developer declares all system objects and their parameters in a domain-specific configuration file.

Furthermore, we make the following assumptions about the applications and the resulting task set: (1) The latter consists of critical and non-critical tasks with high and low demands on predictability and timing-analysis accuracy, respectively. (2) For a given FPGA, the combined computational load cannot be met by deterministic hardware cores. (3) The predominant pattern of synchronization between critical tasks is either a producer/consumer relationship or mutual exclusion. (4) Critical tasks do not depend on non-critical tasks.

3 The R2-D2 Approach

This section introduces our R2-D2 approach: starting with basic design principles, we subsequently detail our approach to automated architecture generation. Finally, we tackle the challenging issue of unavoidable intra-task dependencies and shared resources.

3.1 The Basic Approach

COTS multi-core architectures are designed and optimized for general-purpose applications. To execute arbitrary code, processor features including shared resources, as common memories or bus systems are necessary. If multiple tasks attempt to access them at the same time, it results in interference between these tasks. To avoid this issue from the first, we propose an application-specific processor architecture with dedicated, spatially separated hardware resources for safety-critical tasks. The detailed structure of an exemplary architecture generated by the R2-D2 toolchain is illustrated in Fig. 3. Every safety-critical task is exclusively executed by one DEU that includes all required components to run as an independent basic microcontroller system. The corresponding CPU core is customizable, different architectures from 8-bit to 64-bit can be selected. Additionally, some designs are parameterizable, thus, for example floating point units or hardware dividers can be optionally instantiated. The core is connected as sole master on one AXI bus to the corresponding slave peripherals. By default, a timer for periodical execution of code, a watchdog as a fail-safe option, a controller for *inter-processor communication* (IPC) and memory are instantiated. The abolition of a common memory for all processor cores furthermore eliminates a well-known bottleneck of COTS multi-core systems. Moreover, the distributed, spatially separated memories of safety-critical tasks increase the reliability of the system. Corrupt write accesses from uncritical tasks are inhibited by design. Thus, a *memory protection or management unit* (MPU/MMU) can be omitted. Since only one task is executed on one DEU, no additional scheduling mechanisms are required. As a consequence, an operating system is not necessary any longer. Tasks can run on bare-metal. This reduces the complexity of the system drastically because no preemption of tasks with higher priority can occur. Only the source code itself and data dependencies between tasks have to be regarded for timing analysis of the safety-critical tasks. Therefore, tasks running on dedicated DEUs have the same behavior as they would be executed on bare-metal single-core systems. For such single-core systems there exist a range of scientific [14] and commercial [1] static timing analysis methods, which can now be applied to this application-specific multi-core system. Hence, the verification of the system can be simplified significantly.

However, a dedicated DEU instance for all tasks is inconvertible for multiple reasons. Large systems with a considerable amount of critical and uncritical tasks may not be feasible due to limited FPGA resources. Some tasks require large memory space for example to log data values or to display and plot system states. For such tasks, which are predominantly at a low criticality level, access to external storage is essential. Another point is the missing performance of soft-core processors to execute compute-intensive tasks, which also require OS services, for example, webserver applications. These tasks may be integrated into large systems, but do not belong to the safety-relevant part.

These disadvantages of the R2-D2 approach can be wiped out by extending the architecture by performance aware COTS processor cores and corresponding memory controllers. To preserve the application-specific, deterministic behavior of the tasks executed on the DEUs, the COTS processor cores are only loosely coupled with the deterministic part of the system. Data exchange between the COTS and the deterministic cores is realized by dedicated hardware components, so the DEUs stay isolated and can be analyzed separately. Since for uncritical tasks deterministic behavior is not necessary, all non-safety-critical tasks can be executed on a COTS multi-core processor architecture. Empiric studies show [9] that only a few tasks within an application system are safety-relevant, so just a low number of dedicated DEUs will be required. As a consequence, with the extension of the R2-D2 toolchain by COTS cores more complex systems can be implemented and

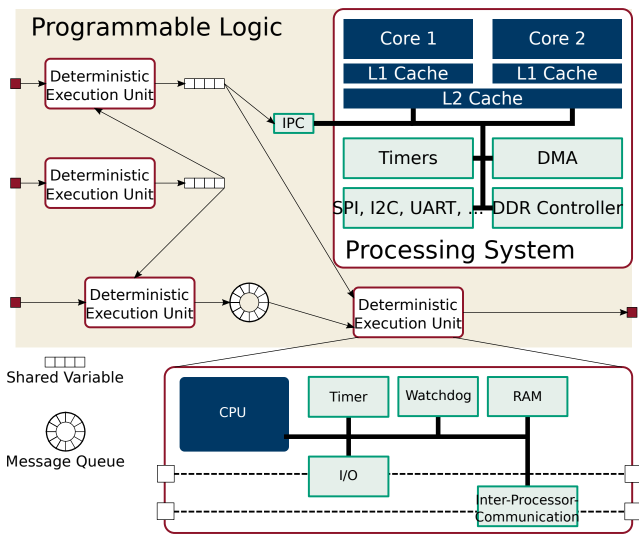


Fig. 3 The structure of an application-specific multi-core architecture is implemented on a SoC with an FPGA part and a firm multi-core processor. Safety-critical tasks were allocated on dedicated *Deterministic Execution Units*. The inter-processor communication is realized by hardware registers.

their real-time behavior can be guaranteed by timing analysis tools.

3.2 Automated Architecture Generation

Safety-critical applications, which have to be certified by authorities, must be developed according to defined standards, as *AUTOSAR* for automotive applications and *ARINC 653* in the domain of avionics. All these standards have in common that all hardware properties and the executed software have to be known and defined beforehand. Typical information, which is stored in such system descriptions are number, priorities, dependencies, shared variables, stimulus, and allocations of tasks. Execution of unknown software is forbidden by the applied standards. This fact is exploited by the R2-D2 approach. The system description is the basis for a multi-core architecture with a highly optimized, application-specific inter-core-communication system to avoid non-deterministic shared hardware resources.

Usually, system specification descriptions have a similar structure, in which all properties and dependencies are stored in an XML file. To receive a better compatibility, the open source system description format of *APP4MC* [2] was taken as input format to generate the processor architecture. For an automated application-specific hardware generation, the parameters of the application system have to be

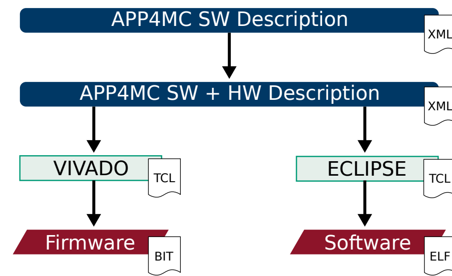


Fig. 4 The tool-flow for the automated hardware generation. An *APP4MC* system description is required as input and will be extended by additional parameters defining the application-specific hardware. Then TCL instructions for *Xilinx Vivado* and *Eclipse* are created to generate a hardware architecture instance and raw task frames for the application.

extracted from the *APP4MC* description. An overview of the different stages of the R2-D2 toolchain to generate an application-specific hardware architecture and the corresponding software system is illustrated in Fig. 4. The implementation details are described below.

3.2.1 APP4MC XML Description

In the first step, parameters of the software system stored in the *APP4MC* XML description are extracted. From there, additional parameters configuring the hardware architecture and the allocation of tasks were derived and added to the XML system specification. For each entry of a safety-critical task, which is classified as safety-critical if it has assigned a higher priority than a watermark level, an entry of a complete DEU gets instantiated. It includes information about a parametrizable processor core, a dedicated memory with the required size, the necessary peripherals, and the isolated bus system. Furthermore, the safety-critical task gets allocated to its application-specific DEU. Uncritical tasks get allocated to the COTS multi-core processor and will be scheduled by an RTOS. The system description stays compliant with the *APP4MC* standard and can be exchanged if further tools are necessary. Existing projects developed with an *APP4MC* compliant XML structure already include information about hardware and task allocation. In this case, only the information about the tasks is used, and the other entries in the XML file will be replaced.

Moreover, processor type and core properties, as well as required peripherals of every DEU can be configured to get sufficient performance and saving valuable FPGA resources. Thus, additional DEUs can be implemented, or smaller FPGA devices can be selected. All available processor cores with their performance values and required FPGA logic are listed in Tab. 1. As default configuration of a DEU

Table 1 Required space of various processor architectures on the *Xilinx Zynq-7020* FPGA device. An additional limitation is the internal memory of 560 KB, which has to be segmented for all DEUs.

Architecture	Bits	LUTs	Registers	DSPs	max. Frequency (MHz)
Microblaze min.	32	1893 (3.6%)	1660 (1.6%)	0 (0%)	148.0
Microblaze max.	32	3543 (6.7%)	3018 (2.8%)	6 (2.7%)	138.9
Picoblaze	8	189 (0.4%)	145 (0.1%)	0 (0%)	123.1
NEO430	16	833 (1.6%)	567 (0.5%)	0 (0%)	62.2
Custom *MIPS I*	32	2918 (5.5%)	2373 (2.2%)	0 (0%)	91.2
Rocket RISC-V	64	29640 (53.8%)	13484 (12.7%)	24 (10.9%)	28.6
PicoRV32 RISC-V	32	1467 (2.8%)	825 (0.8%)	4 (1.8%)	159.8

core, a *Microblaze* 32-bit processor including a JTAG debugging interface in minimum configuration without integer divider, barrel shifter, caches and floating point unit is selected. The processor configuration can be modified later in the *Xilinx Vivado Design Suite* to implement those additional hardware features, or other processor architectures can be chosen and parametrized. If only a minimum of computing power is needed, the *Picoblaze* can be selected. It is an 8-bit architecture, which requires less than 200 *Lookup Tables* (LUTs) and less than 200 registers on a *Xilinx Zynq-7020* FPGA device. With the *NEO430*, an implementation of the 16-bit *MSP430* architecture is provided. Furthermore, a royalty-free custom 32-bit *MIPS I* implementation can be selected. Other cores that are available as open source are two RISC-V cores: the *Rocket* core is a 64-bit architecture with additional non-deterministic hardware features. This processor is not designed to execute safety-critical tasks but might be an option if another powerful COTS core is needed. The *PicoRV32* is one of many uprising open source lightweight RISC-V 32-bit cores. It is optimized for FPGA devices, and its *instruction set architecture* (ISA) is configurable. Besides the basic instructions, the compressed instruction set and the multiplication/division subset can be implemented. The *PicoRV32* does not provide a debug interface, but tracing options are available. As COTS processors all *ARM* cores on the *Xilinx Zynq* and *Zynq UltraScale+* devices can be selected since the toolchain requires the *Vivado Design Suite* from *Xilinx*. Uncritical tasks can, therefore, be scheduled on an *ARM Cortex-A9* dual-core, an *ARM Cortex-R5* or *ARM Cortex-A53* quad-core.

3.2.2 Hardware Generation using Vivado Design Suite

After the *APP4MC* system description is updated with hardware-specific parameters, a set of TCL commands for *Xilinx Vivado* can be derived from the XML system description. Thereof, an instance of this application-specific architecture gets implemented in *Vivado*.

At first, all DEUs defined in the *APP4MC* XML were generated successively. For each one, the corresponding processor core and peripherals were instantiated. Subsequently, those components were connected by a dedicated AXI bus system with the processor core as a single bus master, as shown in Fig. 3. At this stage, the DEUs are completely isolated, which results in a deterministic structure so that every DEU can be regarded and analyzed independently. However, data exchange between tasks running on different DEUs is mandatory, so in the following step, the communication links were implemented.

The R2-D2 toolchain supports two communication types: *shared variables* providing common access to specific values and *message queues*, which include a FIFO storage for buffered data transfers. Moreover, data transfers between non-critical tasks executed on a general-purpose multi-core and data transfers with safety-critical tasks have to be distinguished. Uncritical tasks running on the COTS multi-core processor can simply share data by the common memory. Message queues are realized in software by an RTOS. For safety-critical tasks running on DEUs, a common memory between all cores in the system is not an option. This would result in a bottleneck when accessing shared data and reduce the performance. Furthermore, it would destroy the gained isolation of DEUs by the spatially separated execution of tasks. As all data transfers between tasks are already defined in the system description file, dedicated communication links between DEUs can be implemented in an application-specific hardware. Shared variables are realized as dedicated hardware registers. To transfer data, one task writes values in the dedicated register. The DEUs of all tasks requiring read access to this variable are physically connected to the corresponding register. This principle is realized in the IPC peripheral of the system, as illustrated in Fig. 3. Message queues are realized by hardware FIFO blocks, which can either be mapped to RAM blocks or registers in the FPGA, depending on the buffer size of the message queues. Moreover, message queues own

two AXI slave ports, which are directly connected to the bus system of the corresponding DEUs. Thus, exclusive access for the involved tasks is provided.

After the automated implementation of the architecture instance in *Vivado*, further post configurations can be done, as pinout and timing constraints. In a final step, the hardware bitstream is generated.

3.2.3 Generation of Raw Task Frames using Eclipse

As the decentralized, irregular structured multi-core architecture instance owns isolated DEUs with different processor cores, different memory maps and without common memory, each safety-critical task has to be treated as a separated embedded software project. Every task executed on a corresponding DEU requires a dedicated linker script and has to be compiled with a customized compiler toolchain. This causes a poor usability. Systems with several connected DEUs cannot be handled anymore, as the software structure gets too complex. Moreover, the programmability and the debugging options are inconvenient to use. To avoid these issues an additional software infrastructure layer gets created for every individual project, so from the programmers point of view it has the same behaviour as a single bare-metal c-code project. Therefore, a TCL script for the *Eclipse* IDE from *Xilinx* is derived from the *APP4MC* XML. Thereby one software project is generated for every DEU. Each one includes a basic bare-metal application and the software drivers for all connected peripherals. For the general-purpose multi-core processor, a *FreeRTOS* project with raw task frames is generated. To keep all source files in a common folder structure, the code sections for the different tasks running on various DEUs have to be delimited by *define* symbols. Thus, the different compiler toolchains from various projects generate binaries for all DEUs from a common source-code base. Furthermore, all shared variables have globally the same addresses, so the usage of the shared variables behaves the same as on a single software project.

3.3 Shared Hardware Resources

The basic principle of the presented approach is to avoid shared hardware resources by an application-specific multi-core architecture to prevent interferences between tasks. Shared hardware resources, for example, common SPI controllers of multiple cores, can be resolved by instantiating DEUs explicitly for managing shared peripherals. Thus, no locking mechanisms are necessary. Tasks requiring access to this common resource

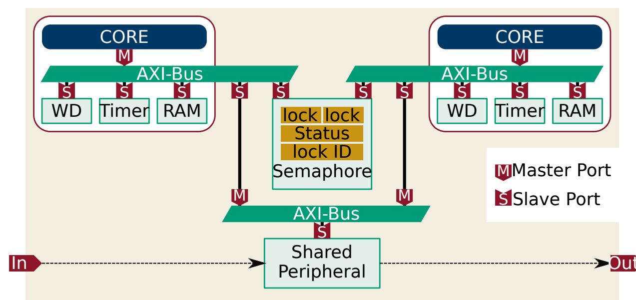


Fig. 5 To support shared peripherals and to avoid a common bus system between DEUs, shared hardware resources are connected by a bus system with a lower hierarchy. Thus, the DEUs stay spatially separated. For correct execution of applications, locking mechanisms are required. Therefore, a semaphore unit is implemented in hardware.

can transfer data via message queues to the DEU maintaining the peripheral. However, if legacy designs shall be ported or there is not enough space on the FPGA device for additional DEUs, it is also possible to connect a shared hardware peripheral to multiple DEUs. To keep the spatial separation of the DEUs, an additional bus in a lower hierarchy level is implemented, as shown in Fig. 5. This lower hierarchy bus has only one slave port, which is connected to the shared peripheral, and several master ports. Those master ports, which are illustrated in Fig. 5, are connected to the slave interfaces of the DEUs' buses. In the lower hierarchy bus, a deterministic round-robin scheduling algorithm arbiter is realized. Therefore it is possible to determine the exact number of clock cycles for a data access on the shared peripheral. All other bus accesses of cores to their dedicated peripherals within a DEU cannot be influenced by other cores. The bus system of the DEUs itself stay isolated. However, most operations on shared peripherals cannot be handled with a single data access. To take up the example of the shared SPI controller again, several bytes need to be transferred to send a complete package of a communication protocol. During this time an exclusive access over a certain time on the shared peripheral is mandatory. To realize this exclusive access on a shared peripheral device, a further locking mechanism is necessary, which must be accessible by all DEUs, which require access to the shared peripheral. Therefore another dedicated hardware block was designed, which implements the functionality of a semaphore. This IP-core gets instantiated for every shared resource. It provides one AXI slave port for all DEUs, which need to access the shared peripheral. Thereby the spatial separation of the DEUs gets not affected. All connected cores can read the common state of the hardware semaphore but have only write access to a dedicated

register for lock requests. If a task needs to lock a semaphore, it can access the semaphore hardware unit and set a flag in its dedicated control register within the IP block. Each of those control registers owns a unique, static priority. An internal state machine checks all control flags, for incoming locking requests. Simultaneous requests are handled considering the static priorities so that a correct real-time behavior can be guaranteed. To check if the task obtains access to a shared peripheral, the status register of the hardware semaphore can simply be polled. In contrast to COTS cores executing multiple tasks by an RTOS, no scheduling is required on the DEUs, which simplifies the timing analysis of the tasks obvious. For increased reliability, the semaphore can only be unlocked by the core, that locked the semaphore.

4 Scalability

4.1 Resource Estimation

To show the scalability of the application-specific multi-core architecture, in the following an estimation of the FPGA resource utilization, as well as an comparison to the results of the corresponding implementations are given. Since the limiting factor in all tests was the number of available LUTs within an FPGA, flip-flops and RAM blocks were not considered for simplification.

Each generated architecture instance can be described in a similar way to a Petri net model as a triple I (1) with a task set T, shared data values S and the connection links L between tasks.

$$I := (T, S, L) \tag{1}$$

Each task $T_i \in T$ with a higher priority P_i than the critical priority level P_c is part of the set of safety-critical tasks $T_{SC} \subseteq T$ (2). For each safety-critical task one DEU has to be generated as a hardware unit.

$$T_{SC} := \{T_i | P_i \geq P_c\} \tag{2}$$

The resource usage of each DEU configuration R_{DEUi} including a core, peripherals and the local bus system can be determined by implementation reports of the applied components. For the following estimation, all DEUs consist of a *Microblaze* core with a dedicated *AXI* bus system and further periperhals, which are listed in Table 2.

Moreover, safety-critical shared data values $S_{SC} \in S$ must be considered, as they have to be implemented in hardware. Those can be identified by detecting the links L (3), which connect shared data values with safety-critical tasks (4).

Table 2 This table lists the required FPGA LUTs for one DEU in the selected configuration, which was used for the following utilization tests in this section.

Component	**LUTs**	**FFs**	**BRAMs**
MicroBlaze Core	1087	965	0
AXI Bus	283	61	0
Memory	224	213	1
Timer	283	215	0
UART	99	89	0
Watchdog Timer	74	94	0
Total	2050	1637	1

$$L \subseteq (T \times S) \cup (S \times T) \tag{3}$$

$$S_{SC} := \{S_i | ((S_i, T_j) \in L \vee (T_j, S_i) \in L) \wedge \exists T_j \in T_{SC}\} \quad \forall i, j \in \mathbb{N} \tag{4}$$

A shared data value S_i can be of type FIFO, semaphore, shared variable or shared peripheral $S_i := \{S_{FIFO}, S_{Semaphore}, S_{Variable}, S_{Peripheral}\}$. Those types have to be distinguished for resource estimation function f:

$$f : S_i \mapsto f(S_i) := R_i, \quad \forall S_i \in S_{SC} \tag{5}$$

S_{FIFO}: Buffered data between tasks can only be realized by a 1:1 connection. The size of the buffer register has influence on the number of flip-flops, but no relevant impact on the utilized LUTs, as the conrol logic stays the same. Therefore, every FIFO block requires only a constant size of LUTs R_{C_FIFO}.

$$\forall S_i = S_{FIFO} : \quad f(S_i) = R_{C_FIFO} \tag{6}$$

$S_{Semaphore}$: Semaphores implemented as hardware blocks are variable in number of connected DEUs. Beside a constant overhead $R_{C_Semaphore}$ for the control logic, the number of connected DEUs has to be considered for calculating the utilized LUTs.

$$\forall S_i = S_{Semaphore} : \quad f(S_i) = R_{C_Semaphore} + |T_{SC} \times S_i| * R_{C_Sem_Port} \tag{7}$$

$S_{Variable}$: A shared variable has a fixed size and no further constant overhead. The resulting dedicated hardware registers can be linked as a 1:n connection.

Table 3 This are the selected parameter values for the estimation test.

Parameter	Value / LUTs
P_C	0
N_{Regs}	16
R_{C_FIFO}	375
$R_{C_Semaphore}$	163
$R_{C_Sem_Port}$	36
$R_{C_VarWrite}$	7
$R_{C_VarRead}$	6

One DEU has a write access and an arbitrary number of DEUs are connected as read only. The LUTs required for write accesses $R_{C_VarWrite}$ and the overhead for read accesses $R_{C_VarRead}$ have to be distinguished.

$$\begin{aligned} &\forall S_i = S_{Variable}: \\ &\quad f(S_i) = R_{C_VarWrite} + |S_i \times D| * R_{C_VarRead} \end{aligned} \tag{8}$$

$S_{Peripheral}$: The resource usage of a shared peripheral strongly depends on the size of the peripheral itself ($R_{C_Peripheral}$). Additionally an AXI bus system has to be implemented to control the data access. For its size, the number of connected DEUs has to be taken into account.

$$\begin{aligned} &\forall S_i = S_{Peripheral}: \\ &\quad f(S_i) = R_{C_Peripheral} + |T_{SC} \times S_i| * R_{C_Per_Port} \end{aligned} \tag{9}$$

The resource usage $Total_LUTs$ of the whole design can now be estimated by adding the resources of each DEU R_{DEU_i} and all safety-critical shared values S_{SC}, which were determined by (6)-(9).

$$Total_LUTs = \sum_{i=1}^{|T_{SC}|} R_{DEU_i} + \sum_{i=1}^{|S_{SC}|} R_i \tag{10}$$

4.2 Estimated and Measured Resource Usage

To review the resource estimation functions, 4 scenarios with different amount of cores, FIFOs, semaphores and shared variables have been selected. The DEU configuration is the one described in Table 2, further required paramters were described in Table 3. The graph of Fig. 6 shows the estimations compared to the implementation results of four scenarios. Even if there is an FPGA utilization of 86% the deviation between the estimation and the LUTs utilized in a implemented design is less than 1%.

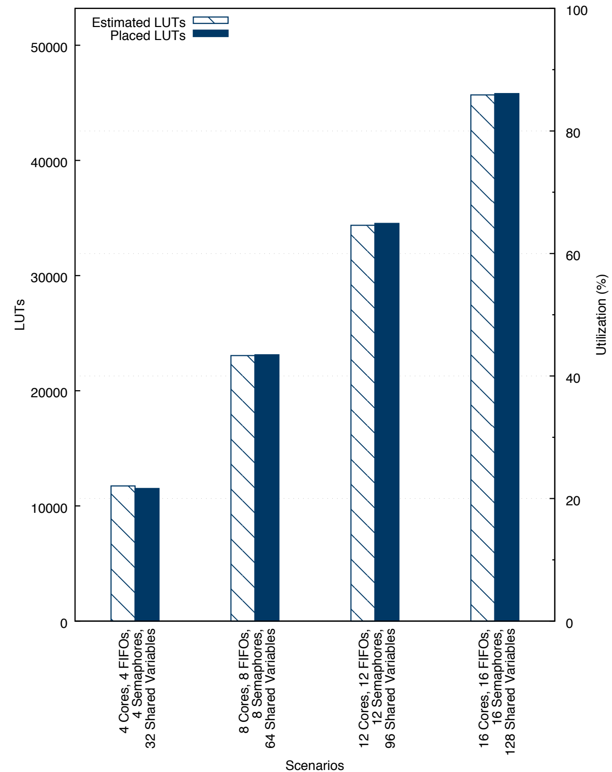


Fig. 6 In this graph the accuration of the estimation function for 4 scenarios is illustrated. The scenarios include 4 to 16 cores, FIFOs semaphores and 32 to 128 shared variables.

4.3 Limitations of Estimation

Beside the resource utilization of placed LUTs, FIFOs and RAM blocks also the routing nets are a delimiting factor. Depending on the kind of shared data values, a different amount of routing resources is required. Fig. 7 illustrates, that every design filled up with FIFOs can utilize the FPGA capacities to more than 90%, where in a design with 1000 shared variables only 75% of LUTS and 51% of flip-flops can be applied.

5 Quadcopter Controller Showcase

To show the advantages of the R2-D2 toolchain practically on an use case, an architecture of a quadcopter flight controller was generated and implemented on a *Xilinx Zynq 7020* device. The structure is shown in Fig. 8. The flight controller combines multiple functionalities: data acquisition from sensors, position estimation in space and motor control. For more complex flight operations the *Mission Control* task can navigate the quadcopter by GPS coordinates. The *MAVLink* task provides an interface over the identically named widespread UAV

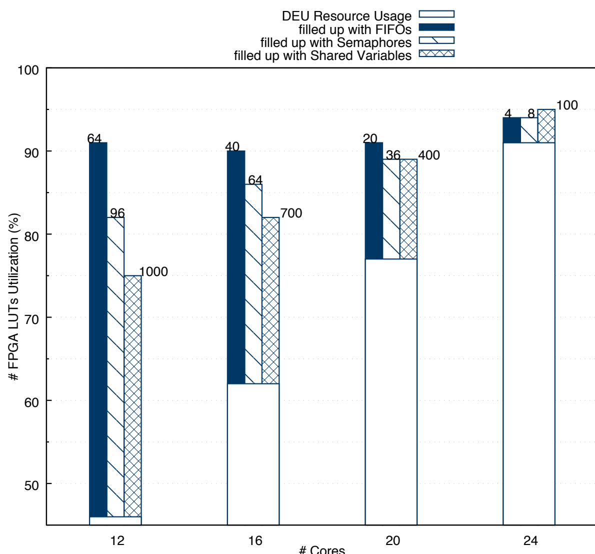


Fig. 7 This plot shows the maximum FPGA utilization when implementing a maximum number of FIFOs, semaphores and shared variables in a design with 12 to 24 cores. Due to routing limitations not all FPGA resources can be used.

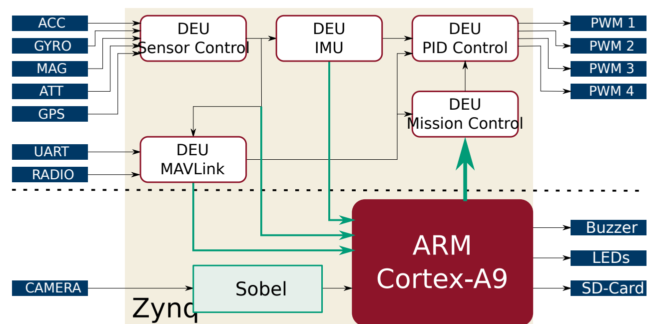


Fig. 8 A generated application-specific architecture for a flight-controller of a quadcopter, which is implemented on a *Xilinx Zynq 7020* device including five DEUs for safety-critical tasks and a firm *ARM Cortex-A9* dual-core processor for uncritical tasks.

communication protocol to configure settings, as PID parameters. Additionally, an image processing unit is implemented that scans the ground for landing marks and automatically activates a landing sequence.

The tasks of the system are partitioned into time critical and non-critical sections, as it is proposed in [18]. Main critical section of the flight controller is the control loop consisting of sensor-, position- and motor control. Therefore these three tasks are executed on isolated DEUs. Data between the *Sensor Control*, *IMU* and *PID Control* tasks are exchanged by shared variables, so all calculations are executed with the latest sensor data available, which is common practice to avoid synchronization locks. Since sensors and tasks deliver continuous values, deviations caused by delayed data updates are negligible. The tasks *Mission Control* and *MAVLink* have no hard deadline requirements, but they are classified as safety-critical, as they directly influence the control loop. Therefore, those tasks are also executed on DEUs. The non-critical part of the system includes the detection of the landing mark since it is an optional feature and the landing sequence can also be started manually, so the corresponding tasks were executed on the *ARM Cortex-A9* dual-core. Data transmissions between the DEUs and the COTS multi-core has to be reviewed in particular, as data dependencies or corrupt data transmissions jeopardize the reliability of the DEUs. Data transmitted from safety-critical tasks on DEUs to uncritical tasks running on the COTS multi-core are robust by design, since those uncritical tasks only have read access to the shared registers and therefore cannot manipulate their safety-critical content. In some cases also data transfers from uncritical tasks to the DEUs are necessary. In this application, the coordinates of a detected landing mark have to be transmitted. As corrupt data could engender damage on the quadcopter, additional mechanisms to increase the reliability of the system have to be added. Here this is realized by a CRC checksum, which is generated over transmitted coordinates. Furthermore, a plausibility check is implemented. Detected landing marks have to be within a radius of 10 meters. To avoid correctly and incorrectly detected landing marks, information from the ARM cores can also be ignored by the *Mission Control DEU*, if a flag is set by the *MAVLink DEU*. Moreover, this image processing application is comparably compute-intensive, so it benefits from the performance of the firm implemented dual-core. Due to the combination of multi-core and FPGA on one chip, the landing application can be additionally improved by a hardware accelerator for the *Sobel* algorithm, which processes the edge detection of an image. The feature detection is described in [16]. Detected landing sections are signaled to the mission control task, which will start the landing sequence.

The deterministic structure of the generated architecture with its isolated DEUs simplifies the later verification of safety-critical sections with static timing analysis tools. The control loop can be considered separately without influences of an RTOS or non-critical tasks, which were spatially separated. Uncritical parts can further be developed independently since the DEUs executing the safety-critical tasks do not share their code base.

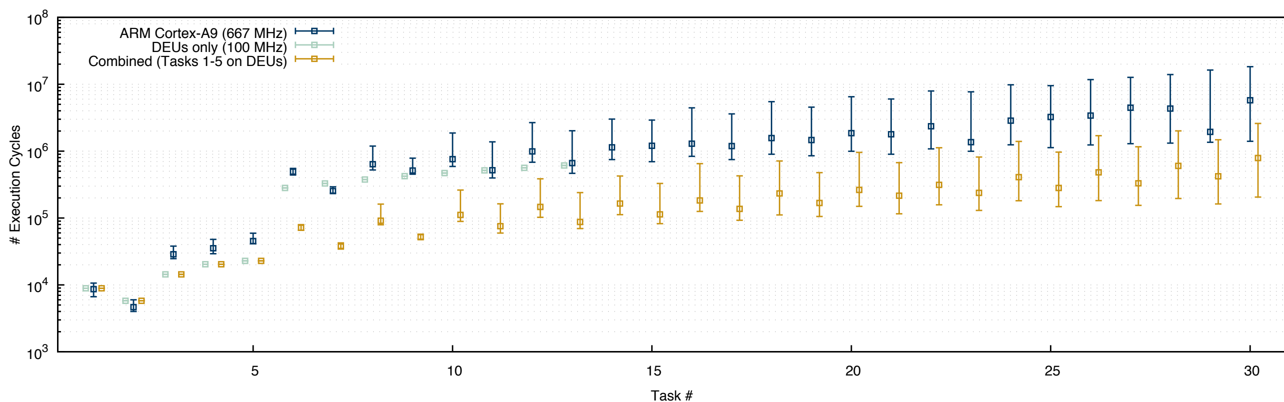


Fig. 9 In this graph the execution cycles of the 5 safety-critical tasks from the flight controller application and 25 further non-safety critical tasks are shown. The DEUs have a highly deterministic behavior, but the FPGA device does not provide enough space for 30 of them. By applying only the COTS dual-core, all task could be accommodated on the price of determinism. A combination of both architectures unifies the advantages.

6 Results

As hardware models to analyze the WCET of the lightweight DEU cores do not exist yet, we evaluted the system by measurements. The tasks of the application in Section 5 were implemented with a minimum amount of branches and all loops unrolled to emulate a constant, deterministic load for the cores. Thus interferences caused by hardware design can be revealed. In Fig. 9 the minimum, maximum and average number of execution cycles for the 5 critical tasks and 25 further non-critical tasks are shown. Task 1 to 5 have to fulfill a deadline of 1 ms. The other uncritical tasks were executed with a periodical repetition of 1 $ms * Task\ \#$. To compare the results, two reference systems have been implemented. In the first one, all tasks were implemented on the *ARM Cortex-A9* dual-core. Tasks with odd ID are allocated on core 1, those with even ID on core 2. On the second reference system, all tasks were executed on DEUs. The applied DEUs include *Microblazes* in extended configuration with, floating-point unit, integer divider, barrel shifter, stack protection and debug unit. The required FPGA resources for the core can be found in Table 1, the resources for the applied peripherals in Table 2. During the measurements, all deadlines were met.

The measured results show that the number of required execution cycles varies on the COTS multi-core even for safety-critical tasks. This is caused by task- interference and scheduling overhead of the RTOS. In contrast to that, the execution cycles of tasks running on the DEUs stay constant. The worst-case observed values of safety-critical tasks on the COTS are already higher than the corresponding execution cycles on the DEUs. A timing analysis would deliver even more pessimistic values for the multi-cores. The ARM cores benefit from a more extensive ISA. Especially the double precision FPU compared to the single precision FPU on the DEUs reduce the number of instructions. Therefore the average values of task number 1 and 2 on the COTS multi-core are lower than on the DEUs. The scheduling overhead and therefore the increased number of memory accesses lead in the worst-case to a higher amount of required clock cycles than the same tasks executed on DEUs. For the safety-critical tasks with the ID 3 to 5 the task-interference has further increased. More scheduling events and so an increased number of memory accesses is required. This leads to better results on the DEUs for those tasks, but due to limited FPGA resources the complete system is not realizable solely with DEUs. With this architecture only 13 tasks could be realized. The results of the combined architecture of ARM cores and DEUs shows, that the usage of DEUs for critical tasks unloads the ARM cores and also the scheduling overhead. Moreover, it reduces interferences, which is presented in Fig. 10.

Furthermore, this approach benefits from the fact that real-time critical tasks require rather low latencies than computing power and because of the high repetition rates they only need a few of KB memory for instructions. This favors the use of small soft-core processors, which have only a low performance in comparison to firm implemented cores. However the overhead of an RTOS for scheduling is not required, so low latencies can be realized. The instruction code is small enough to fit into the internal memory of the chip, so the principle of dedicated memory for all DEUs can be realized.

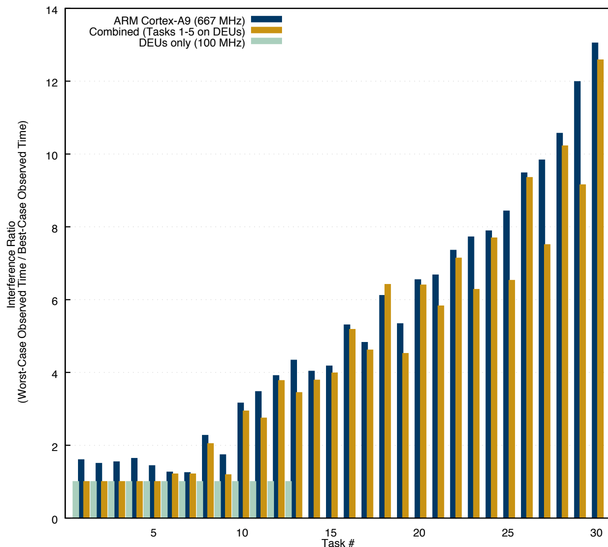


Fig. 10 This plot represents the ratio of the worst-case to the best-case observed execution times, as well as the FPGA utilization for the 3 implemented architectures.

7 Related Work

To certificate safety-critical systems several widespread domains have to be considered. Functional safety of the applications has to be proven by extensive testbenches taking every possible malfunction into account [3]. In terms of reliability, risk management is required. Redundant hardware platforms and software implementations guarantee the robustness of the system. Deterministic behaviour is one further relevant aspect for safety-critical systems, which affects various fields as hardware architectures, scheduling, compilers and WCET analysis tools [7,5].

Approaches that aim a deterministic behaviour on embedded multi-core systems either rely on COTS or on self-designed deterministic hardware architectures. The benefit of COTS hardware is the high performance, the availability of low-cost chips and the existing software infrastructure. There are various toolchains and operating systems, which can be applied. The challenge when using COTS multi-cores is to guarantee deterministic behaviour on partly non-deterministic hardware. An increasing trend to tackle this problem is the mixed-criticality WCET approach [13]. The key idea is that a mixed critical system consists of different modes of operation. In each mode a set of tasks and corresponding deadlines are defined. If a critical task seem to miss its deadline, the mode changes to a higher criticality level. Now low-critical tasks were not scheduled anymore and it can be guaranteed, that the critical tasks meet their deadlines. Therefore, this approach leads to a higher core utilization when executing uncritical and critical tasks on shared resources. However, the main problem remains: the analyzed WCET for critical tasks on multi-cores stay too pessimistic.

To avoid those pessimistic assumptions, other approaches focus on deterministic multi-core architectures, which are optimized for a low WCET. The *PATMOS* core [17] therefore provides separated caches for instructions, data and stack. Furthermore, scratchpads are available to control the contention for local memory. On the other hand this requires a complete redesign of existing applications and a tailored software for this processor to benefit from this design-specific improvements. *PRET* [12,23,11] is designed to reduce the effort of static timing analysis. Key concept is an architecture with repeatable timing, that is realized by thread interleaved pipelines. Thus, pipeline hazards are avoided by design. However, the performance of the core gets drastically reduced, if only one or two threads are active. Nonetheless, there is a need for such processor architectures, which are for example implemented in the *XMOS xCORE* family [4].

The cores of a deterministic multi-core design are either connected by a common deterministic bus [20, 8], or by a NoC to provide a scalable architecture. The required isolation between the cores is created by time division multiple access (TDMA). Thus, especially on NoCs the bandwidth for a single core decreases, if cores need random access to other ones. Round robin arbiters only achieve a low bus utilization, so several approaches try to improve the core communication bandwidth. In [10] for example a benes-based NoC structure is used. The *IDAMC* [19] focuses on reliable routing of critical and uncritical data on one NoC.

The R2-D2 approach avoids the disadvantages of WCET-specific hardware. Established single-core processors are used, so no additional instructions are required. Data dependencies between cores are realized by lightweight communication links, so no NoC structure with TDMA bottleneck is necessary. This results in a application-specific, scalable system at the expense of flexibility.

8 Conclusion

In this paper, we presented an approach to execute safety-critical and uncritical tasks in parallel on one chip by generating an application-specific multi-core architecture. The safety-critical tasks of a demonstration system were isolated and executed on dedicated, deterministic hardware units. Hence, timing

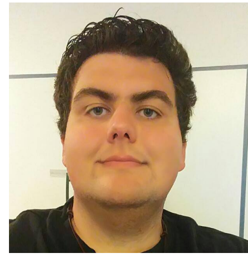
analysis for those tasks gets simplified significantly, as hardware interferences were avoided by design. Further measurements have shown that the selected FPGA device does not provide enough resources to implement all tasks on DEUs, so a combination of COTS multi-cores and DEUs is required. Thereby the jitter of the tasks was reduced, as well as the average execution time of the uncritical tasks executed on the COTS multi-core.

References

1. AbsInt aiT. URL https://www.absint.com/ait/
2. APP4MC. URL http://www.eclipse.org/app4mc/
3. Position Paper CAST-32A. URL https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-32A.pdf. Certification Authorities Software Team (CAST)
4. XMOS xCORE-200. Tech. rep. URL http://www.xmos.com/products/silicon/xcore-200
5. Castrillon, J., Sheng, W., Jessenberger, R., Thiele, L., Schorr, L., Juurlink, B., Alvarez-Mesa, M., Pohl, A., Reyes, V., Leupers, R.: Multi/many-core programming: Where are we standing? In: 2015 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1708–1717 (2015). DOI 10.7873/DATE.2015.1129
6. Ferdinand, C., Heckmann, R., Wolff, H.J., Renz, C., Parshin, O., Wilhelm, R.: Towards model-driven development of hard real-time systems. In: Model-Driven Development of Reliable Automotive Services, pp. 145–160. Springer (2008)
7. Fernandez, G., Abella, J., Quiñones, E., Rochange, C., Vardanega, T., Cazorla, F.J.: Contention in Multicore Hardware Shared Resources: Understanding of the State of the Art. In: 14th International Workshop on Worst-Case Execution Time Analysis, OpenAccess Series in Informatics (OASIcs), pp. 31–42. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. DOI http://dx.doi.org/10.4230/OASIcs.WCET.2014.31. URL http://drops.dagstuhl.de/opus/volltexte/2014/4602
8. Fernández, M., Gioiosa, R., Quiñones, E., Fossati, L., Zulianello, M., Cazorla, F.J.: Assessing the Suitability of the NGMP Multi-core Processor in the Space Domain. In: Proceedings of the Tenth ACM International Conference on Embedded Software, EMSOFT '12, pp. 175–184. ACM, New York, NY, USA (2012). DOI 10.1145/2380356.2380389. URL http://doi.acm.org/10.1145/2380356.2380389
9. Fürst, S.: BMW_20151005.pdf. URL http://image.mdstec.com/mail/SES/BMW_20151005.pdf
10. Kerrison, S., May, D., Eder, K.: A Benes Based NoC Switching Architecture for Mixed Criticality Embedded Systems. In: 2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC), pp. 125–132 (2016). DOI 10.1109/MCSoC.2016.50
11. Lee, E., Reineke, J., Zimmer, M.: Abstract PRET Machines. In: 2017 IEEE Real-Time Systems Symposium (RTSS), pp. 1–11 (2017). DOI 10.1109/RTSS.2017.00041
12. Liu, I., Reineke, J., Lee, E.A.: A PRET architecture supporting concurrent programs with composable timing properties. In: 2010 Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers, pp. 2111–2115 (2010). DOI 10.1109/ACSSC.2010.5757922
13. Mollison, M.S., Erickson, J.P., Anderson, J.H., Baruah, S.K., Scoredos, J.A.: Mixed-Criticality Real-Time Scheduling for Multicore Systems. In: 2010 10th IEEE International Conference on Computer and Information Technology, pp. 1864–1871 (2010). DOI 10.1109/CIT.2010.320
14. Montag, P., Altmeyer, S.: Precise WCET calculation in highly variant real-time systems. In: 2011 Design, Automation Test in Europe, pp. 1–6. DOI 10.1109/DATE.2011.5763149
15. OSEK/VDX Group: Operating system specification 2.2.3. Tech. rep., OSEK/VDX Group (2005). http://portal.osek-vdx.org/files/pdf/specs/os223.pdf, visited 2014-09-29
16. Pfundt, B., Reichenbach, M., Hartmann, C., Häublein, K., Fey, D.: Teaching heterogeneous computer architectures using smart camera systems. In: 2016 11th European Workshop on Microelectronics Education (EWME). DOI 10.1109/EWME.2016.7496484
17. Schoeberl, M., Pezzarossa, L., Sparsø, J.: A Multicore Processor for Time-Critical Applications. IEEE Design Test **35**(2), 38–47 (2018). DOI 10.1109/MDAT.2018.2791809
18. Schreiner, S., Grüttner, K., Rosinger, S., Rettberg, A.: Autonomous Flight Control Meets Custom Payload Processing: A Mixed-Critical Avionics Architecture Approach for Civilian UAVs. In: 2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, pp. 348–357. DOI 10.1109/ISORC.2014.28
19. Tobuschat, S., Axer, P., Ernst, R., Diemer, J.: IDAMC: A NoC for mixed criticality systems. In: 2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 149–156 (2013). DOI 10.1109/RTCSA.2013.6732214
20. Ungerer, T., Cazorla, F., Sainrat, P., Bernat, G., Petrov, Z., Rochange, C., Quinones, E., Gerdes, M., Paolieri, M., Wolf, J., Casse, H., Uhrig, S., Guliashvili, I., Houston, M., Kluge, F., Metzlaff, S., Mische, J.: Merasa: Multicore Execution of Hard Real-Time Applications Supporting Analyzability. IEEE Micro **30**(5), 66–75 (2010). DOI 10.1109/MM.2010.78
21. Vaas, S., Reichenbach, M., Margull, U., Fey, D.: The R2-D2 toolchain – Automated porting of safety-critical applications to FPGAs. In: 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig). DOI 10.1109/ReConFig.2016.7857192
22. Vaas, S., Ulbrich, P., Reichenbach, M., Fey, D.: The best of both: High-performance anc deterministic real-time executive by application-specific multi-core socs. In: 2017 Conference on Design and Architectures for Signal and Image Processing (DASIP) (2017). DOI 10.1109/DASIP.2017.8122107
23. Zimmer, M., Broman, D., Shaver, C., Lee, E.A.: FlexPRET: A processor platform for mixed-criticality systems. In: 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 101–110 (2014). DOI 10.1109/RTAS.2014.6925994

Steffen Vaas received his master degree in Information and Communication Technology at the Friedrich-Alexander-University (FAU) Erlangen-Nürnberg in 2015. Currently he is a research fellow at FAU at the chair of computer architecture. There he focuses on heterogeneous, application-specific and deterministic multi-core architectures.

Peter Ulbrich is research associate and lecturer at the Chair of Distributed Systems and Operating Systems at FAU Erlangen-Nürnberg, Germany. He received his diploma from FAU in 2007 and joined the group of Wolfgang Schröder-Preikschat at FAU, where he received his Ph.D. in 2014. He research focuses on real-time and dependable systems. Dr. Ulbrich is a member of GI, ACM, and IEEE.



Marc Reichenbach received his Diploma Degree in Computer Science at the Friedrich-Schiller University Jena in 2010 and his PhD in 2017 at Friedrich-Alexander-University Erlangen-Nürnberg (FAU). He works now as a postdoctoral researcher at FAU at the chair of Computer Architecture. His research interests are embedded systems, especially smart sensor architectures for different application fields.

Dietmar Fey holds a diploma degree in Computer Science from Friedrich-Alexander-University (FAU) Erlangen-Nürnberg, Germany. In 1992 he received a Ph.D. from FAU with a work on an investigation about Using Optics in Computer Architectures. From 1994 to 1999 he researched at Friedrich-Schiller-University Jena where he made his habilitation. From 1999 to 2001 he worked as lecturer at University Siegen before he became a Professor for Computer Engineering at University Jena. Since 2009 he leads the Chair for Computer Architecture at Friedrich-Alexander-University Erlangen-Nürnberg (FAU). His research interests are in parallel computer architectures, parallel programming environments, parallel embedded systems, and memristive computing.