HSB
Hochschule Bremen
City University of Applied Sciences

A Survey of Time and Space Partitioning for Space Avionics

Jan Bredereke

City University of Applied Sciences Bremen

Flughafenallee 10, 28199 Bremen, Germany

jan.bredereke@hs-bremen.de

Technical Report

Version: 1.0

Document Version History

Version	Date	Changes
1.0	27 June 2016	Initial version.

Abstract

We present a survey of the current state of the reseach on time and space partitioning for space avionics. The availability of ever more powerful computers allows to assign many control tasks to a single computer easily, in principle. But in its naïve form, this would mean too much effort and thus cost for demonstrating dependability. For aircraft, there is already an approach to solve this problem, the Integrated Modular Avionics (IMA) architecture. For spacecraft, the basic problem is similar. But in detail, the setting is different, though. This report compiles a survey and identifies relevant research challenges. Based on some of them, we propose to design a multi-core processor architecture that avoids fundamental problems of the current architectures with respect to time partitioning, and that can be used in the space domain.

Contents

Chapter 1

Introduction

We present a survey of the current state of the reseach on time and space partitioning for space avionics. Avionics is the set of electronic systems, in particular computers, on board of an aircraft or spacecraft. Such systems must be dependable. There are significantly different levels of dependability, for example for the cabin electronics of an aircraft and for its autopilot. The higher the level required, the higher the effort necessary for demonstrating dependability. The traditional approach therefore provisions separate computer hardware for each control task. The effort for demonstrating dependability then follows from the level targeted, for each task.

The availability of ever more powerful computers allows to assign many control tasks to a single computer easily, in principle. This would have the advantage of saving considerable weight and thus cost for the aircraft or spacecraft. But this idea, in its naïve form, would mean too much effort and thus cost for demonstrating dependability. A level demanded by one of the tasks must be applied to all of them.

For aircraft, there is an approach for this already. The Integrated Modular Avionics (IMA) architecture (see Chap. 4 below) is a software and hardware platform which isolates software programs from each other. The dependability needs to be demonstrated at the highest level only once, for the platform. This concept is used in practice already for the aircraft Airbus A380, Airbus A400M, and Boeing 787 Dreamliner.

For spacecraft, the basic problem is similar. We want to run several tasks on a single computer (or a few computers) for cost reasons, with dependability requirements that are high, too. But in detail, the setting is different, though. There is already some research on time and space partitioning of control tasks in space craft. But there is not yet a concept proven in practice.

Further domains have similar problems and according solution approaches. In the automotive domain, the AutoSAR initiative (automotive open system architecture) is related. We do not survey it here since it is not as practice proven yet as IMA in the avionics domain, and since our research resources are limited.

The aim of this work is to compile a survey and to identify relevant research challenges. We hope to then pursue some of these research questions further. Our work here was conducted during a (partial) sabbatical in winter term 2015/16 and during the following summer term 2016.

We thank the people at the Data Handling Avionics department of Airbus DS, Bremen, for pointing us at this interesting research subject.

Chapter 2

Systems with Mixed Dependability

In this chapter, we introduce to systems with mixed dependability. We briefly introduce to the notions of dependability and of mixed dependability first. We then describe two mechanisms to handle mixed dependability.

2.1 Dependability

The survey paper by Avižienis *et. al.* [Avi+04] provides definitions for dependability in the domains of computing and communication systems. The original definition of dependability is the ability to deliver service that can be justifiably trusted. This definition stresses the need for justification of trust. The alternate definition that provides the criterion for deciding if the service is dependable is: the dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable.

Dependability is an integrating concept that encompasses the following attributes [Avi+04]:

availability: readiness for correct service.

reliability: continuity of correct service.

safety: absence of catastrophic consequences on the user(s) and the environment.

integrity: absence of improper system alterations.

maintainability: ability to undergo modifications and repairs.

2.2 Mixed Dependability

A system with mixed dependability is a system where components with different dependability levels coexist on the same execution platform. Crespo *et. al.* [Cre+14] call this a "mixed criticality system" (MCS): Increasing processing power makes it possible to integrate more and more components on a single execution platform. However, if at least some of the components must be dependable, adequate validation (and often certification) is necessary, such that validation costs can become prohibitive for further integration. This trend can be observed in many different domains. Examples are the aeronautical domain, the space domain, the automotive domain, and industry automation.

2.3 Handling Mixed Dependability

A general approach to this is to separate the components on the single execution platform so well that only the separation mechanism and the high dependability components need to be validated for a high dependability. Mechanisms to achieve such a separation comprise a *separation kernel* and *virtualization*.

2.3.1 Separation Kernel

A separation kernel is a combination of hardware and software for allowing multiple functions to be performed on a common set of physical resources without interference [Cre+14]. It was first proposed by Rushby [Rus81], aiming at security problems.

According to Crespo *et. al.* [Cre+14], the MILS architecture, developed by the MILS (Multiple Independent Levels of Security and Safety) initiative [Alv+05], is a separation kernel. In addition, the ARINC-653 standard [Aer05] uses these principles to define a baseline operating environment for application software used within Integrated Modular Avionics (IMA), based on a partitioned architecture. Wie will describe IMA in more detail in Chap. 4.

2.3.2 Virtualization

Crespo *et. al.* [Cre+14] use virtualization as a separation mechanism. A hypervisor implements partitions or virtual machines that are isolated from each other in the temporal and spatial (i.e., storage) domains.

Different kinds of isolation must be considered [Cre+14, Sect. 2.1]:

Fault isolation: a fault in an application must not propagate to other applications.

Spatial isolation: applications must execute in independent physical memory address spaces.

Temporal isolation: the real-time behaviour of an application must be correct independent of the execution of other applications.

Crespo *et. al.* [Cre+14, Sect. 2.1] propose a predefined and static allocation of resources to partitions, in order to achieve a separation that is sufficiently simple to allow for separate validation. The resources comprise CPU time, memory areas, IO ports, etc. Static allocation of CPU time should be achieved by a cyclic scheduling policy for partition execution.

Crespo *et. al.* [Cre+14, Sect. 3] state that there is some confusion of terminology on virtualization, and they propose the following definitions: A *type 1 hypervisor* (also named native or bare-metal hypervisor) runs directly on the native hardware, while a *type 2 hypervisor* is executed on top of an operating system. *Full virtualization* provides a complete re-creation of the hardware behaviour of a native system to a guest system, while *para-virtualization* requires the guest system to be modified: Some machine instructions are replaced by functions provided by the hypervisor. In full virtualization, certain "conflicting" machine instructions must be caught during runtime, in order to maintain the spatial and temporal separation. They are then handled by the hypervisor. With para-virtualization, in contrast, no catching is necessary, and the handling can use more information from the guest. This improves the performance greatly, and it simplifies the hypervisor. Of course, the source code of the guest must be available for recompiling. Since the latter usually is not a problem for mixed dependability systems, para-virtualization is preferable here.

Crespo *et. al.* [Cre+14, Sect. 3.2] categorize the IMA architecture (see Chap. 4 below) as a separation kernel. However, part of it comes close to the functionality provided by a hypervisor.

Conceived by the same research group, XtratuM (see Sect. 6.3.1) is an open-source, type 1 hypervisor that uses para-virtualization. It was designed specifically for mixed-criticality systems.

Virtualization aims at presenting a virtual machine to an application software which is exactly like the real machine. However, it cannot hide one kind of difference: the machine instructions are not executed evenly in the time sense anymore. There are "holes" where other partitions get time. The application software can notice this when it interacts with the system's environment. This is relevant for real-time applications, for example for control applications where the controlled system does not stop while the application is on hold. Another example is the interaction with peripheral devices which change state by progress of time, such as timers.

The latency of an interrupt can become substantially higher, i.e., until the partition of the interrupt is scheduled again. This can break assumptions about the timing of interrupts made by an application or by an operating system. For example, Ripoll *et. al.* [Rip+10, Sect. 3.2] report that they used the RTEMS operating system in a partition, and that the timer tick of the RTEMS operating system was faster than the schedule period of the partition. Therefore, they had to take measures to ensure that accumulated clock ticks were presented to the partition at the beginning of its time slot.

2.3.3 Separation Kernel vs. Virtualization

Separation is a solution to the problem of mixed dependability, and virtualization is one possible mechanism to achive separation. Virtualization comprises more than necessary to achive separation. For example, the application in a (fully) virtualized machine cannot "see" any differences (besides "holes in time") to a dedicated real machine. This is not necessary for separation. Instead, we can provide some modification of a real machine to the application, as long as it guarantees separation. The approach of para-virtualization (compare Sect. 2.3.2 above) goes a step into exactly this direction.

A separation kernel imposes the use of the same operating system onto all applications, while virtualization allows for different operating systems (or even bare-metal applications without any operating system) in its partitions. The latter can be a substantial advantage if heterogeneous applications are to be integrated.

A separation kernel usually comprises more functionality (e.g., on communication, and maybe on multi-threading) than a hypervisor used for virtualization. Therefore, we suspect that the effort necessary for verifying the time and space separation property is usually higher for a separation kernel.

See also Sect. 7.3 on the research challenge on which of the two mechanisms has which relative pros and cons, under which conditions.

Chapter 3

Timing Analysis

This chapter provides some basic knowledge on timing analysis. We need this knowledge for further discussions of time partitioning and, later on, for an idea on a processor architecture supporting time partitioning.

3.1 Proof Techniques for Real-Time Properties

In this section, we introduce to proof techniques for real-time properties of embedded systems such as space avionics. This provides the background for the discussion of time partitioning further below. Embedded systems such as space avionics often must satisfy hard real-time properties. That is, they provably must provide the reaction to a stimulus before a given limit of time. Techniques exist to provide such proofs. The worst-case execution time (WCET) is the maximum time a piece of code will ever need for execution on a given hardware, regardless of input and internal state. The proof of a hard real-time property consists in establishing an upper bound for the WCET which does not exceed the given limit. Usually an upper bound is used, instead of the actual WCET, because of a lack of more precise information. This information has been abstracted away in order to reduce the effort of analysis (or to make it feasible in the first place).

In a partitioned system, there is an execution time budget for each partition. Accordingly, the proofs of the hard real-time properties of all partitions must be carried out, hopefully in a (mostly) independent way.

Kopetz [Kop11, p. 245] describes how to determine an upper bound for the execution time of machine code commands on a target hardware. If the processor of the target hardware has fixed instruction execution times, the duration of the hardware instructions can be found in the hardware documentation and can be retrieved by an elementary table look-up. The adding-up of the instruction execution times must be done for all combinations of input and initial state. Such a simple approach does not work if the target hardware is a modern RISC processor with pipelined execution units and instruction/data caches. While these architectural features result in significant performance improvements, they also introduce a high level of unpredictability. Dependencies among instructions can cause pipeline hazards, and cache misses will lead to a significant delay of the instruction execution. To make things worse, these two effects are not independent. A significant amount of research deals with the execution time analysis on machines with pipelines and caches. The excellent survey article by Wilhelm *et. al.* [Wil+08] presents the state of the art of execution time bounds analysis in research and industry in depth, and it describes many of the tools available for the support of execution time bounds analysis.

A book chapter by Wilhelm [Wil09] elaborates even further on how to determine bounds on execution times.

Wilhelm *et. al.* [Wil+08, Chap. 2.1.3] point out the problem of timing anomalies. A timing anomaly is a counterintuitive influence of the (local) execution time of one machine instruction on the (global) execution time of the whole task. For example, a cache miss may speed up the global execution, instead of slowing it down, if it prevents a more expensive branch misprediction. Besides such speculation-caused anomalies, there also can be instances of the well-known scheduling anomalies. One consequence of these anomalies is that no worst case initial execution state exists. For example, an initially empty cache may not be the worst case. This prevents safe timing measurements on isolated snippets of code. All this holds if and only if the processor architecture is sufficiently complex for timing anomalies to occur. It usually does not hold for simpler 8- and 16-bit processors.

Kopetz [Kop11, p. 247] continues to describe the state of practice in determining a reasonable upper bound for the WCET of a task. It includes the use of a restricted architecture that reduces the interactions among the tasks and the extensive testing of the complete implementation. He finishes that the state of current practice is not satisfactory. We add that Wilhelm *et. al.* [Wil+08] have presented satisfactory methods and tools; they are just not yet the general state of practice.

Cooling [Coo03, Chap. 14] surveys methods for performance engineering for real-time systems. That is, which design management approaches can help to achieve a desired performance.

3.2 Trends in Processor Architecture with Relevance to Timing Analysis

Wilhelm *et. al.* [Wil+08, Sect. 11.2] summarize current trends in processor architecture which are relevant to timing analysis:

"The hardware used in creating an embedded real-time system has a great effect on the ease of predictability of the execution time of programs.

The simplest case are traditional 8- and 16-bit processors with simple architectures. In such processors, each instruction basically has a fixed execution time. Such processors are easy to model from the hardware timing perspective, and the only significant problem in WCET analysis is how to determine the program flow.

There is also a class of processors with simple in-order pipelines, which are found in cost-sensitive applications requiring higher performance than that offered by classic 8- and 16-bit processors. Examples are the ARM7 and the recent ARM Cortex R series. Over time, these chips can be expected to replace the 8- and 16-bit processors for most applications. With their typically simple pipelines and cache structures, relatively simple and fast WCET hardware analysis methods can be applied.

At the high end of the embedded real-time spectrum, performance requirements for applications, like flight control and engine control, force real-time systems designers to use complex processors with caches and out-of-order execution. Examples are the PowerPC 750, PowerPC 7448, and ARM11 families of processors. Analyzing such processors requires more advanced tools and methods, especially in the hardware analysis.

The mainstream of computer architecture is steadily adding complexity and speculative features in order to push the performance envelope. [...] This mainstream trend of ever-more complex processors is no longer as dominant as it used to be, however. In recent years, several other design

alternatives have emerged in the mainstream, where the complexity of individual processor cores has been reduced significantly.

Many new processors are designed by using several simple cores instead of a single or a few complex cores. This design gains throughput per chip by running more tasks in parallel, at the expense of single-task performance. Examples are the Sun Niagara chip, which combines eight in-order four-way multithreaded cores on a single chip [OH05] and the IBM-designed PowerPC for the Xbox 360, using three two-way multithreaded in-order cores [Kre05]. These designs are cache-coherent multiprocessors on a chip and thus have a fairly complex cache and memory system. The complexity of analysis moves from the behavior of the individual cores to the interplay between them as they access memory.

Another very relevant design alternative is to use several simple processors with private memories (instead of shared memory). This design is common in mobile phones, where you typically find an ARM main processor combined with one or more DSPs on a single chip. Outside the mobile phone industry, the IBM–Sony-Toshiba Cell processor is a high-profile design using a simple in-order PowerPC core along with eight synergistic processing elements (SPEs) [Hof05]. The Cell will make its first appearance in the Sony PlayStation 3 gaming console, but IBM and Mercury Computing systems are pushing the Cell as a general-purpose real-time processor for high-performance real-time systems. The SPEs in the Cell are designed for predictable performance and use local program-controlled memories rather than caches, just like most DSPs. Thus, this type of architecture provides several easy-to-predict processors on a chip as an alternative to a single hard-to-predict processor.

Field-programmable gate arrays (FPGAs) are another design alternative for some embedded applications. Several processor architectures are available as "soft cores" that can be implemented in an FPGA together with application-specific logic and interfaces. Such processor implementations may have application-specific timing behavior, which may be challenging for off-the-shelf timing analysis tools, but they are also likely to be less complex and thus easier to analyze than general-purpose processors of similar size. Likewise, some standard processors are now packaged together with FPGA on the same chip for implementing application-specific logic functions. [...]

There is also work on application-specific processors or application-specific extensions to standard instruction sets, again creating challenges for timing analysis. [...]"

Chapter 4

Integrated Modular Avionics (IMA) for Aircraft

In this chapter, we give an overview of the Integrated Modular Avionics (IMA) architecture which is used for aircraft. We describe why it was developed, and which are the key properties of its constituent data network and of its operating system interface. Furthermore, we outline the extension to Distributed Modular Electronics (DME).

Our overview draws on the well-written background chapter of the recent dissertation thesis of Efkemann [Efk14, Chap. 2]. A good source for further reading is Ott [Ott07]. She gives a broad view on both architectures and development processes, and there in particular on testing processes.

4.1 From a Federated Architecture to the IMA Architecture

Efkemann [Efk14, Chap. 2] presents the following overview of Integrated Modular Avionics (IMA) in his dissertation thesis:

"The traditional *federated* aircraft controller architecture [Fil03, p. 4] consists of a large number of different, specialised electronics devices. Each of them is dedicated to a special, singular purpose (e. g. flight control, or fire and smoke detection) and has its own custom sensor/actuator wiring. Some of them are linked to each other with dedicated data connections. In the *Integrated Modular Avionics* (IMA) architecture this multitude of device types is replaced by a small number of modular, general-purpose component variants whose instances are linked by a high-speed data network. Due to high processing power each module can host several avionics functions, each of which previously required its own controller. The IMA approach has several main advantages:

- Reduction of weight through a smaller number of physical components and reduced wiring, thereby increasing fuel efficiency.
- Reduction of on-board power consumption by more effective use of computing power and electrical components.
- Lower maintenance costs by reducing the number of different types of replacement units needed to keep on stock.
- Reduction of development costs by provision of a standardised operating system, together with standardised drivers for the avionics interfaces most widely used.

- Reduction of certification effort and costs via incremental certification of hard- and software.

An important aspect of module design is *segregation*: In order to host applications of different safety assurance levels on the same module, it must be ensured that those applications cannot interfere with each other. Therefore a module must support resource partitioning via memory access protection, strict deterministic scheduling and I/O access permissions. Bandwidth limitations on the data network have to be enforced as well.

The standard aircraft documentation reference for IMA is ATA chapter 42. The IMA architecture is currently in use in the Airbus A380, A400M, the future A350XWB, and Boeing 787 Dreamliner aircraft. Predecessors of this architecture can be found in so-called fourth-generation jet fighter aircraft like the Dassault Rafale."

4.2 AFDX Data Network

Efkemann [Efk14, Chap. 2.1] continues with an overview of the AFDX data network employed by the IMA architecture:

"A data network is required for communication between (redundant) IMA modules as well as other hardware. This role is fulfilled by the *Avionics Full DupleX Switched Ethernet* (AFDX) network. It is an implementation of the ARINC specification 664 [Aer09] and is used as high-speed communication link between aircraft controllers. It is the successor of the slower ARINC 429 networks [Aer04].

AFDX is based on 100 Mbit/s Ethernet over twisted-pair copper wires (IEEE 802.3u, 100BASE-TX). This means it is compatible with COTS Ethernet equipment on layers 1 and 2 (physical layer and link layer). Ethernet by itself is not suitable for real-time applications as its timing is not deterministic. Therefore AFDX imposes some constraints in order to achieve full determinism and hard real-time capability.

In AFDX, so called *Virtual Links* (VLs) are employed for bandwidth allocation and packet routing. Each VL has a 16-bit ID, which is encoded into the destination MAC address of each frame sent through this VL. For each VL, only one end system can send frames, while there can be one or more receivers (unidirectional multicast communication, similar to ARINC 429). AFDX switches use a pre-determined configuration to deliver frames based on their VL ID to a set of receiving end systems.

Each VL is allocated a part of the full bandwidth of an AFDX link. To that end, each VL has two attributes: a maximum frame length (L_{max}) in bytes and a bandwidth allocation gap (BAG). The BAG value represents the minimum interval (in milliseconds) between two frames on that VL. Thus, the maximum usable bandwidth in bit/s of a VL can be calculated as:

$$b_{\text{max}} = L_{\text{max}} \cdot 8 \cdot 1000/\text{BAG}$$

End systems use a VL scheduler to ensure minimum latency and jitter for each VL.

[...]

In order to increase reliability, an aircraft data network consists of two independent switched networks. AFDX frames are sent on both networks. If no frame is lost, the other end systems will receive two frames. In order to identify matching frames sent over the redundant links, the message payload is followed by a sequence number field. Of two frames received on different networks with an identical sequence number, only the first is passed up the protocol stack."

Ott [Ott07, Chap. 1.6.3] provides a substantially more detailed overview of AFDX.

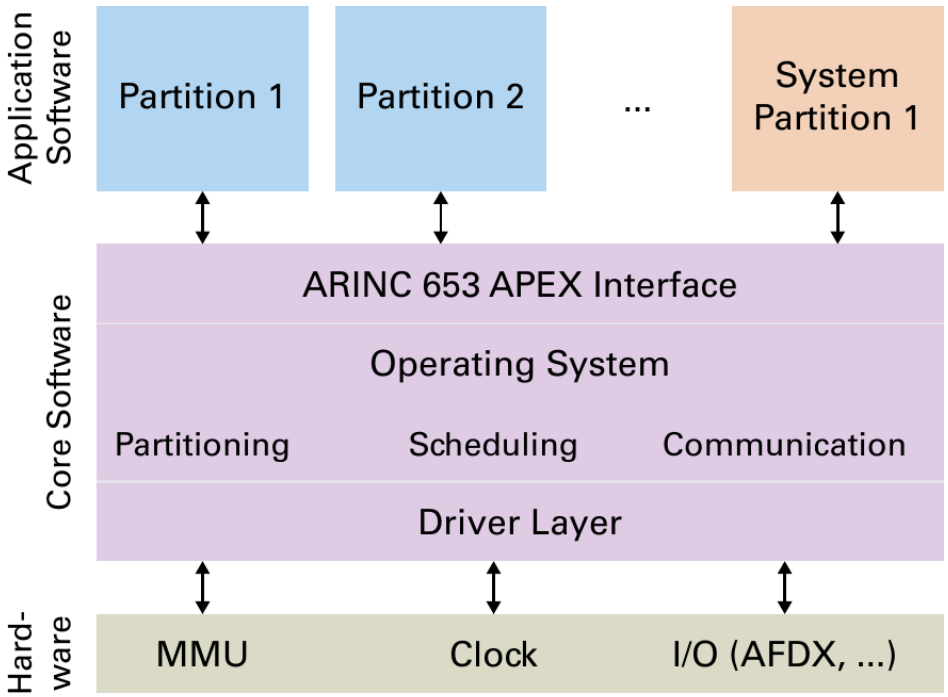

Figure 4.1: IMA module system architecture (taken from [Efk14, Fig. 2.2])

4.3 Operating System Interface ARINC 653

Efkemann [Efk14, Chap. 2.2] then introduces to the operating system interface ARINC 653 of the IMA architecture:

"[...], an IMA module can host multiple avionics functions. The interface between avionics applications and the module's operating system conforms to a standardised API which is defined in the ARINC specification 653 [Aer05].

The system architecture of an IMA module is depicted in figure 4.1. A real-time operating system kernel constitutes the central component. It uses a driver layer for access to the module's I/O hardware (either solely AFDX, or other interfaces like Discretes, CAN, and ARINC 429 buses as well, depending on module type) [Aer05, p. 11]."

Efkemann [Efk14, Chap. 2.2.1] details the partitioning as follows:

"In order to guarantee the same isolation of avionics functions residing on a shared module as a federated architecture would provide, it is the operating system's responsibility to implement a concept called *partitioning*. According to [RTC01], a partitioning implementation should comply with the following requirements:

- A software partition should not be allowed to contaminate another partition's code, I/O, or data storage areas.
- A software partition should be allowed to consume shared processor resources only during its period of execution.
- A software partition should be allowed to consume shared I/O resources only during its period of execution.
- Failures of hardware unique to a software partition should not cause adverse effects on other software partitions.

- Software providing partitioning should have the same or higher software level[1] than the highest level of the partitioned software applications.

On the IMA platform, a partition is a fixed set of the module's resources to be used by an avionics application. In particular, each partition is assigned a portion of the module's memory. The operating system ensures that other partitions can neither modify nor access the memory of a partition, similar to memory protection in a UNIX-like operating system. Each partition also receives a fixed amount of CPU time. The operating system's scheduler ensures that no partition can spend CPU time allotted to another partition [Aer05, p. 13].

Two kinds of partitions reside on a module:

Application partitions contain application code that makes up (part of) the implementation of an avionics function.

System partitions on the other hand provide additional module-related services like data loading or health monitoring.

Inside a partition there can be multiple threads of execution, called *processes*. Similar to POSIX threads, all processes within a partition share the resources allocated to the partition. Each process has a priority. A process with a higher priority pre-empts any processes with a lower priority. ARINC 653 defines a set of states a process can be in (Dormant, Waiting, Ready, Running) as well as API functions for process creation and management [Aer05, p. 18–25]."

In the following, Efkemann [Efk14, Chap. 2.2.2] discusses the methods of communication in length. A brief summary is:

"The operating system must provide a set of different methods of communication. All of them fall into either of two categories: intra-partition or interpartition communication. Intra-partition communication always happens between processes of the same partition, while inter-partition communication happens between processes of different partitions or even partitions on different modules. [. . .]"

4.4 Distributed Modular Electronics (DME)

Distributed Modular Electronics (DME) is an extension of the IMA concept. It was developed in the SCARLETT research project (SCAlable & ReconfigurabLe elEctronics plaTforms and Tool). SCARLETT was a joint European research and technology project of airframers, large industrial companies, SMEs, and universities [SCA13].

According to Efkemann [Efk14, Chap. 2.5], "the DME concept aims at the separation of processing power from sensor/actuator interfaces, thereby reducing the number of different component types to a minimum. This also makes DME suitable for a wider range of aircraft types by giving system designers the possibility to scale the platform according to required hardware interfaces and computing power. Figure 4.2 shows an example of a network of components: two Core Processing Modules (CPM), three Remote Data Concentrators (RDC), and two Remote Power Controllers (RPC) linked via two redundant AFDX networks. The CPM components provide the computing power and host the avionics applications, but apart from AFDX they do not provide any I/O hardware interfaces. Instead, the RDC and RPC components provide the required number of sensor/actuator and bus interfaces.

[1] as defined in [RTC92]

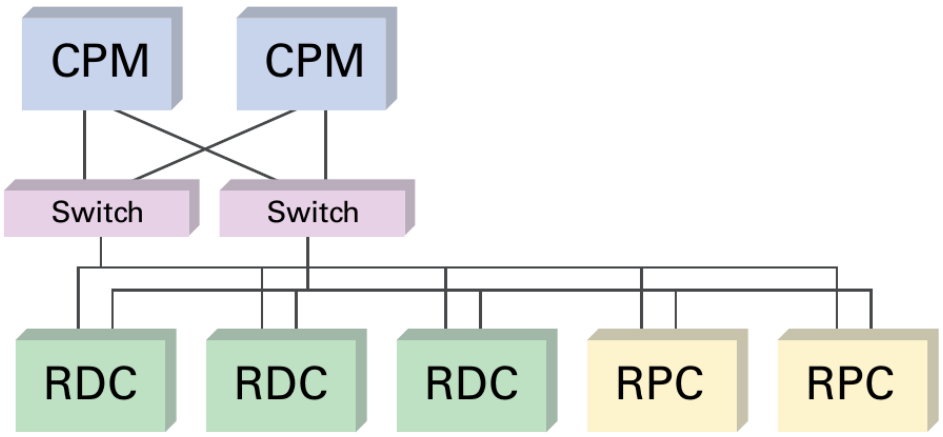

Figure 4.2: Distributed Modular Electronics (DME) architecture (taken from [Efk14, Fig. 2.3])

The project also investigates ways of increasing fault tolerance through different reconfiguration capabilities, for example transferring avionics functions from defective modules to other, still operative modules. Finally, the design of a unified tool chain and development environment has led to improvements of the avionics software implementation process. [. . .]"

Chapter 5

Differences between the Aeronautical and the Space Domain

There are significant differences between the aeronautical and the space domain, with respect to time and space partitioning.

5.1 The Speed of Growth of Complexity

Windsor and Hjortnaes [WH09, Sec. IV.B] state that the space domain mastered integrating applications on a single CPU earlier than the aeronautical domain. This was achieved by validating everything at the highest integrity level. But this now becomes more and more expensive due to the increasing complexity of mission applications. The Central Flight Software (CFS) encompasses the Data Management (DMS) and the Attitude & Orbital Control (AOCS) functional chains. Recently, they are integrated on one computer instead of independent computers linked by a data bus ([WH09, Sec. V.A]). Similarly, the payload software consists of several components of differing criticality: command and control has a higher criticality than payload data processing. These payload functions co-exist on one or several on-board computers (OBCs) and are under the responsibility of one or several organizations ([WH09, Sec. V.B]).

The complexity problems hit the space domain later than the aeronautical domain. We suppose the reason is that the space domain must use slower and therefore less powerful computers due to the harsh radiation environment in space (compare Sect. 5.6 below). Therefore, the space domain only recently reached the critical region of complexity.

Windsor *et. al.* [WDD11] provide some concrete numbers on complexity for the Airbus A380: its IMA platform is composed of up to 30 IMA modules of 8 different types, hosting 21 avionics functions which were developed by 10 function suppliers. In contrast, the IMA-SP architecture envisioned by Windsor *et. al.* will have less computing nodes (e.g., central on board computer, payload computer, and intelligent sensors and actuators), connected to a less powerful network based on SpaceWire or MIL-STD-1553B), and hosting fewer functions developed by small numbers of suppliers.

5.2 Scale of Communication Demands

The space domain appears to have significantly smaller demands on communication. Even though the International Space Station (ISS) features a complexity similar to that of an airplane, a typical satellite or launcher is much simpler; compare the end of the previous section. Accordingly, only a few hardware nodes need to be connected, with less bandwith. Windsor *et. al.* [WDD11] even explicitly consider the communication bus optional for the IMA-SP platform, for the case of a single hardware node. We add that the necessary redundancy against hardware failures nevertheless demands at least two hardware nodes. Depending on the redundancy concept, there might be little demand for them to communicate, however.

5.3 Online/Offline Maintenance

All aeronautical software maintenance is performed off-line, while the aircraft ist at the ground safely. In contrast, spacecraft operations require the system to be active ([WH09, Sec. VI.B].

Usually, a spacecraft is launched with the complete mission definition implemented in the flight software, from a pre-launch support mode to payload operations phase [WH09, Sect. V.D]. However, there is interest for adding new applications to extend the mission. This has been done in a few cases, but with extensive validation effort. And in the commercial satellite market, operators would like to have the in-flight capability to safely upload private payload applications to their spacecraft without the involvement of the manufacturer.

We add that the redundancy of multiple computer hardware may be used to take one of the computers offline in order to install a new software version. But this can be done for a short period of time only. And the new software version must be already validated fully. We have devised such an update process for the first in-space flight software update of the main computers of the European Columbus module which is part of the International Space Station [Bre08].

The IMA concept of the aeronautical domain requires substantial offline effort for reconfiguring communication paths. The virtual links (VLs) of the AFDX network are static. When the network routing shall be changed, each node affected must be accessed by a technician in order to apply this software update. This approach of IMA cannot be applied in the space domain, where physical access is impossible in nearly all cases.

5.4 Pronounced Mission Phases

Satellites have more pronounced mission phases than aircraft. Aircraft operate in different modes on ground, during ascent, cruise, and decent. Switches between these modes can occur rapidly. Long-lived satellites have longer-lasting and more predictable mission phases, such as ascent, orbit insertion, orbital payload operation, and deorbiting.

This offers the opportunity to reconfigure the computing resources between mission phases. At least, no computing time needs to be allocated to functions not active in the current mission phase. Since there is plenty of time after orbit insertion, the software even could be updated from ground, compare Sect. 5.3 above. See also [WDD11, page 8A6-4].

All of this does not apply to launchers, which are short-lived, of course.

5.5 Availability of a Hardware-Based Memory Protection Unit

In the space domain, only the most recent space qualified microprocessors (e.g., LEON2/3) have some kind of memory management unit (MMU) available [WH09, Sect. VI.E]. The Leon2 processor provides two memory write protection registers only. This is not sufficient if security is relevant, too. The Leon3 processor has a full MMU and also provides read-protection [Mas+10a]. (Sect. 6.3.1 on page 21 below provides more details of the Leon processors.)

Future work: describe current status concerning MPU in the aeronautical domain, in particular in IMA.

5.6 Radiation

In the space environment, the high level of hard radiation causes frequent malfunctions or even the permanent destruction of electronic circuits. This radiation could be shielded by a substantially massive casing only. Its launch weight usually prohibits this solution. The effect of radiation can be reduced by using electronic circuits with larger chip structures, too. However, larger chip structures also mean less functionality per chip and less speed. Therefore, less powerful computers can be used in space than on ground.

The atmosphere of the Earth shields most of this radiation, even for aircraft at high altitudes. Therefore, the restriction does not apply to the aeronautical domain. Accordingly, aeronautical hardware such as IMA modules cannot simply be taken and used in the space domain.

Higher computing power may be obtained in the space domain by giving up reliability and availability for some tasks, to a certain degree. For example, a radiation-hard, but slow computer can take care of the vital tasks of the spacecraft, while a much faster "number cruncher" computer can perform payload data processing, for example video encoding, even though it will crash a few times a day. In order to not affect the vital system adversely, such a number cruncher needs a bus interface with validated high dependability, only. However, such an approach with differentiated hardware does not match well the idea of interchangeable IMA hardware components.

5.7 Miscellaneous: Differences Between Launchers and Satellites

Inside the space domain, there are notable differences between launchers and satellites. Launch vehicles live a short time only. The first stage is spent in a few minutes, and the launcher's payload ist deployed within hours, often even less. In contrast, satellites usually last at least months, and often more than a decade.

Accordingly, any comparison between the aeronautical domain and the space domain should state whether it refers to launchers or to satellites, if this matters.

Chapter 6

Existing Work on Time and Space Partitioning Suitable for Space Avionics

In this chapter, we survey existing work on time and space partitioning that is geared towards or particularly suitable for space avionics.

6.1 The IMA-SP Project: an Adaption of IMA for Space Avionics

6.1.1 The Original IMA-SP Project

Integrated Modular Avionics for Space (IMA-SP) [WDD11; WH09] was a project of the European Space Agency (ESA). It aimed at incorporating the benefits of time and space partitioning, based upon the aeronautical IMA concept, into the spacecraft avionics architecture. Windsor and Hjortnaes [WH09] motivate the benefits of IMA-SP in general, and they give an overview of the approach chosen, but yet without concrete experiences and without a defined, concrete architecture. Windsor *et. al.* [WDD11] provide such an architecture and some experimental applications of it.

The IMA-SP architecture is a two-layer architecture, consisting of a System Executive layer and an application layer [WH09]. The System Executive includes a software kernel responsible for partition scheduling and communication services as well as handling hardware signals. Memory partitioning is ensured either by a Memory Management Unit (MMU) or by a (simpler) Block Protection Unit (BPU). A BPU prohibits access to memory (at a minimum, write access) outside of a partition's defined memory areas. Partitions are scheduled on a fixed, cyclic basis. The order of partition activation is defined at configuration time using configuration tables. This provides a deterministic scheduling scheme. Tasks within a partition can be scheduled statically or dynamically. Temporal and spatial partitioning therefore ensures each partition uninterrupted access to common resources and non-interference during their assigned time period.

IMA-SP defines the role of the system integrator explicitly [WH09, Sect. III.D]. They are responsible for the system design including the detailed on-board resource allocation; and they are responsible for the final integration and configuration of the components. Furthermore, there are the role of the platform supplier and the role of the application supplier.

A communication bus is optional for the IMA-SP platform, for the case of a single hardware node

[WDD11]. This reflects the significantly smaller demands on communication in the space domain compared to the aeronautical domain, see Sect. 5.2.

The IMA-SP approach customizes its architecture quite specifically to the requirements of the space domain. Windsor *et. al.* [WDD11] list the following "user requirements":

Mode requirements: Different partition scheduling plans shall be supported for different operational modes (e.g., initialization phase, operational phase, safe/survival phase).

Allocation of basic physical resources: The resource allocation scheme shall be defined and predictable.

Time services: The IMA-SP platform shall provide On Board Time (OBT) services.

On Board Events: Applications may raise and access on-board operational events, such as launcher separation and alarms (for FDIR: Failure Detection, Isolation, and Recovery).

Access to on-board data stores: There shall be on-board data stores for data generated between communication windows with ground, for mission programming data (e.g., long-term mission time line and telecommand files), for context data required to resume operations after a recovery action, and for software images and patches.

Flight software maintenance: The applications and the kernel shall be maintainable via standard operations (dump, patch, and update).

Fault protection: The IMA-SP platform shall support an FDIR service both at partition level and at system level.

Application communication interface: Applications may interact with the platform or other applications through and only through well defined interfaces.

Observability requirements: Ground shall have the possibility to reconstruct from telemetry the conditions leading to a change in a partition's state or an operational event being raised.

We think that these "user requirements" must make the resulting architecture rather specific for a narrow application area. The IMA-SP project apparently did not do a generalization step by identifying common requirements of the aeronautical domain and the space domain first, before adding the space specific requirements. Instead, the project put an emphasis on preserving long-proven ideas, approaches, and even hardware from the space domain. Therefore, it has become less visible which conceptual changes are necessary for the transfer of IMA from the aeronautical domain to the space domain, and what are just customizations to a the specific application area. The aeronautical domain dared a more radical change of its ways when introducing IMA, compared to what the IMA-SP project is prepared to do.

We even think that the approach is tailored more to satellites than to launchers. Launchers have no opportunity to do flight software maintenance, and often there is no time for recovery from a safe mode.

Windsor *et. al.* [WDD11] describe the architecture of the IMA-SP platform. It is composed of:

- hardware node(s)
- a System Executive Platform (SEP), consisting of
  - a partitioning kernel,

  - a TSP abstraction layer, and
  - (optionally) guest OSs inside the partitions
- system support services (e.g., for I/O) inside dedicated partitions
- application support services (e.g. implementations of standard protocols)

The services of the TSP abstraction layer were derived from the ARINC 653 [Aer05] API of the IMA architecture [WDD11]. In this, re-use of existing space concepts, which are similar to ARINC 653, took precedence over the (literal) incorporation of the ARINC 653 specification. Windsor *et. al.* describe the similarities and differences in detail [WDD11].

One example for a difference is that IMA-SP adds to IMA a set of services which allow to exchange data between partitions via shared memory. This is a mandatory addition. We think that this was driven by rather specific applications in mind. There appears to be no generalizing stock-taking of communication mechanisms which are or are not necessary for different application areas.

Three SEPs took part in the IMA-SP study: AIR (an RTEMS modification to support time and space partitioning by GMV, Portugal, see Sect. 6.3.4 below), PikeOS (a commercial microkernel from SYSGO, Germany, used in the the aeronautical domain, see Sect. 6.4.1 below), and XtratuM (an open source hypervisor from the University of Valencia, Spain, see Sect. 6.3.1 below). These SEPs have been ported to the LEON3 Sparc microprocessor which is the ASIC targeted for future European spacecraft [WDD11]. See the following sections for more details on these three SEPs.

Windsor *et. al.* [WDD11] report on three use cases which the IMA-SP study planned to investigate. In these use cases, it was planned to apply and try out the IMA-SP approach practically. Windsor presents some results in a talk at ADCSS 2012 [Win12]. The feedback on the system executive platforms was:

"**XtratuM:**

Pro: product maintenance & adaptability – good level of maturity – OBT synchronisation

Cons: Missing GDB debug tool – bootloader

Pike OS:

Pro: Extensive documentation – Development tools very good – Easily configurable – Debug available on TSIM

Cons: No tracing mechanism – Missing GDB debug tool – RTEMS interrupt management not implemented

Edisoft RTEMS:

[it was] para-virtualised for all SEP kernels"

And the conclusion of the talk included:

- "AIR, PikeOS and XtratuM have been ported to LEON2/3 and pre-validated
  - PikeOS offers DO-178 certification datapack
- Lessons learnt from Use Cases
  - Mastering role definition and process is key to success
  - IO management with current HW is challenging but not impossible
  - Execution Platform support tools & test bench needed
  - HW improvements have been identified"

Silva *et. al.* [SCS12] report on the development of an input/output component for IMA-SP, by GMV, Portugal. The aim is to be able to reuse the drivers for at least some system resources. It supports RS-232, UDP on Ethernet, SpaceWire and MIL-STD-1553B, in beta version quality.

The original IMA-SP project ended in December 2012.

6.1.2 Follow-Up Assessment Study on Partitioning and Maintenance of Flight Software in IMA-SP

Hardy *et. al.* [HHC14] report on a follow-up assessment study on partitioning and maintenance of flight software in IMA-SP. They ported a small satellite flight software to the Xtratum hypervisor on a (simulated) Leon3 processor. As some of the lessons learned, they provide experiences with the hypervisor used (XtratuM, compare Sect. 6.3.1 below) and with the guest operating system used (RTEMS, compare Sect. 6.3.4 below).

Concerning XtratuM, a few additions were made to XtratuM in order to support the boot scheme, the software maintenance approach, and the board chosen by the project. The authors report a bug found in the version 3.4.2 of XtratuM used as a major issue: A division-by-zero exception in the floating point unit of the processor halted the entire system. Thus, it could break the partitioning. Minor issues include that the virtualization of the registers of the LEON processor does not seem to be implemented correctly and completely. We conclude that the XtratuM version used did not yet meet quality criteria for a high level of dependability.

Concerning RTEMS, the authors found two bugs which they classified as major issues. One of them led to unpredictable scheduling inside the RTEMS partition. Both have been fixed in the latest version of RTEMS. However, the port of this version to XtratuM was still missing, at least at the time the presentation was made.

6.1.3 The IMA-SP System Design Toolkit Project

The IMA-SP System Design Toolkit project is a follow-up project to the original IMA-SP project. Hann *et. al.* [Han+15] describe its first phases (2014 to 2015). The project defined a data model, with associated files and file formats, describing the complete setup of a partitioned system and allowing for a system feasibility assessment; it developed a prototype of a tool set, called the IMA-SP System Design Toolkit (SDT); and it plans to demonstrate the toolkit on a case study.

The system feasibility assessment means to show that the partitioning and resource allocation meet all constraints and requirements. These include timing, fault containment, communication, and memory allocation. The system feasibility assessment is performed by the system integrator [Han+15].

The data model consists of, according to [Han+15]:

the partition model: partition properties, activities/tasks, communication ports, memory requirements, and resource requirements

the physical data model: memory, cache, interrupts, and peripheral devices

the IMA-SP system data model: allocation of physical resources to the partitions

the partitioning kernel data model: partitioning kernel memory requirements, health monitoring events and actions, default health monitoring system configuration table, and default health monitoring partition configuration table

the partitioning kernel data model – configuration: health monitoring table, scheduling plan, connection table, and memory configuration table

the common data model: basic types of the data model

The IMA-SP SDT data model is specified using the "ecore" format, of the Eclipse Modelling Framework (EMF) project [Ste+08]. The model files are in XML Meta-data Interchange (XMI) format [ISO14].

Hann *et. al.* [Han+15] also further describe the system design work flow and the design of the IMA-SP System Design Toolkit.

6.2 The SAVOIR-IMA Initiative: A TSP Based Software Reference Architecture

SAVOIR (space avionics open interface architecture) is an initiative by the European Space Agency (ESA) which aims at improving the ways in which the European space community builds avionics sub-systems. It it geared towards satellites, thus excluding launchers. The initiative defined a reference avionics architecture for spacecraft platform hardware and software in general. The reference architecture either uses a "classic" execution platform or an execution platform providing time and space partitioning. SAVOIR is organized in specialized working groups. Two of them are SAVOIR-FAIRE on the software reference architecture in general and SAVOIR-IMA on the TSP based software reference architecture. [Hjo14]

Hiller and Hernek [HH13] report on the results and roadmap of the SAVOIR-IMA initiative. There are working groups and activities on:

- use cases and system requirements
- terminology
- reference architecture and interface description (COrDeT-3, SIFSUP)
- methods and tools for on-board software engineering
- adding multi core support to AIR (MultIMA)
- SW elements for security
- on-board software reference architecture for payloads (OSRA-P)
- IMA system design toolkit
- preparation for SEP kernel qualification package

The SIFSUP activity (Objectives of Savoir-IMA and Savoir-Faire Support) is, amongst others, on the difference in paradigm between IMA-SP (Time and space partitioning) and the On-board Software Reference Architecture (OSRA) (a component-based architecture on top of a centralized Execution Platform). [Jun14a]

On OSRA, see, e.g., Jung [Jun14b]. It does not cover time and space partitioning specifically.

Future work: find out more on SAVOIR-IMA.

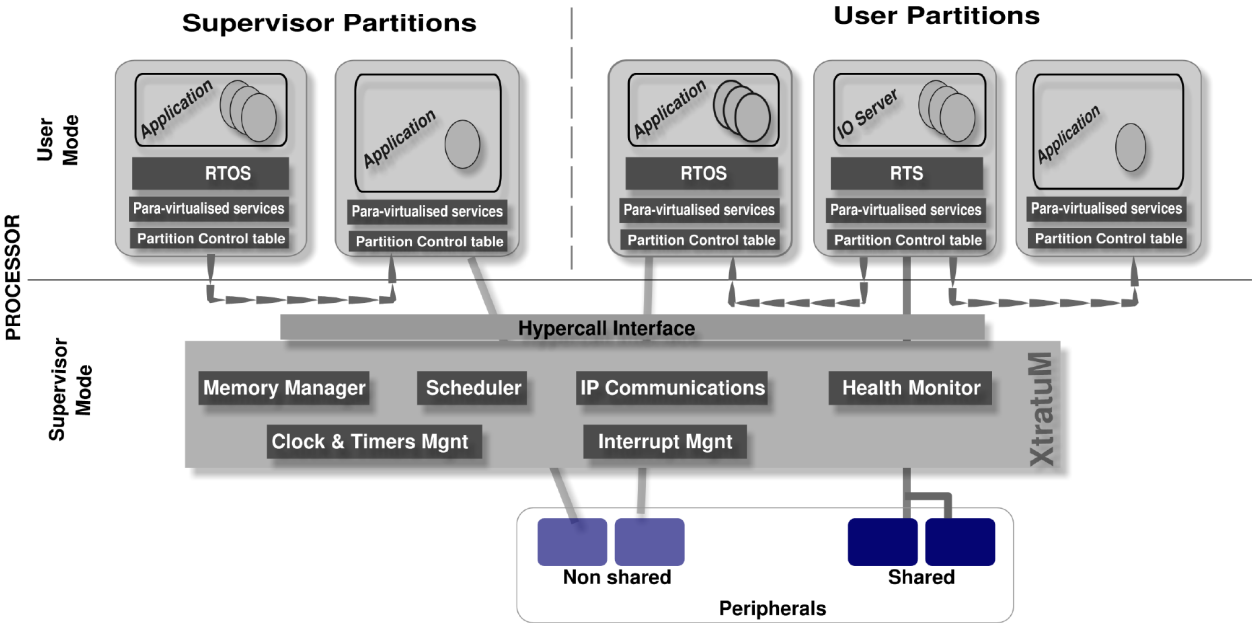

Figure 6.1: The XtratuM architecture, taken from [Pei+10, Fig. 1].

6.3 Virtualization Solutions Suitable for Space Avionics

In this section, we survey virtualization solutions suitable for space avionics. They are the bare-metal hypervisor XtratuM and the partition management kernel AIR.

6.3.1 XtratuM: a Hypervisor for Safety-Critical Embedded Systems

XtratuM is a bare-metal hypervisor which implements para-virtualization and dedicated device techniques [Pei+10; Mas+09; Cre+09]. It was designed to achieve time and space partitioning for safety-critical embedded systems.

We already presented the notions of *virtualization in general*, of a *bare-metal hypervisor*, and of *para-virtualization* in Sect. 2.3.2 above.

Figure 6.1 shows an overview of the XtratuM architecture [Pei+10]. XtratuM executes in the supervisor mode of the processor. The applications execute in the user mode, each in its own partition. The hypervisor XtratuM applies a static scheduling scheme to grant execution time of the processor to the partitions. This achieves the separation of the partitions in the time domain. Each partition gets its statically determined share of the execution time. No user partition can overrun or change the schedule. Timer interrupts are caught and handled by the hypervisor.

Similarly, the hypervisor XtratuM allocates areas of memory to the partitions. It catches any illegal access to memory addresses outside a partition's memory space. To be precise, this is true only for processors providing suitable hardware support. The Leon2 processor does not have a memory management unit (MMU), which could translate virtual memory addresses into physical memory addresses. The Leon2 provides two memory write protection registers only. Therefore, XtratuM can enforce write protection among partitions, but not read protection. This is sufficient in order to meet safety requirements, but it is not sufficient to enforce security against malicious software in a partition. On processors providing a MMU, such as the Leon3 processor, XtratuM provides read protection, too [Mas+10a]. The Leon processors are designed for and used in a space environment, in particular with an increased level of radiation.

A peripheral device can be associated to a specific partition. In this case, no other partition may

access this peripheral device. This is similar to memory protection.

Some partitions may be special, these are called system partitions or supervisor partitions, in contrast to the user partitions. The system partitions are allowed to manage the other partitions, for example by stopping and resuming them via calls to the hypervisor. However, since these partitions run in the user mode of the processor, too, they cannot break the time and space isolation described above.

The XtratuM hypervisor provides a hardware abstraction to the partitions similar to the aeronautical ARINC 653 standard [Aer05] API of the IMA architecture (compare Sect. 4.3). However, these two interfaces have several differences. The most notable difference is that communication in the IMA architecture is based on the fast AFDX data network (compare Sect. 4.2), while XtratuM does not prescribe any particular network technology. There even can be no inter-computer network at all. This is probably due to the difference between the aircraft and the spacecraft domain, that communication demands are often lower in the latter domain, compare Sect. 5.2.

Each partition appears as a normal, dedicated computer to the software running in it. Accordingly, a partition can contain a bare application (no operating system at all, just an infinite loop), a general-purpose operating system, or a real-time operating system. In case there is some operating system, an application in a partition may have several processes/threads, as usual. The para-virtualization approach requires an adaption of the software inside the partitions, however: some privileged machine instructions in the lower layer of the operating system (or in the bare application) must be substituted by calls to the hypervisor, as discussed in Sect. 2.3.2.

Operating systems that have been ported to XtratuM include LithOS and RTEMS (see Sect. 6.3.4 on RTEMS below), and also Linux, PaRTiKle, and ORK+ [FEN+13]. The real-time operating system LithOS [Mas+10b] was designed to provide an ARINC 653 inspired API. LithOS adds multi-process support, communication between processes, and a process scheduler to the services provided by XtratuM. However, there is still no mandatory AFDX network with LithOS. Similarly, PaRTiKle is an open source real-time kernel for embedded systems, distributed under the terms of the GNU Public License; PaRTiKle has been initially developed by the University of Valencia, Spain [FEN+13]. ORK+ (Open Ravenscar Kernel) is a small, high performance real-time kernel that provides restricted tasking support for Ada programs [FEN+13].

The processors on which XtratuM has been implemented include the Intel x86 processor family, supporting multiprocessors [Mas+09; Cre+09], the Leon2 processor [Pei+10], the Leon3 processor [Mas+10a], the Leon4 processor [Cre+14; MCC12], and the ARM Cortex R4 [Cre+14]. XtratuM has been adapted to deal with heterogeneous multicore architectures in the MultiPARTES project, see Sect. 6.3.3 below.

XtratuM is open software distributed under the Gnu Public Licence version 3. Some ancillary tools are sold under a proprietary licence by FentISS [Fen16], a spin-off of the University of Valencia, Spain.

The configuration of a specific software image must be described in a central configuration file. XtratuM's system integration tools compile this configuration file together with the application software images into the system software image. This image then is loaded onto the processor an run.

The central XtratuM configuration file is in XML format and can be written by hand. Alternatively, the configuration file can also be generated using the Xoncrete tool. Xoncrete is a graphical tool to assist the system designer when configuring the resources (memory, communication ports, devices, processor time, etc.) allocated to each partition. It has two parts: a resource editor and a scheduling analysis tool [Bro+10]. The scheduling analysis tool provides support for the complexities of hierarchical scheduling (i.e., scheduling the partitions and scheduling tasks inside

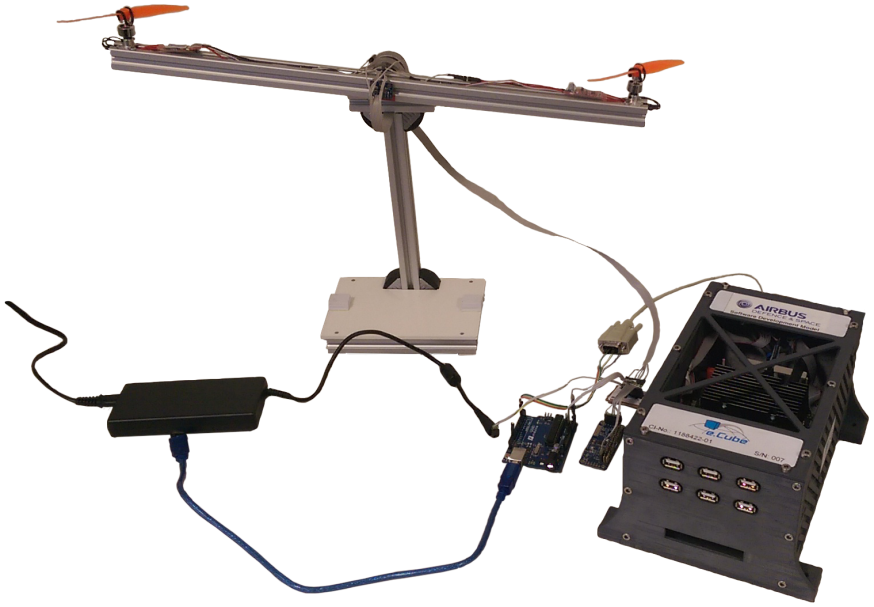

Figure 6.2: The attitude control demonstrator.

the partitions) [Bro+10; Rip+10].

Maturity: The study of Hardy *et. al.* [HHC14] (see Sect. 6.1.2 above for details) indicates that the XtratuM version 3.4.2 used there did not yet meet quality criteria for a high level of dependability.

6.3.2 Our Case Study Using XtratuM for a Portable Real-Time Application

We performed a case study using XtratuM [Ehr+15]. It used an e.Cube computer designed by AirbusDS for space missions, featuring an x86-family Atom processor. The application was an attitude control demonstrator with one degree of freedom, see Fig. 6.2. The application ran on the Atom processor inside one of the XtratuM partitions. The application ran bare-metal, only using XAL (the XtratuM Abstraction Layer), without an operating system. (It had turned out that the port of the Linux operating system to XtratuM didn't run anymore in the then current version of XtratuM.) The application had real-time properties. It had to be formally proven that these real-time requirements were met by the implementation running on top of XtratuM.

The project was performed in a collaboration of the City University of Applied Sciences Bremen with AirbusDS, Bremen.

The goals of the project included gaining familiarity with real-time systems and in particular with real-time proofs. The proofs based on the idea of determining the worst-case execution time by testing with exhaustive path coverage. Furthermore, the effort necessary for achieving portability of the software (including the real-time proofs) to other hardware architectures was of interest.

The project provided experience with XtratuM. We used XtratuM version 3.7.3, current as of September 2014. It turned out that the adaption of XtratuM to the specific main board could be solved only partially due to missing documentation of the details of the main board used. Configuring the software image for the hardware and the applications needed substantial effort. It would probably have been of considerable help if the Xoncrete tool for XtratuM would have been

available for this.

Preparatory work for the real-time proofs was performed. But due to delays because of the hardware adaption problems mentioned, the full proof was not carried out anymore.

In retrospect, using a hypervisor like XtratuM has the advantage that several applications with different criticality can run safely on a single processor; but there were no indications that the hypervisor would ease porting the application and its real-time proof to another hardware architecture. The reason for this is that the hypervisor does not abstract away from the details of the hardware used, they remain fully visible to the application. The hypervisor only hides the other partitions running on the same processor. We did not attempt to port the application to another processor anymore in the course of the project. But both the compiled application binary and the real-time proof depend on the specifics of the processor. Of course, the application can be compiled for a different processor. But using a hypervisor makes no difference in this. The same holds for adapting the real-time proof to another processor and its timing characteristics.

6.3.3 The MultiPARTES Project: an Extension of XtratuM for Multi-Core Processors

The MultiPARTES project (Multi-cores Partitioning for Trusted Embedded Systems) adapted XtratuM to deal with heterogeneous multicore architectures [TCA13; CA14]. Besides the adaption of XtratuM, the project also defined a development methodology and provided supporting tools.

The hardware platform consists of two different systems: a dual core x86 based processor (Atom Core Duo at 1.7 GHz) and an FPGA with several synthesized LEON3 processors. The x86 subsystem provides comparably high computation capabilities, and the LEON3 processors provide a hardware base for time-predictable computations.

Therefore, the challenges of multi-core CPUs to time partitioning (compare Sect. 7.1.1 on page 30 below) have been responded to by simply using several independent LEON3 CPUs for the time critical computations. However, these LEON3 CPUs have been integrated onto a single FPGA chip (and under a single hypervisor) at least, thus reducing weight and other resource demands.

Trujillo *et. al.* [TCA13] discuss mixed-criticallity systems in their description of the MultiPARTES project, but they do not mention the specific challenges of a space environment, like radiation and the restrictions on computing power caused by it (compare Sect. 5.6 on page 15 above). Some further reading on XtratuM in the MultiPARTES project is [CC14; FEN+13; Mas+11].

6.3.4 AIR: A Partition Management Kernel Based on RTEMS

The AIR and AIR-II projects developed a partition management kernel based on the RTEMS operating system kernel.

RTEMS

RTEMS (Real-Time Executive for Multiprocessor Systems) is a real-time operating system kernel. Originally designed for military applications, it is now used in a wide area of application domains, including the space domain, in particular. It is open software distributed under a license close to the GNU General Public License. RTEMS provides multi-tasking, intertask communication, different kinds of scheduling, and support for homogeneous and heterogeneous multiprocessor systems. [RTE16b]

RTEMS is not a partitioning kernel. Thus, it does not support time and space partitioning. However, RTEMS can be and has been used as a real-time operating system inside a partition. (For example, Ripoll *et. al.* report on a use of RTEMS inside XtratuM [Rip+10], and Hardy *et. al.* [HHC14] report on another such use of RTEMS in XtratuM.)

Versions and maturity: Edisoft RTEMS (version 4.8.0) has been qualified for space use by the European Space Agency [Sil+10]. According to Silva *et. al.* [Sil+10, Sect. 4], "the qualified version and tools are distributed as open source free package". The online RTEMS Centre [Edi09], however, states: "All the Downloads in the RTEMS CENTRE website are restricted to authorized users (the authorization is granted by the project administration)." Furthermore, the RTEMS Centre has been inactive since 2010. The improvements of Edisoft RTEMS appear to not have been merged back into the further RTEMS development, according to some discussions on the Web. The current released version of RTEMS is 4.10.2 from December 2011. In November 2014, the RTEMS Project services have been relocated from OAR Corporation to the Oregon State University Open Source Lab [RTE16c].

The study of Hardy *et. al.* [HHC14] (see Sect. 6.1.2 above for details) hints that the quality of the Edisoft RTEMS software needs further investigation. However, RTEMS appears to have been used on several real space missions, according to its Web site [RTE16a] (unverified).

AIR

The European Space Agency studies AIR and AIR-II (**A**RINC **I**nterface in **R**TOS – Industrial Initiative) developed several components for time and space partitining [RF07b; RF07a]. They comprise [Sch11]

- a partition management kernel,
- support libraries, drivers, etc.
- operating system APIs to be used inside of a partition,
- a configuration and compilation tool chain, and
- analysis tools.

Figure 6.3 on the next page shows the AIR architecture. It allows to sepate applications into different partitions, each with their own memory space and own time budget. Inter-partition communication is by queueing ports and sampling ports. All this is similar to XtratuM, see Sect. 6.3.1 above. But there are differences, too.

The preferred operating system to be used inside a partition ("personality") is RTEMS, possibly extended by the ARINC 653 API ("APEX"). The partition management kernel internally uses RTEMS as a hardware abstraction layer, too. Therefore, there are close links from AIR to RTEMS, even though any operating system can be run inside a partition, or even a bare application. [Sch11]

The AIR project used RTEMS version 4.6.6 [RF07a].

Only the so-called "system" partitions may access hardware devices and thus perform input/output. The corresponding drivers run in kernel mode, but are scheduled as "co-partitions". Co-partitions share execution windows with their client partitions, up to a pre-defined percentage ("sharing quota"), iff critical tasks have terminated ("sharing barrier"). Efficient inter-partition communication via shared memory is possible, too. [Sch11]

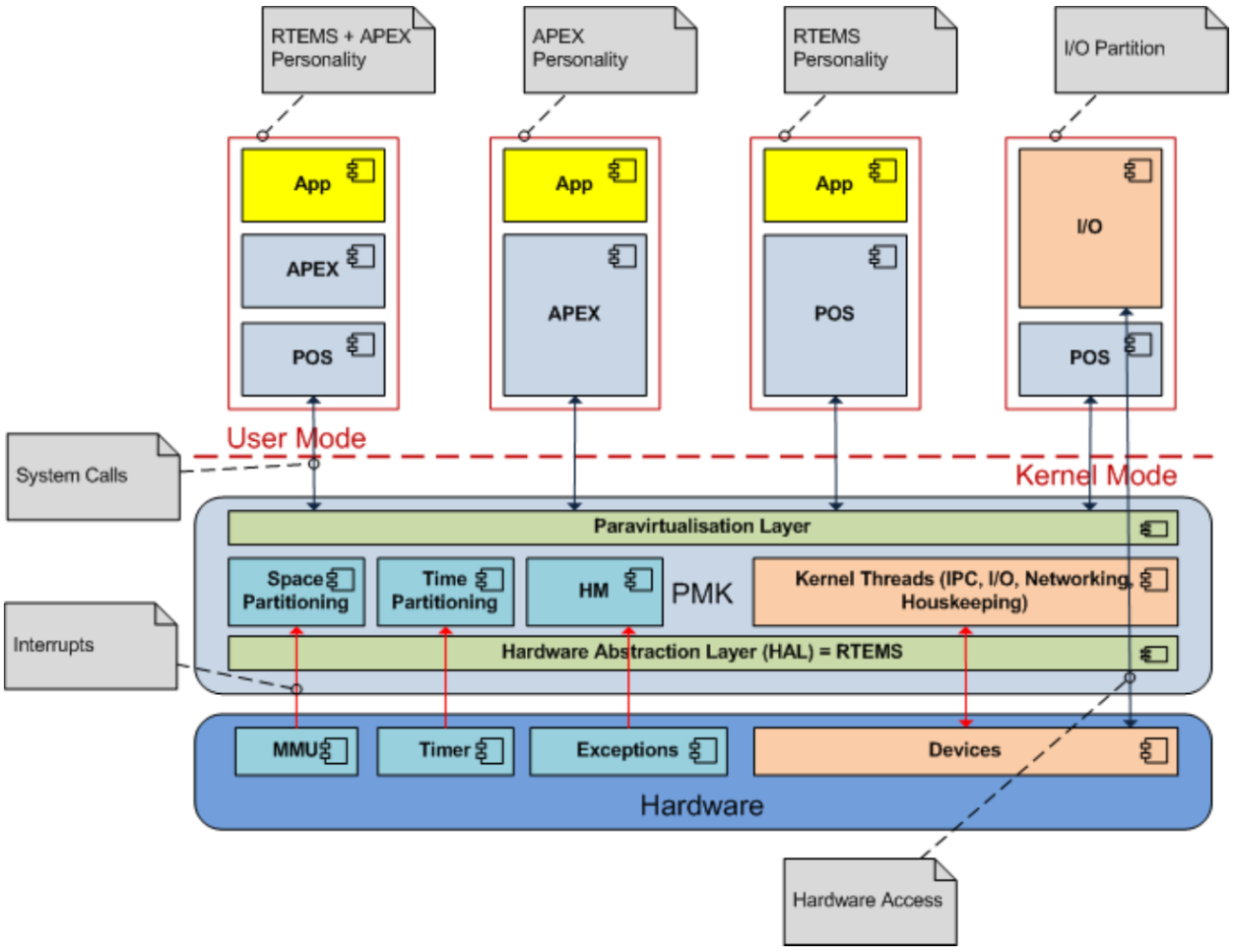

Figure 6.3: The AIR architecture (taken from [Sch11]).

The partition management kernel is a micro-kernel. There are no kernel threads. Instead, the concurrency of drivers is implemented through the co-partitions, which are scheduled with their client partitions. [Sch11]

Silva *et. al.* [SCS12, Sect. 3.1] note that the co-partition approach is more efficient than having regular partitions for input/output, but that this comes not without a cost. A system that allows a partition to be pre-empted before the end of its execution window in order to perform I/O operations is more difficult to analyse, qualify, and in perspective, to fully guarantee as predictable.

The studies were performed by GMV, Portugal, together with the University of Lisbon, Portugal, and Thales-Alenia Space. The initial target processor architecture was Sparc-Leon. [Sch11]

Santos *et. al.* [San+08] describe in detail the implementation of an ARINC 653 API on an underlying POSIX layer. The AIR-II consortium decided not to implement an entirly new in-partition operating system to provide ARINC 653 API services to hosted applications, but instead to build the ARINC 653 API on top of available real-time operating systems.

Future work: Evaluate the AIR project reports [RF07b; RF07a] and the theses resulting from the AIR-II project [Cra09; Ros11]. Possibly look for even more literature on the subject.

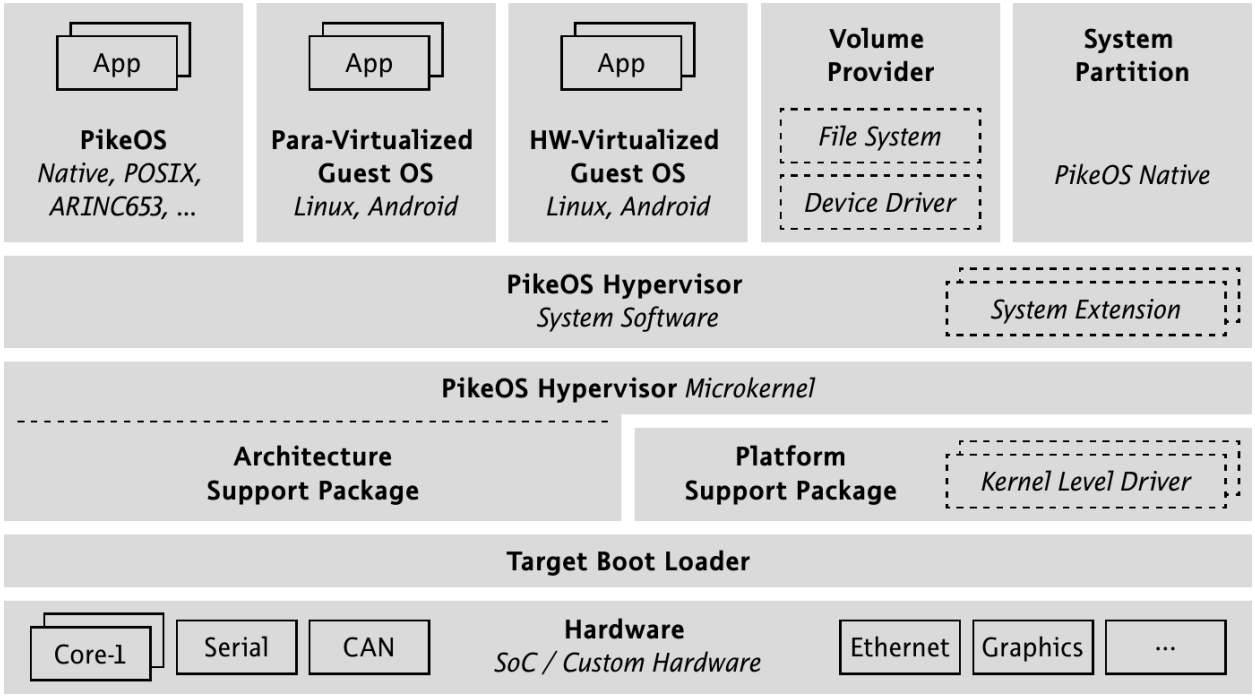

Figure 6.4: PikeOS architecture (taken from [SYS16, Fig. 1])

6.4 Separation Kernels Suitable for Space Avionics

In this section, we survey separation kernels suitable for space avionics. These comprise PikeOS, VxWorks 653, LynxSecure, and POK.

6.4.1 PikeOS

PikeOS is a separation kernel for real-time systems in safety-relevant and security-relevant domains, such as the aeronautical domain and the space domain [SYS16]. Alternatively, it also can be viewed as a virtualization solution plus an optional guest operating system. Figure 6.4 shows the architecture of PikeOS. PikeOS is a commercial product by Sysgo, Germany.

According to the product datasheet [SYS16], the PikeOS Hypervisor runs on x86 as well as ARM, PowerPC, SPARC V8/LEON or MIPS and can be adapted to other CPU types. The virtualization concept supports multi-core architectures. PikeOS is completely developed according to safety standards such as DO-178B, IEC 61508, EN 50128, ISO 26262 or IEC 62304. The available guest operating systems, runtime enviroments and APIs are: PikeOS, Linux, Android, ARINC 653, AUTOSAR, RTEMS, legacy RTOS, POSIX, Realtime Java, ADA, and others. Optimized implementations such as ARINC 664 (AFDX) and CAN are available for PikeOS Native partitions. There is a development tool chain called CODEO. It is an Eclipse-base IDE with configuration tools, remote debugging with operating system awareness, target monitoring, remote application deployment, and timing analysis tools.

Future work: Further evaluate PikeOS.

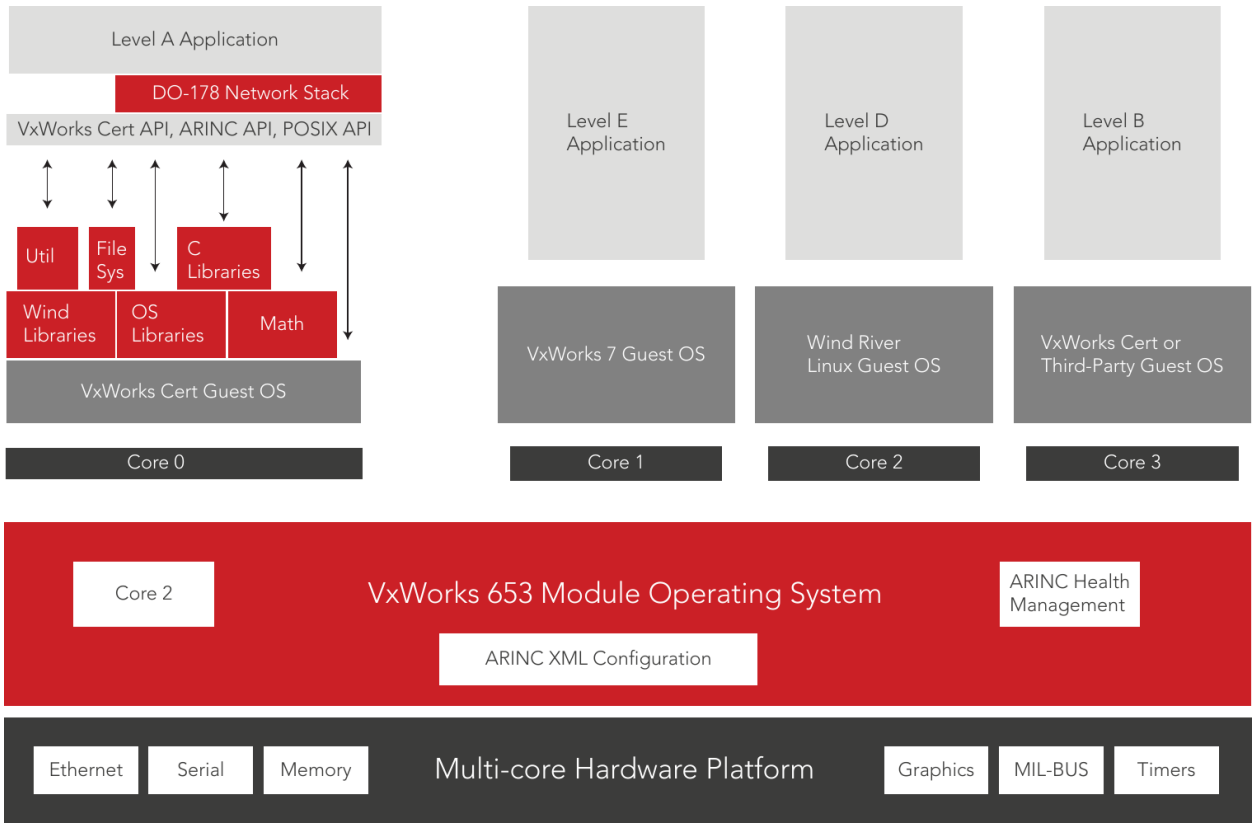

Figure 6.5: VxWorks 653 architecture (taken from [PK15, Fig. 2])

6.4.2 VxWorks 653

VxWorks 653 is a real-time operating system for safety-critical domains, such as the aeronautical domain, supporting IMA standards, in particular the ARINC 653 standard [PK15]. VxWorks 653 is that member of the VxWorks real-time operating system family that provides a separation kernel. Figure 6.5 shows the architecture of VxWorks 653. VxWorks 653 is a commercial product by Wind River, which is a subsidiary of Intel.

According to a white paper by the manufacturer [PK15], there is a *module operating system* providing global resource management, scheduling, and health monitoring. There is also a VxWorks *partition operating system* providing scheduling and resource management within a partition. There is no mention of running bare-metal applications without any operating system inside a partition. Accordingly, a guest operating system inside a partition appears to need adaptions to run under VxWorks 953. VxWorks 653 provides an option for priority preemptive scheduling of partitions. This permits *slack stealing* by allowing designated partitions to consume what would otherwise be idle time in the defined ARINC schedule. The VxWorks 653 3.0 Multi-core Edition supports multi-core processors. However, certification of this is still under review by authorities in both the FAA and EASA. The operating system comes with a tool chain including an Eclipse-based workbench, the simulator Simics, and further development, system configuration, and debugging tools. The white paper also mentions some security related mechanism of VxWorks 953, but no comprehensive security concept. VxWorks 653 is used for many avionics systems and safety-critical applications, including systems of the Boeing 787 Dreamliner and of the Airbus A330.

Future work: Further evaluate VxWorks 653.

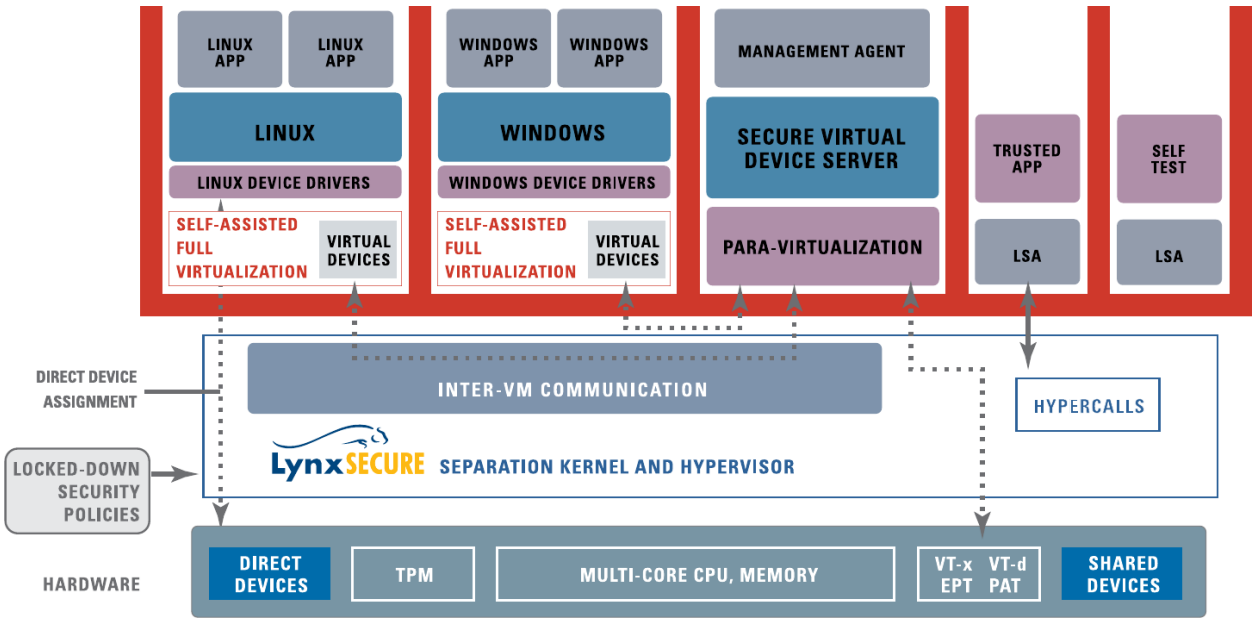

Figure 6.6: LynxSecure architecture (taken from [Lyn16, Fig. 3])

6.4.3 LynxSecure

LynxSecure is a real-time operating system for security-relevant systems with a separation kernel [Lyn16]. Its emphasis is mainly on security and only to a lesser extent on safety. It supports multi-core processing. Figure 6.6 shows the architecture of LynxSecure. LynxSecure is a commercial product by Lynx Software Technologies, CA, USA.

Future work: Further evaluate LynxSecure.

6.4.4 POK

POK ("a partitioned operating system") is an open-source real-time embedded operating system intended for safety-critical systems such as aeronautical systems [DL11], providing a separation kernel. The project appears to be dormant since 2012. Nevertheless, we briefly report its main features here.

According to Delange and Lec [DL11], the tool Ocarina is an important feature of POK, which serves to configure and deploy applications based on descriptions in the Architecture Analysis and Design Language (AADL). There is also a tool to enforce modelling rules, safety rules, and security rules. POK supports the x86, PowerPC, and SPARC/LEON architectures. It has been used by different research projects.

6.5 Component Base Software Architectures

Watney *et. al.* [WRC14] write on partitioning and multi-core flight software systems. They are mostly interested in component frameworks. The aspect of criticality and the aspect of the high effort for validating a monolithic system is missing. They authors use the object oriented language C++. However, they do not discuss how to meet timing constraints and memory constraints while using object orientation. The key word "multi-core" appears in the title of the paper. But it is not substantiated in the body of the text.

Chapter 7

Research Challenges

In this chapter, we collect some open problems in the area of time and space partitioning. They deserve further research.

7.1 CPU-Related Challenges for Time Partitioning

Current CPUs pose challenges to time partitioning. They are, in particular, the intricacies of multi-core CPUs and of direct memory access.

7.1.1 Multi-Core CPUs

Current multi-core CPUs make it difficult to achieve true time partitioning. Advances in chip technology allow for more transistors per chip than a CPU core requires. Therefore, the trend is to integrate several CPU cores onto a single chip, in order to increase the computing power per chip further. However, these cores are not totally separate CPUs, but they share some resources for efficiency reasons. In particular, often they share memory caches. A cache can improve the speed of execution considerably, if the data to access is already present in the cache.

A cache shared between CPU cores therefore causes a dependency of the worst-case execution time of a piece of code running on one core on the behaviour of code running on another core. If the other core has flushed some data of the first core from the cache, the first core will run slower. Using a cache for a core therefore does not improve the worst-case execution time, and it makes it much harder to determine this worst-case execution time. A simple solution is to disable all caches. But this comes at a speed penalty that might be larger than the gain by using more than one core in the first place.

In the case of a single-core CPU, a cache can be used without destroying temporal separation of the partitions. On every partition change, the cache must be flushed. If partition changes occur in a deterministic way, the remaining gain by the cache is guaranteed. Furthermore, a defined initial (cache) state and determinism together prevent timing anomalies to occur (compare Sect. 3.1). In the case of a multi-core CPU, this is more complicated, since the partition changes of the different cores usually do not happen at the same points in time, or coordinated at all.

Several research projects already investigate this subject. Some of them are listed by Crespo *et. al.* [Cre+14, Sect. 2.3] (even though none of them primarily addresses the space domain).

Future work: make an overview of research work on multi-core TSP.

7.1.2 Direct Memory Access (DMA)

There are also other components than further CPU cores that compete with a first CPU core for the memory bus. Direct memory access (DMA) is a technique by which a dedicated controller transfers data from and to a mass memory storage device, without involving the CPU. Nevertheless, a DMA controller can slow down the CPU by contending for the memory bus. If at least one of the partitions uses DMA, all of them can be slowed down by this effect. Therefore, a non-real-time partition can affect a real-time partition adversely in this way.

Future work: Have a closer look into DMA and its bus use. See also [SCS12, Sec. 4.4] on this.

7.2 Challenges for Real-Time Property Proofs

Proving real-time properties poses many challenges, including those from processor architecture, virtualization, and distributed computing.

7.2.1 Processor Architecture

The following two processor architecture aspects are relevant when validating real-time properties: worst-case performance and timing anomalies.

Worst-Case Performance and Processor Architecture

Conventional processor architectures aim to optimize the average-case performance. However, when validating real-time properties, this usually is irrelevant. Instead, the worst-case performance matters. Consequently, hard real-time tasks require an entirely different processor architecture design. A processor feature that improves the average execution time, but not the worst case execution time, does not help to meet hard real-time constraints. When choosing a processor architecture, it should be left away, if not needed otherwise.

We continue this line of thought in Sect. 8 below.

Future work: Look for possible literature on processor architectures for hard real-time proofs.

Timing Anomalies and a Processor Architecture for Space Use

Timing anomalies occur in complex processor architectures, and they complicate real-time property proofs greatly. This is because they make it much more difficult to determine the worst-case execution time (compare Sect. 3.1). In consequence, a found upper bound can become so loose that one can find a lower upper bound for a simpler processor. Then, the simpler, less capable processor becomes the better choice.

The space domain uses simpler processors than other domains because they have to be more robust with respect to radiation. Selecting or designing a processor architecture for space use that is not susceptible to timing anomalies may prove useful for achieving acceptable real-time performance and validating it.

We continue this line of thought in Sect. 8 below, too.

7.2.2 Virtualization

Virtualization poses additional challenges when performing a real-time property proof. Virtualization aims at presenting a virtual machine to an application software which is exactly like the real machine. However, it cannot hide that the machine instructions are not executed evenly in the time sense anymore (compare Sect. 2.3.2). Any real-time property proof must account for this. If the hypervisor employs a static cyclic scheduling, this can be done.

Future work: Survey the literature on real-time proof techniques and tools which can handle virtualization.[1]

A separation kernel is another mechanism to achieve time and space partitioning. A real-time proof for code running under a separation kernel must take its scheduling policy into account, as well as the timing of calls to the separation kernel.

Future work: Survey the literature on real-time proof techniques and tools for separation kernels. In particular, approaches for the IMA architecture in the aeronautical domain (compare Chap. 4) are interesting.

The interplay between the scheduling of a hypervisor or of a separation kernel on the one hand with the scheduling of a communication bus on the other hand poses its own difficulties, which result in corresponding challenges with respect to the according real-time property proofs. Silva *et. al.* [SCS12] found that scheduling partitions that communicate via the MIL-STD-1553B bus poses a particular challenge. The design of the MIL-STD-1553B bus protocol assumes that the communicating system is scheduled synchronously to the bus schedule. Additionally, the bus schedule is at a high rate, usually in the order of dozens of Hertz. This combination of properties is difficult to achieve when independent systems shall be integrated onto a single platform using virtualization. The global schedule of the hypervisor can be synchronized to at most one external schedule.

7.2.3 Distributed Computing

Distributed Modular Electronics (DME, compare Sect. 4.4) and other approaches which combine time and space partitioning with distribution add corresponding challenges to proving real-time properties. Here, we need to determine the reaction time of a computation which is distributed over several processing nodes. In general, there will not even exist a global scheduling cycle, such that the local scheduling latencies will add up with their worst-case values.

Brocal *et. al.* [Bro+10] and Ripoll *et. al.* [Rip+10] present work on a special case, in the context of the Xoncrete tool for the XtratuM hypervisor. They consider an End-to-End flow (ETEF). This is a sequence of tasks with temporal attributes. The tasks can belong to different partitions. However, this approach assumes that all tasks run under the same supervisor, and that they are scheduled by a single global scheduling scheme.

Future work: Survey the literature on real-time proof techniques and tools for distributed systems.

We performed a case study on grid computing using space hardware [Ber+14]. The project was performed in a collaboration of the City University of Applied Sciences Bremen with Astrium, Bremen (now AirbusDS). The study aimed at providing currently unused computing resources in a network of space computers to computing-intensive tasks, such as image recognition. The

[1] The Xoncrete tool for the XtratuM hypervisor (compare Sect. 6.3.1) is *not* such a tool. Xoncrete can perform a scheduling analysis in the presence of virtualization, but it cannot do a timing analysis for a snippet of code running on a specific processor. The worst-case execution time resulting from such an analysis must be fed as input to Xoncrete. [Bro+10]

means for distributing the tasks were grid computing software tools. However, grid computing intrinsically is a best effort approach, making the most out of the computing resources available currently. Therefore, despite other merits, grid computing probably is no viable way for providing high-speed real-time computing.

7.3 Separation Kernel vs. Virtualization for TSP

As discussed in Sect. 2.3.3, both a separation kernel and virtualization are mechanisms to achieve time and space separation. It should be investigated which mechanism has which relative pros and cons, under which conditions.

7.4 Identification of the Common Properties of the Aeronautical and the Space Domain With Respect to TSP

As already discussed in Sect. 6.1.1, the original IMA-SP project apparently did not do a generalization step by identifying common requirements of the aeronautical domain and the space domain first, before adding the space specific requirements. Instead, the project put an emphasis on preserving long-proven ideas, approaches, and even hardware from the space domain. Therefore, it has become less visible which conceptual changes are necessary for the transfer of IMA from the aeronautical domain to the space domain, and what are just customizations to a the specific application area.

Accordingly, it is still an open research challenge to identify the common properties of the aeronautical and the space domain with respect to time and space partitioning.

7.5 Adaption of IMA-SP to Launchers

In addition to the previous section, the original IMA-SP project is tailored more to satellites than to launchers (see Sect. 6.1.1). Launchers have no opportunity to do flight software maintenance, and often there is no time for recovery from a safe mode. Compare Sect. 5.7 and also Sect. 5.4.

Consequently, the results of the original IMA-SP project could be adapted to the peculiarities of launchers.

7.6 An Existing Survey on Research Challenges for Mixed-Criticality Systems

Crespo *et. al.* [Cre+14, Sect. 2.2] summarize some technical reports dealing with the future challenges of mixed-criticality systems (MCSs). They are:

System modelling: Notations are required for describing a model of the system under development: for the functional components, the partitioning, and the deployment.

Methodology and development tools: Additional activities, such as partitioning and system integration, require according extensions of the development methodology. An example is

the "system integrator" role. The methodology extensions must be complemented by tool support.

Scheduling techniques for MCS: Partitioned systems demand solutions for the incremental scheduling of partitions.

Support for multi-core platforms: The cores of multi-core platforms interfere because of shared resources such as L2/L3 cache, memory, bus, IO, etc.; this must be addressed.

We discussed this aspect in Sect. 7.1.1 above.

Crespo *et. al.* [Cre+14, Sect. 2.3] list nine research projects in this area (ACROSS, ARAMiS, CERTAINTY, IMA-SP, MCC, MultiPARTES, perMERASA, RECOMP, and vIrtical). However, they provide the projects' URLs only instead of references to publicized work. IMA-SP ist the only of these projects where the primary application domain is the space domain. All of these projects except IMA-SP include the "multi-core" aspect in some way.

7.7 Probably Not: Hierarchical Scheduling for Mixed-Criticality Systems

In the context of virtualization (compare Sect. 2.3.2 above), we often get the problem of hierachical scheduling. On the one hand, the partitions are scheduled by the hypervisor, and on the other hand, the operating systems inside the partitions schedule their processes/tasks.

According to Ripoll *et. al.* [Rip+10, Sect. 5], there are several open research questions on hierachical scheduling. There is no decision procedure yet for some of the more complex scheduling strategy combinations, whether a given system is schedulable.

However, we think that any proof of separability of the partitions depends on the simplicity of the architecture. It might be tempting to improve the efficiency of a system by a new, elaborate hierachical scheduling scheme. But this must decrease our trust in the composed system. Even if there is some proof for the new scheme, its implementation is more prone to errors than the implementation of a simple scheme.

Chapter 8

Idea: A Processor Architecture for Time Partitioning in the Space Domain

Based on some of the research challenges in the previous chapter, we propose to design a multi-core processor architecture that avoids the problems of the current, ever more powerful multi-core processors with respect to time partitioning, and that can be used in the space domain.

8.1 The Proposed Processor Architecture

As we have seen in the previous chapter, the increasing complexity of modern processor architectures poses increasing challenges to time partitioning in the space domain. Some of the reasons are shared caches in multi-core CPUs and timing anomalies in complex processor architectures. A more radical approach to this than the approaches in the previous chapter is to design a processor architecture with less obstacles for time partitioning.

A fundamental cause of the increasing problems is the following: the advances in chip technology allow for an ever larger integration of tasks on a single chip. This allows to have several tasks processed on a single such chip, provided we can guarantee that they don't interfere.

A first answer to this was time and space partitioning. But the increasing integration also allowed for *optimizations transcending a single processing unit*, such as shared caches and out-of-order execution. These optimizations create problems when the ultimate goal is to execute the tasks without any interference among them.

A more substantial answer therefore is to take the advantages of modern chip techonology, but to design a processor architecture that allows to execute several small tasks in parallel and without any interference among them. In this context, transcending optimizations are not helpful and even have an adverse effect. Therefore, they should not be part of such an architecture.

8.2 Implementing a Custom Processor Architecture for a Small Market

How can we implement such an architecture at affordable costs? The market for space computers is small. We cannot expect a mass production of custom microcontrollers.

The answer is that modern chip technology also offers field programmable gate arrays (FPGAs).

kind of task	kind of core assigned
large task	large core
small task	small core
several very small tasks	small core with time and space partitioning

Table 8.1: Assigning a CPU core to a task.

These are generic chips which can be loaded with an arbitrary digital design by an end user. The gates and flipflops in an FPGA are fixed, but the connections among them are programmable. Furthermore, a wealth of IP cores is available (complete digital designs to be loaded as building blocks into an FPGA), including many CPU designs from a broad spectrum of architectural complexity. Therefore, it is feasible to combine several such ready-made CPU cores onto a single FPGA, according to our own architectural considerations.

8.3 Employing the Architecture for Time Partitioning in the Space Domain

How can we assign tasks with different computing power demands to CPU cores? When the capabilities of a CPU core roughly match the demands of a task, we can simply allocate the task to this single CPU core. When a task has large demands on computing power, we can include a CPU core with correspondingly larger capabilities into the FPGA. When there are several tasks with small demands on computing power, we can assign them together to a simple CPU core, which must not have the problematic complexities discussed above, and employ a time and space partitioning regime. Table 8.1 summarizes this.

Designing such a custom-tailored FPGA requires that the size of the tasks is static and known beforehand. (However, dynamic reconfiguration of FPGAs is available for some models. But we suspect that the validation effort for this approach would be prohibitive.)

If the properties of tasks are more dynamic, a more generic and less custom-tailored processor architecture can be employed: it provides a few classes of CPU cores with different computing power. A task is then assigned to a roughly matching CPU core, as described above.

8.4 Discussion of the Approach

The key idea of this architecture is that there are no performance optimizations which transcend the individual CPU cores, or the individual processing units inside a core. This is different to current multi-core CPUs. They are optimized to increase the average overall performance, at the price of bad performance in rare worst case scenarios, and at the price of interactions which let the costs for temporal analysis explode.

Employing a set of independent cores for a set of tasks can be viewed as a return from a TSP architecture to a federated architecture (compare Sect. 4.1). At first glance, this appears as a contradiction. But it does make sense. The disadvantage of the orginal federated architectures was the overhead of having several computers, and that the CPUs had become so powerful that using them for a single task was a waste. Both arguments are not true when using an architecture of federated, tailored, independent cores on a single chip. It is important that these cores are independent from each other. This avoids the time separation problems that appear when using

conventional multi-core CPUs. The resulting line of the proposed historical development is:

1. several federated, independent single-core CPUs
2. TSP on one more powerful single-core CPU
3. several federated, independent cores in one even more powerful multi-core CPU (possibly employing TSP on a few of these cores for very small tasks)

The use of conventional multi-core CPUs, where the cores are not independent, appears to be a difficult path, when it comes to validate real-time properties.

An FPGA is more expensive than an off-the-shelf CPU with the same computing power. But aligning the processor architecture more to the problems of time partitioning and temporal analyzability might very well be the cheaper solution, in the end.

8.5 Related Research

There is some research related to our approach, already.

8.5.1 Non-Mainstream Trends in Processor Architecture

In Section 3.2, we surveyed trends in processor architecture with relevance to timing analysis. One (non-mainstream) trend is that many new processors are designed by using several simple cores instead of a single or a few complex cores.

Another such trend is to use several simple processors with private memories instead of shared memory. (For more details, see Sect. 3.2.) In particular the latter trend in general processor design is related to our proposed processor architecture for the space domain.

8.5.2 The T-CREST Project: a Time-Predictable Multi-Core Architecture

One particlular project is particularly close to our ideas described above: the T-CREST project developed a time-predictable multi-core architecture which eases temporal analysis [Sch14].

The project's idea starts out with the observation that worst-case execution time analysis is about ten years behind current processors, and that multiprocessors are therefore currently not analyzable. The project then designs a new computer architecture, where the worst-case execution time is the main design constraint, and the average-case performance is not (so) interesting. The entire multiprocessor is put onto a single chip, employing an FPGA.

This system-on-a-chip (SoC) comprises

- newly-designed, bare-essentials processor nodes, with local memory (caches and scratchpad memory)
- a network-on-a-chip (NoC), applying deterministic time-division multiplexing
- an SDRAM memory controller node

Figure 8.1 on the following page presents an overview of the T-CREST architecture.

Integral part of the project was the co-design of the processor node, the compiler, and the worst-case execution time analysis tool.

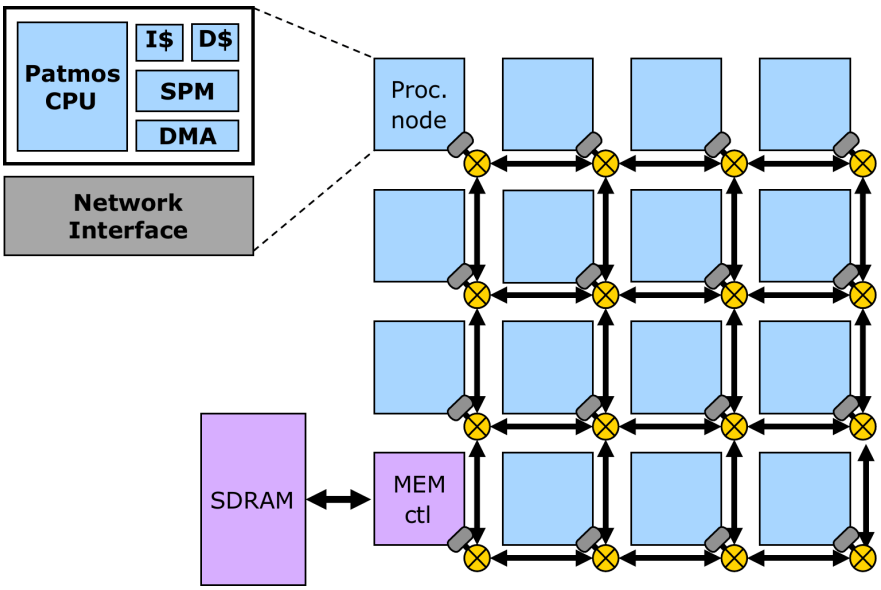

Figure 8.1: T-CREST architecture (taken from [Sch14, p. 9])

Most artifacts are open-source, and usage and collaboration are welcome, according to Schoeberl [Sch14]. Quite a number of further papers were published, on individiual aspects of the project, according to its web site [Han16]. The project war partially funded by the EU, and it ran from 2011 to 2014.

One result from the project is its experience with the network-on-a-chip. The time-division multiplexing employed wastes bandwith, but there is plenty of bandwith on-chip. Therefore, this is no problem, since only bandwith relative to cost matters. Having everything on a single chip is a crucial advantage here.

The T-CREST project touches only some aspects of our idea described above. All processing nodes are the same in T-CREST, the aspect of tailoring the core to the task is not covered. Furthermore, the project stops at the compiler level, no operating system research is done. Therefore, the time and space partitioning aspect is excluded, too.

Chapter 9

Sketch of an Idea: a Processor Architecture Specifically Supporting WCET Measurements

The following is a further idea, the potential of which we did not yet evaluate, due to limited time. Current proof tools for real-time properties determine the worst-case execution time (WCET) of a piece of code on a specific processor by simulating it, compare Sect. 3.1 on page 5. All relevant execution paths are simulated. Of course, a simulation is orders of magnitude slower than the execution on the real hardware.

When we think about adapting the processor architecture to the problem of temporal analysis, there is a further way for doing this: We could use the real hardware for determining the WCET, if there is hardware support for initializing rapidly the run of each execution path.

There has been similar hardware support built into microcontrollers, since long, for debugging the functional behaviour of the code. But it might be interesting to adapt this to support WCET measurements specifically.

Future work: look for research in this direction maybe already done.

Chapter 10

Summary

We presented a survey of the current state of the reseach on time and space partitioning for space avionics. There is already a body of existing work, it has been presented in Chap. 6:

- the IMA-SP project, which is an adaption of the aerospace Integrated Modular Avionics (IMA) architecture for space avionics
- the SAVOIR-IMA initiative, which is a time and space partitioning based software reference architecture
- virtualization solutions suitable for space avionics like XtratuM and AIR
- separation kernels suitable for space avionics like PikeOS, VxWorks 653, and others
- further work, e.g., on component base software architectures

But substantial research challenges remain to be tackled, as shown in Chap. 7:

- CPU-related challenges for time partitioning like multi-core CPUs and direct memory access (DMA)
- challenges to performing real-time property proofs, including those from processor architecture, virtualization, and distributed computing
- an investigation of the mechanisms separation kernel and virtualization with respect to time and space partitioning
- the identification of the common properties of the aeronautical and the space domain with respect to time and space partitioning
- an adaption of IMA-SP to launchers
- further ideas from an existing survey on research challenges for mixed-criticality systems

In contrast to others, we rather don't think it is helpful to investigate hierarchical scheduling for mixed-criticality systems.

Based on some of the above research challenges, we propose to design a multi-core processor architecture that avoids fundamental problems of the current architectures with respect to time partitioning, and that can be used in the space domain (Chap. 8). One key idea is to optimize

the processor architecture for the worst case, instead of for the average case. Another important idea is to strip away optimizations transcending a single processing unit, such as shared caches and out-of-order execution, since they make real-time property proofs much harder, due to the interferences they introduce. Such an architecture can be implemented even for a small market such as the space domain, by using field programmable gate arrays (FPGAs). We discussed how the capabilities of the CPU cores can be matched to the demands of the tasks to perform. For small tasks, this can include time and space partitioning, again.

Bibliography

[Aer04] Aeronautical Radio, Inc. *Digital Information Transfer System (DITS), Part 1, Functional Description, Electrical Interface, Label Assignments and Word Formats. ARINC Specification 429P1-17 Mark 33*. May 2004.

[Aer05] Aeronautical Radio, Inc. *Avionics Application Software Standard Interface, Part 1 – Required Services. ARINC Specification 653P1-2*. Dec. 2005.

[Aer09] Aeronautical Radio, Inc. *Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet. ARINC Specification 664P7-1*. Sept. 2009.

[Alv+05] Jim Alves-Foss et al. 'The MILS Architecture for High-Assurance Embedded Systems'. In: *Intl. Journal of Embedded Systems* 2 (3-4 Feb. 2005), pp. 239–247.

[Avi+04] Algirdas Avižienis et al. 'Basic Concepts and Taxonomy of Dependable and Secure Computing'. In: *IEEE Trans. on Dependable and Secure Computing* 1.1 (Jan.–Mar. 2004).

[Ber+14] Andreas Bergmeier et al. *Grid-Computing in der Raumfahrt*. German. Project report. Univ. of Applied Sciences Bremen, Germany, 2nd Feb. 2014.

[Bre08] Jan Bredereke. *Flight SW Transition to Cycle 11*. ESO-IT-TN-0116. Version 2. Internal technical note. Astrium GmbH. 26th Aug. 2008.

[Bro+10] Vicent Brocal et al. 'Xoncrete: a scheduling tool for partitioned real-time systems'. In: *5th Intl. Conf. and Exhibition on Embedded Realtime Software and Systems, ERTSS-2010*. (Toulouse, France). Ed. by Gérard Ladier and Jean-Luc Maté. 19th–21st May 2010. URL: `http://www.fentiss.com/documents/xoncrete_overview.pdf`.

[CA14] Alfons Crespo and Alejandro Alonso. 'MultiPARTES – Partitioning Heterogeneous Multicores'. In: *2nd Int'l workshop on the Integration of mixed-criticality subsystems on multi-core and many-core processors at HiPEAC Conference 2014*. (Vienna, Austria). 20th–22nd Jan. 2014. URL: `http://138.4.11.199:8080/multipartes/public/HiPEAC/MPT-MCS-Viena2014.pdf` (visited on 09/06/2016).

[CC14] Javier Coronel and Alfons Crespo. *Mechanisms for hardware virtualization in multicore architectures. Xtratum Integration on Bespoke TTNoC-Based HW. Virtualization layer design and implementation*. Version v1.1. MultiPARTES project. 5th June 2014. URL: `https://alfresco.dit.upm.es/multipartes/public/MPT-D6.5.2_Virtualization%20layer%20design%20and%20implementation_v1.1.pdf`.

[Coo03] Jim Cooling. *Software Engingeering for Real-Time Systems*. Addison Wesley, 2003.

[Cra09] João Craveiro. 'Integration of Generic Operating Systems in Partitioned Architectures'. MSc. thesis. University of Lisbon, Portugal, 2009. URL: `http://air.di.fc.ul.pt/air-ii/downloads/Craveiro09MSc.pdf`.

[Cre+09] A. Crespo et al. 'XtratuM: an open source hypervisor for TSP embedded systems in aerospace'. In: *Proc. of DASIA 2009 Data Systems In Aerospace*. (Istanbul, Turkey, 26th–29th May 2009). Ed. by L. Ouwehand. SP-669. ESA Spacebooks Online, Aug. 2009. URL: http://www.fentiss.com/documents/dasia09.pdf.

[Cre+14] Alfons Crespo et al. 'Mixed Criticality in Control Systems'. In: *Preprints of the 19th World Congress of the International Federation of Automatic Control*. (Cape Town, South Africa). 24th–29th Aug. 2014.

[DL11] Julien Delange and Laurent Lec. 'POK, an ARINC653-compliant operating system released under the BSD license'. In: *Proc. of the 13th Real-Time Linux Workshop*. (Prague, Czech Republic). 20th–22nd Oct. 2011. URL: https://lwn.net/images/conf/rtlws-2011/proc/Delange_POK.pdf (visited on 09/06/2016).

[Edi09] *Homepage of the RTEMS Centre*. Edisoft. 2009. URL: http://rtemscentre.edisoft.pt/ (visited on 01/02/2016).

[Efk14] Christof Efkemann. 'A Framework for Model-based Testing of Integrated Modular Avionics'. PhD thesis. Univ. of Bremen, Germany, 6th Aug. 2014. urn:nbn:de:gbv:46-00104131-10.

[Ehr+15] Chris Daniel Ehrenberg et al. *Plattformunabhängige Softwareentwicklung für eingebettete Echtzeitsysteme unter Verwendung eines Hypervisors*. (Platform Independent Software Development for Embedded Real-Time Systems Using a Hypervisor). German. Project report. Univ. of Applied Sciences Bremen, Germany, 9th Feb. 2015.

[FEN+13] FENTISS et al. *Tool chain implementation*. Version v1.0. MultiPARTES project. Feb. 2013. URL: https://alfresco.dit.upm.es/multipartes/public/MPT-D4.3-V10-final.pdf.

[Fen16] *Homepage of FentISS*. Fent Innovative Software Solutions. 2016. URL: http://www.fentiss.com/ (visited on 12/01/2016).

[Fil03] Bill Filmer. 'Open systems avionics architectures considerations'. In: *Aerospace and Electronic Systems Magazine, IEEE* 18 (9 Sept. 2003), pp. 3–10. DOI: 10.1109/MAES.2003.1232153.

[Han+15] Mark Hann et al. 'System Design Toolkit for Integrated Modular Avionics for Space'. In: *Proc. of DASIA 2015 Data Systems in Aerospace*. (Barcelona, Spain, 19th–21st May 2015). ESA Special Publication ESA SP-732. Sept. 2015.

[Han16] Scott Hansen. *T-CREST project website*. The Open Group. 2016. URL: http://www.t-crest.org (visited on 23/06/2016).

[HH13] Martin Hiller and Maria Hernek. 'Integrated Modular Avionics: SAVOIR-IMA status and progress'. In: *7th ESA Workshop on Avionics, Data, Control and Software Systems, ADCSS 2013*. 22nd Oct. 2013. URL: https://indico.esa.int/indico/event/22/session/8/contribution/13/2/material/slides/0.pdf.

[HHC14] Johan Hardy, Martin Hiller and Philippe Creten. 'Partitioning and Maintenance of Flight Software in Integrated Modular Avionics for Space'. In: *TEC-ED & TEC-SW Final Presentation Days 2014*. ESA ESTEC. 21st–22nd May 2014. URL: https://indico.esa.int/indico/event/57/contribution/4/material/slides/0.pdf.

[Hjo14] Kjeld Hjortnaes. 'Introduction to SAVOIR'. In: *8th ESA Workshop on Avionics, Data, Control and Software Systems, ADCSS 2014*. (Noordwijk, The Netherlands). Ed. by Kjeld Hjortnaes, Alain Benoit and Philippe Armbruster. 27th–29th Oct. 2014. URL: https://indico.esa.int/indico/event/53/session/1/contribution/3/material/1/.

[Hof05] H. Peter Hofstee. 'Power efficient processor architecture and the Cell processor'. In: *Proc. of the 11th Intl. Symp. on High-Performance Computing Architecture, HPCA-11 2005*. (12th–16th Feb. 2005). 2005, pp. 258–262.

[ISO14] *Information technology - Object Management Group XML Metadata Interchange (XMI)*. Standard ISO/IEC 19509:2014(E). Apr. 2014. URL: http://doc.omg.org/formal/2014-04-06.pdf.

[Jun14a] Andreas Jung. 'Focus on the Execution Platform – Introduction'. In: *8th ESA Workshop on Avionics, Data, Control and Software Systems, ADCSS 2014*. (Noordwijk, The Netherlands). Ed. by Kjeld Hjortnaes, Alain Benoit and Philippe Armbruster. 27th–29th Oct. 2014. URL: https://indico.esa.int/indico/event/53/session/6/contribution/20/material/1/.

[Jun14b] Andreas Jung. 'On-board Software Reference Architecture (OSRA) – Introduction'. In: *8th ESA Workshop on Avionics, Data, Control and Software Systems, ADCSS 2014*. (Noordwijk, The Netherlands). Ed. by Kjeld Hjortnaes, Alain Benoit and Philippe Armbruster. 27th–29th Oct. 2014. URL: https://indico.esa.int/indico/event/53/session/5/contribution/14/material/1/.

[Kop11] Hermann Kopetz. *Real-Time Systems. Design Principles for Distributed Embedded Applications*. 2nd ed. Springer, 2011.

[Kre05] K. Krewell. 'IBM speeds Xbox 360 to market'. In: *Microprocessor Report* (2005).

[Lyn16] *LynxSECURE. Software security driven by an embedded hypervisor*. Product Datasheet. Lynx Software Technologies, Inc. San Jose, CA, USA, 2016. URL: http://www.lynx.com/pdf/LynxSecureDatasheetFinal.pdf (visited on 09/06/2016).

[Mas+09] M. Masmano et al. 'XtratuM: a Hypervisor for Safety Critical Embedded Systems'. In: *11th Real-Time Linux Workshop*. (Dresden, Germany). 2009.

[Mas+10a] Miguel Masmano et al. 'XtratuM for LEON3: an Open Source Hypervisor for High Integrity Systems'. In: *5th Intl. Conf. and Exhibition on Embedded Realtime Software and Systems, ERTSS-2010*. (Toulouse, France). Ed. by Gérard Ladier and Jean-Luc Maté. 19th–21st May 2010. URL: www.fentiss.com/documents/xtratum-leon3.pdf.

[Mas+10b] M. Masmano et al. 'LithOS: a ARINC-653 guest operating [sic!] for XtratuM'. In: *12th Real-Time Linux Workshop*. (Nairobi, Kenia). 27th Oct. 2010. URL: http://www.xtratum.org/files/lithos-2010.pdf.

[Mas+11] Miguel Masmano et al. *XtratuM Hypervisor for LEON3. Volume 4: Reference Manual*. Polytechnical University of Valencia, Spain. Mar. 2011. URL: http://www.xtratum.org/files/xm-3-reference-023d.pdf.

[MCC12] Miguel Masmano, Alfons Crespo and Javier Coronel. *XtratuM Hypervisor for LEON4. Volume 2: XtratuM User Manual*. Polytechnical University of Valencia, Spain. Nov. 2012. URL: http://microelectronics.esa.int/ngmp/xm-4-usermanual-047d.pdf (visited on 12/01/2016).

[OH05] Kunle Olukotun and Lance Hammond. 'The future of microprocessors'. In: *ACM Queue* 3 (7 Sept. 2005), pp. 26–29.

[Ott07] Aliki Ott. 'System Testing in the Avionics Domain'. PhD thesis. Univ. of Bremen, Germany, Oct. 2007. urn:nbn:de:gbv:46-diss000108814.

[Pei+10] S. Peiró et al. 'Partitioned Embedded Architecture based on Hypervisor: the XtratuM approach'. In: *8th European Dependable Computing Conference, EDCC-8 2010*. (Valencia, Spain). IEEE Computer Society. 28th–30th Apr. 2010.

[PK15] Paul Parkinson and Larry Kinnan. *Safety-Critical Software Development for Integrated Modular Avionics*. Rev. 10/2015. White Paper. Wind River Systems, Inc. Oct. 2015. URL: http://www.windriver.com/whitepapers/safety-critical-software-development-for-integrated-modular-avionics/wp-safety-critical-software-development-for-integrated-modular-avionics.pdf (visited on 26/05/2016).

[RF07a] José Rufino and Sérgio Filipe. *AIR Project Final Report*. Tech. rep. DI-FCUL TR–07–35. Comp. Sce. Dept. of the University of Lisbon, Portugal, Dec. 2007. URL: http://air.di.fc.ul.pt/air/downloads/07-35.pdf.

[RF07b] José Rufino and Sérgio Filipe. *AIR Project Summary Report*. Tech. rep. DI-FCUL TR–07–36. Comp. Sce. Dept. of the University of Lisbon, Portugal, Dec. 2007. URL: air.di.fc.ul.pt/air/downloads/07-36.pdf.

[Rip+10] I. Ripoll et al. 'Configuration and Scheduling Tools for TSP Systems Based on XtratuM'. In: *Proc. of DASIA 2010 Data Systems in Aerospace*. (Budapest, Hungary). 1st–4th June 2010. URL: http://www.xtratum.org/files/dasia2010.pdf.

[Ros11] Joaquim Rosa. 'Development and Update of Aerospace Applications in Partitioned Architectures'. MSc. thesis. University of Lisbon, Portugal, 2011. URL: http://air.di.fc.ul.pt/air-ii/downloads/JRosa_Msc.pdf.

[RTC01] RTCA, Inc. *Final Annual Report For Clarification Of DO-178B 'Software Considerations in Airborne Systems and Equipment Certification'. RTCA/DO-248B*. Oct. 2001.

[RTC92] RTCA, Inc. *Software Considerations in Airborne Systems and Equipment Certification. RTCA/DO-178B*. Washington, D.C., USA, 1st Dec. 1992.

[RTE16a] *RTEMS Applications (on the homepage of the RTEMS project)*. 2016. URL: https://devel.rtems.org/wiki/TBR/UserApp/RTEMSApplications (visited on 12/05/2016).

[RTE16b] *RTEMS C User's Guide*. Version RTEMS 4.10.2. 2016. URL: https://docs.rtems.org/releases/rtemsdocs-4.10.2/share/rtems/html/c_user/index.html (visited on 28/01/2016).

[RTE16c] *RTEMS News (on the homepage of the RTEMS project)*. 2016. URL: https://www.rtems.org/ (visited on 01/02/2016).

[Rus81] John Rushby. 'The Design and Verification of Secure Systems'. Reprint of a paper presented at the 8th ACM Symposium on Operating System Principles, Pacific Grove, CA, USA, 14–16 Dec. 1981. In: *ACM Operating Systems Review* 15.5 (1981), pp. 12–21.

[San+08] Sérgio Santos et al. 'A Portable ARINC 653 Standard Interface'. In: *27th Digital Avionics Systems Conference, DASC 2008*. (St. Paul, MN, USA). Ed. by John Moore. 26th–30th Oct. 2008, 1.E.2-1–1.E.2-7. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4702767.

[SCA13] *SCARLETT project website*. SCARLETT Consortium. 2013. URL: http://www.scarlettproject.eu/ (visited on 17/11/2015).

[Sch11] Tobias Schoofs. *AIR – Overview*. Presentation. Lisbon, Portugal: GMV, 2011. URL: http://www.gmv.com/export/sites/gmv/DocumentosPDF/air/Presentation_GMV-AIR-1.1.pdf.

[Sch14] Martin Schoeberl. 'T-CREST: Time-predictable Multi-Core Architecture for Embedded Systems'. In: *2nd Int'l workshop on the Integration of mixed-criticality subsystems on multi-core and many-core processors at HiPEAC Conference 2014*. (Vienna, Austria). 20th–22nd Jan. 2014. URL: http://138.4.11.199:8080/multipartes/public/HiPEAC/T-CREST-20140121-MCS.pdf (visited on 09/06/2016).

[SCS12] Cláudio Silva, João Cristóvão and Tobias Schoofs. 'An I/O Building Block for the IMA Space Reference Architecture'. In: *Proc. of DASIA 2012 Data Systems In Aerospace*. (Dubrovnic, Croatia, 14th–16th May 2012). Ed. by L. Ouwehand. SP-701. ESA Spacebooks Online, Aug. 2012. URL: http://www.gmv.com/export/sites/gmv/DocumentosPDF/air/Paper_DASIA_2012.pdf.

[Sil+10] Helder Silva et al. 'RTEMS Improvement – Space Qualification of RTEMS Executive'. In: *INForum 2009 – I Simpósio de Informática*. (Lisbon, Portugal, 10th–11th Sept. 2010). Sept. 2010. URL: http://air.di.fc.ul.pt/air-ii/downloads/Silva09inforum.pdf.

[Ste+08] Dave Steinberg et al. *EMF: Eclipse Modeling Framework*. 2nd ed. Eclipse Series. Addison-Wesley Professional, 16th Dec. 2008.

[SYS16] *PikeOS. Hypervisor*. Review 5/2016. Product Datasheet. SYSGO. Klein-Winternheim, Germany, May 2016. URL: https://www.sysgo.com/fileadmin/user_upload/www.sysgo.com/redaktion/downloads/pdf/data-sheets/SYSGO-Datasheet-PikeOS-4.0.pdf (visited on 19/05/2016).

[TCA13] Salvador Trujillo, Alfons Crespo and Alejandro Alonso. 'MultiPARTES: Multicore for Mixed-criticality virtualization Systems'. In: *16th Euromicro Conference on Digital System Design*. (Santander, Spain). 4th–6th Sept. 2013. URL: http://www.dit.upm.es/~str/papers/pdf/trujillo&13a.pdf (visited on 09/06/2016).

[WDD11] James Windsor, Marie-Hélène Deredempt and Regis De-Ferluc. 'Integrated Modular Avionics for Spacecraft – User Requirements, Architecture and Role Definition'. In: *30th Digital Avionics Systems Conference, DASC 2011*. (Seattle, WA, USA). IEEE/AIAA. 16th–20th Oct. 2011, 8A6-1–8A6-16.

[WH09] James Windsor and K. Hjortnaes. 'Time and Space Partitioning in Spacecraft Avionics'. In: *Third IEEE Int'l Conf. on Space Mission Challenges for Information Technology, SMC-IT 2009*. (Pasadena, CA, USA). 19th–23rd July 2009, pp. 13–20. DOI: 10.1109/SMC-IT.2009.11.

[Wil+08] R. Wilhelm et al. 'The Worst-Case Execution Time Problem – Overview of Methods and Survey of Tools'. In: *ACM Trans. on Embedded Computer Systems* 7 (3 2008), pp. 1–53.

[Wil09] Reinhard Wilhelm. 'Determining Bounds on Execution Times'. In: *Embedded systems handbook. Embedded systems design and verification*. Ed. by Richard Zurawski. 2nd ed. Boca Raton, FL, USA: CRC Press, 2009.

[Win12] James Windsor. 'Integrated Modular Avionics for Space. IMA4Space'. In: *6th ESA Workshop on Avionics, Data, Control and Software Systems, ADCSS 2012*. 23rd Oct. 2012. URL: http://congrexprojects.com/docs/12c25_2310/sa1440_deredept.pdf?sfvrsn=2 (visited on 10/12/2015).

[WRC14] Garth Watney, Leonard J. Reder and Timothy Canham. 'Modeling for Partitioned and Multi-core Flight Software Systems (Instrument Software Framework)'. In: *5th IEEE International Conference on Space Mission Challenges for Information Technology, SMC-IT 2014*. (Laurel, Maryland, USA). 24th–26th Sept. 2014, pp. 54–61. DOI: 10.1109/SMC-IT.2014.15.