# Avionics Applications on a Time-predictable Chip-Multiprocessor

André Rocha and Cláudio Silva
*GMV*
*Lisbon, Portugal*
*Email: [andre.rocha, claudio.silva]@gmv.com*

Rasmus Bo Sørensen, Jens Sparsø, and Martin Schoeberl
*Department of Applied Mathematics and Computer Science*
*Technical University of Denmark*
*Email: [rboso, jspa, masca]@dtu.dk*

***Abstract*—Avionics applications need to be certified for the highest criticality standard. This certification includes schedulability analysis and worst-case execution time (WCET) analysis. WCET analysis is only possible when the software is written to be WCET analyzable and when the platform is time-predictable. In this paper we present prototype avionics applications that have been ported to the time-predictable T-CREST platform. The applications are WCET analyzable, and T-CREST is supported by the aiT WCET analyzer. This combination allows us to provide WCET bounds of avionic tasks, even when executing on a multicore processor.**

## I. Introduction

The mission of the T-CREST project [1], [2] is to develop and build a multicore processor that is time-predictable and easy to analyze for the worst-case execution time (WCET). Furthermore, the T-CREST platform increases the performance with multicore technology. Consequently, we expect that the T-CREST platform results in lower costs for safety-relevant applications, reducing system complexity, and ensuring faster and time-predictable execution. A processor designed for real-time systems and safety-critical systems is radically different from one optimized for average-case performance. For certification only the worst-case performance is important, not the average-case performance.

The T-CREST project builds the time-predictable multicore platform from scratch, encompassing the following five technology elements: (1) a new processor named Patmos [3], (2) the network-on-chip (NoC) infrastructure supporting time-predictable communication between processors [4], [5] and a specific memory NoC for shared main memory access [6], [7], (3) a time-predictable memory controller [8], (4) a compiler infrastructure for the Patmos processor [9], and (5) the aiT WCET analysis tool [10] adapted to support Pamtos and to guide the compiler optimizations for the WCET. The overall purpose of the T-CREST platform is building time-predictable computer architecture [11].

This paper demonstrates the platform's ability to host real-time applications with delicate predictability requirements. We use avionics use cases where time-predictability is of primary importance. More challenging use cases combine two or more applications. We use the term "demonstrators" to refer to the individual applications and the combination of two or more applications.

The demonstrators were ported to the T-CREST platform using its compiler tool-chain and analysis tools. The demonstrators validate the T-CREST platform. Part of the exercise, however, is to evaluate the added value of the platform. The platform shall enable application developers to determine the WCET of their applications more precisely or more easily. Therefore, we compare the T-CREST platform with a well-established platform in the avionics domain.

The T-CREST platform was evaluated with the aid of the following three real-world avionic applications: (1) an Airlines Operational Centre (AOC), (2) a Crew Alerting System (CAS), and (3) an I/O Partition (IOP). We chose T-CREST as platform for avionics applications as this is currently the only multicore processor where static WCET analysis of C programs is possible.

This paper is organized in 7 sections: Section II introduces the T-CREST platform used in the evaluation. Section III gives the background on Integrated Modular Avionics and presents the avionic demonstrators. Section IV describes part of the software stack that has been adapted or developed for the T-CREST platform. Section V is the core part of this paper, presenting and discussing the evaluation results. Section VI presents related work on time-predictable multicore processors. Section VII concludes.

## II. The T-CREST Multicore Processor

The T-CREST project builds the time-predictable multicore platform from scratch. The main design objective of T-CREST is its time-predictability—predictable timing is a first-order design factor [12]. It is developed according to the following design principle [11]:

> Make the worst-case fast and the whole system easy to analyze.

All components of a platform, including processor, network-on-chip, memory hierarchies, and the compiler, need to be designed for time-predictability for future, safety-critical avionics applications.

Designing and optimizing a multicore processor for real-time systems results in a radically different design than that of a standard processor. For example, communication between cores via shared memory, and then implicitly via a cache coherence protocol is not analyzable for the WCET.

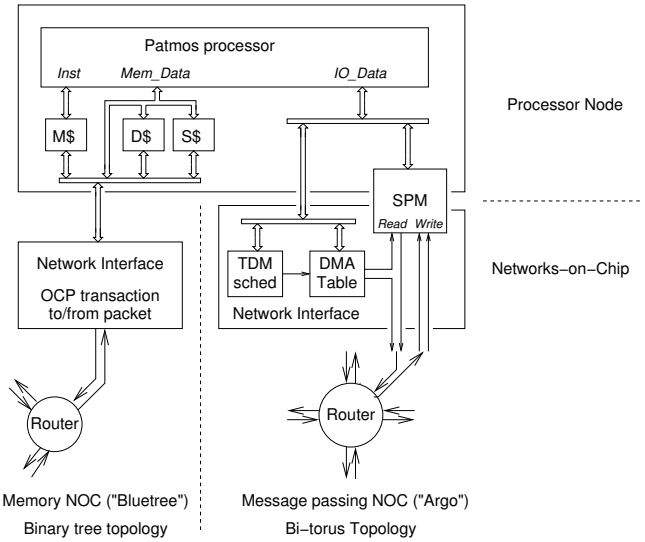


Figure 1. A Patmos processor node and its interfaces to the memory-tree NoC and the Argo message passing NoC.

Therefore, we support explicit message passing between cores via a NoC.

Figure 1 shows a Patmos processor node and its interfaces towards both the Bluetree memory NoC and the Argo message-passing NoC. The memory tree NoC provides access to a shared off-chip memory and is used exclusively for booting the platform and servicing the (non-coherent) cache memories. The Argo message-passing NoC provides inter-processor communication and is extensively used by the applications described in this paper.

A new processor named Patmos has been developed within T-CREST. Patmos is a 32-bit, RISC-style dual-issue VLIW processor core [3]. To support the single-path programming paradigm [13], Patmos supports predicates on all instructions.

For time-predictable caches, the processor supports several variants of caches and on-chip memories: a method cache [14] handles the caching of complete methods/functions. Moreover, stack allocated data can be cached in the special stack cache [15]; other data is cached in the data cache. The method cache is supported by a scope-based WCET analysis [16]; the stack cache, by a intra-procedural data-flow analysis [17]. Functions and data can also be placed into the instruction and data scratchpad memories (SPM). Another SPM serves as a communication SPM with the Argo NoC.

Argo is a packet-switched source-routed NoC that implements virtual channels using statically scheduled time-division multiplexing (TDM) of the resources (i.e., links) in the NoC [4]. In this way, once a packet is injected into the network of routers, it traverses the end-to-end path (i.e., the virtual channel) in a pipelined fashion without ever competing with other packets for access to resources in the

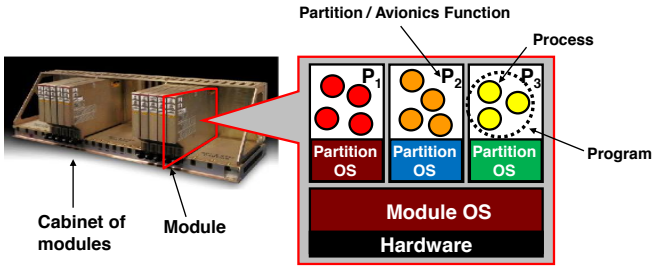


Figure 2. IMA terminology

NoC. A physical DMA controller is shared by all outgoing virtual channel from a processor node. As messages are usually larger than the payload of a single packet, messages are typically sent from a processor node in an interleaved fashion, following the TDM schedule.

The hardware implementation of Argo has two novel features that distinguish it from other NoCs with similar functionality. First, the DMA controllers are moved into the network interfaces and tightly integrated with the TDM scheduling [18], as illustrated in Figure 1. Second, the Argo NoC use asynchronous routers [5], [19]. The result is a hardware-efficient implementation that at the same time enables a globally asynchronous, locally synchronous timing organization of the entire multiprocessor platform.

All processor cores are connected via a memory tree called “Bluetree” [20]. Bluetree provides access to a shared external SDRAM memory controlled by a real-time memory controller [8], [21]. An alternative memory NoC implements distributed TDM arbitration [7].

The compiler infrastructure is an essential part of the system developed in the project [9]. WCET-aware optimization methods [22] were developed, along with detailed timing models, so that the compiler benefits from the known behavior of the hardware. The T-CREST partner AbsInt adapted the aiT WCET analysis tool [10] to support the Patmos processor. Furthermore, aiT provides information to the compiler that can then optimize along the WCET path.

## III. Avionics Applications

This section first introduces the Integrated Modular Avionics (IMA) concept and then presents the avionics demonstrators based on IMA.

### A. IMA and ARINC 653

The goal of IMA (shown in Figure 2) is to reduce hardware and thereby weight and power consumption by sharing resources among applications. IMA enables multiple unrelated applications, with different criticality levels, to share the same computing platform *without interference*.

Partitioning separates applications in two dimensions: space and time. Spatial separation means that the memory of a partition is protected. Temporal separation means that only one application at a time has access to system resources,

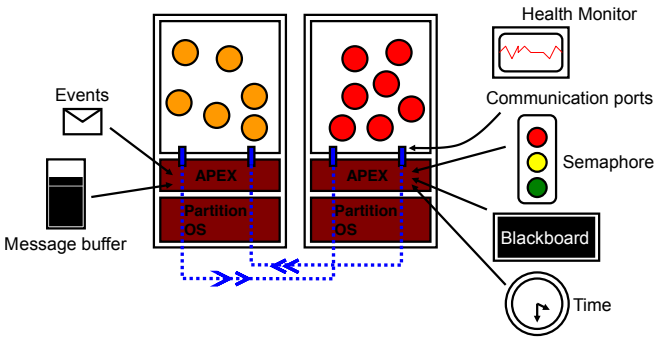


Figure 3. ARINC 653 API

including the processor; therefore, only one application is executing at any time—with no competition for system resources between partitioned applications. Furthermore, processors are statically assigned to partitions. A processor cannot change its partition at runtime.

ARINC 653 is an IMA-compliant specification that defines an application-programming interface (API) called "APEX" [23]. APEX provides applications with a common interface for accessing system services and specifying the distribution of time and memory among the partitions. As illustrated in Figure 3, APEX provides applications with a set of services:

- inter-process communication within a partition (events, message buffers, black boards, semaphores, etc.)
- time services
- an interface to the health monitor
- interfaces to the partition associated with the application and the processes it consists of

Communication among partitions occurs through ports defined at the partition level. A port is either incoming or outgoing and may be either a queueing port, holding zero or more messages, or a sampling port containing the current instance of a periodically updated message. Ports are linked by channels, which are transparent to applications.

### B. The Airlines Operational Centre Use Case

The Airlines Operational Centre (AOC) is the on-board part of an air traffic management system that enables digital text communication between the aircrew and ground units. It was developed according to DO-178B Design Assurance Level C and is thus a moderately critical application. The AOC is a communication router and message database. It stores reports sent from ground stations or created by aircraft subsystems.

Typical reports sent from the ground *to the aircraft* are weather reports, messages advising the aircraft about unforeseen events, constraint lists for aircraft trajectory planning, flight plans, in-flight traffic management messages, and free text messages. Typical reports sent *from the aircraft* to the ground are information on the main flight events (out of parking position, off the runway, on the runway, in the


Figure 4. AOC MCDU display

parking position), the expected time of arrival (updated and sent periodically), reports on engine and fuel state, reports on observed weather conditions, aircraft trajectory plans, and free text messages.

The AOC solution contains a user interface for the pilot, the Multifunction Control Display Unit (MCDU). Figure 4 shows the MCDU with the main menu. This interface is used for displaying received reports, changing report settings such as contracts, requesting reports from the ground and creating reports such as free text messages or malfunction reports. The MCDU also presents engine data such as engine and oil temperature, fuel flow, and engine RPM (Revolutions Per Minute).

The actual AOC application provides the link between the ground unit and the pilot's MCDU. This means that the AOC is responsible for storing all the messages exchanged between those interfaces, receiving new messages as they are transmitted, and dispatching the proper data whenever the pilot switches its interface to a new menu or new data needs to be displayed in the current MCDU menu. The AOC is an IMA application, compliant with ARINC 653, and therefore running on all ARINC 653-compliant operating systems.

### C. The Crew Alert System Use Case

Modern aircrafts have a Crew Alert System (CAS) to aid the aircrew. The CAS receives signals from on-board subsystems, such as doors, engines, or the environment control system, and displays relevant aircraft information such as engine parameters (e.g., temperature values, fuel flow, and quantity). The CAS improves situational awareness by allowing the aircrew to view complex information in a graphical format and by alerting the crew to unusual or hazardous situations.

The CAS prototype follows the DO-178B guidance for Design Assurance Level A, the highest certification level.

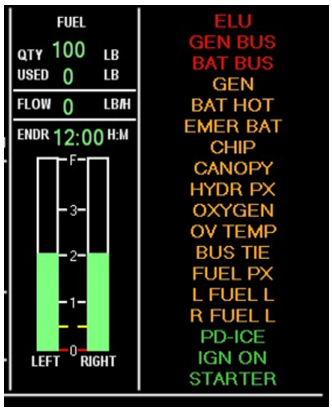


Figure 5. CAS display unit

The CAS comprises two elements: the display unit and the processing unit. The display unit, shown in Figure 5, is responsible for presenting information to the aircrew. The processing unit receives input signals from selected aircraft subsystems, translates this information if needed, and manages the messages to be presented to the aircrew in accordance with specified requirements. The CAS processing unit input signals are simulated in the demonstrator.

The processing unit must assign each received signal to one of the following message-criticality groups: *Warning* messages in red, *Caution* messages in amber, and *Advisory* messages in green.

The ranking of a message in the displayed list of messages, an example shown in Figure 5, is determined by its time of activation, acknowledgement status, and criticality group. Warning and Caution messages are initially presented in flashing mode, until the pilot acknowledges them. Upon acknowledgement, all messages of that criticality group lose their highlight status and return to normal mode. Advisory messages do not need acknowledgement.

### D. The I/O Partition

General-purpose operating systems hide the custom code that controls I/O devices and provide applications with clean high-level interfaces to the I/O devices. The same approach cannot be applied to a robust partitioned operating system, as doing so may violate spatial partitioning. If I/O devices are shared across partitions, there is a risk of the devices being left in an unknown state following a partition error [24]. Consequently, the independence and isolation of partitions could be compromised.

We solve this problem by introducing a central generic I/O module executing in its own partition to route data to and from applications in several partitions. Figure 6 shows this configuration.

The I/O partition (IOP), being subject to time and space partitioning, does not forward the data directly. Instead, communication is asynchronous by design. In its execution

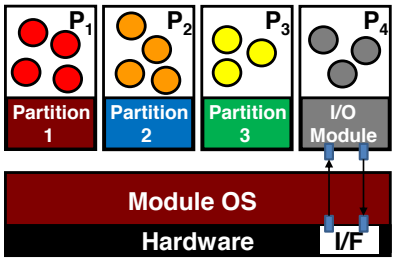


Figure 6. System configuration with an I/O module implemented by a dedicated partition

window, an application sends its data to the IOP. The IOP will handle this data only during its next execution window. On the other side of the channel, the same process may occur. The IOP receives the data and forwards it to the final destination. But the targeted partition will receive this data only during its next execution window. Execution windows of IOPs thus extend the path through the network. The IOP currently implements (1) drivers and protocol stacks for RS-232, SpaceWire and MIL-STD-1553B including Bus Controller and Remote Terminal modes, and (2) a deterministic IP stack providing UDP on top of an Ethernet driver.

From an application's perspective, there are three ways of accessing the IOP services: through the middleware API, through remote ports, or through shared memory. For most cases, the underlying communication mechanism will be a set of queuing ports [23]. This set of queuing ports is used for sending requests to the I/O device and sending results back to the requesting partition. In cases involving the exchange of huge amount of data, an additional interface (such as shared memory) will be used.

Internally, the IOP consists of a set of tasks that are periodically executed according to the partition frequency. The scheduler executes the internal schedule, i.e., the sequence of task execution according to their periods and priorities, to always achieve a full read-and-write cycle per partition execution window.

## IV. The Software Stack

Along with the hardware development and the porting of the avionics applications, a software stack was ported or developed. The software stack consists of: (1) the RTEMS real-time operating system, (2) ARINC 653 functions, and (3) a message-passing library for the NoC.

### A. RTEMS Port

A major effort involved in porting the avionics demonstrators to the T-CREST platform was porting a real-time operating system to the new platform. The Patmos CPU was added to the collection of supported architectures of RTEMS 4.10.2.

Porting RTEMS to Patmos entailed developing a set of RTEMS-internal functions: platform-specific initialization, context management (context initialization and context

switching), clock management, and support for a serial port. In addition, a message-passing library for the NoC was added to RTEMS and is available to application developers.

Porting RTEMS had a positive side effect. Because it is a complex piece of software, it covers a wider range of features from the processor and associated tool-chain than would a simple application. In a way, RTEMS acted as a use case for the platform in its own right. Building and running RTEMS ended up assessing the maturity of the T-CREST platform.

### B. ARINC 653 Functions

Both avionics applications (AOC and CAS) rely on ARINC 653 functions to operate. They use processes to run modular sequences of instructions, force process switching through timed waits, adopt buffers to store and transmit large amounts of data, and employ sampling and queuing ports to communicate with other applications.

We use RTEMS tasks, message queues, and timer functions for implementing ARINC processes, buffers, and timers. The mapping between RTEMS and ARINC functions is straightforward, as each ARINC function always has a corresponding RTEMS counterpart. For the bare applications (without RTEMS), buffers have been implemented, whereas the existing process structure was erased. Instead of running different processes, the applications consist of one single line of execution, running the different modules in an endless loop, switching periodically between the modules.

The implementation of sampling and queuing ports is the same for the RTEMS and the bare-bone configuration. It relies on platform-specific features, namely, the communication scratchpad memory for the NoC local to each core. In our implementation, sampling and queuing messages lie in each core's local communication scratchpad memory, as well as auxiliary information for the messages. These messages are then transmitted via the NoC to all participating cores, thereby avoiding the expensive access to shared memory. In addition, this implementation imposes a static configuration of ports, i.e., there must be a set of configuration files declaring and defining the characteristics of the ports to be created during the execution of the use case.

### C. Message-passing Library

Porting the avionics applications to the T-CREST platform, we consider executing single partitions on separate cores of the platform. For executing the individual partitions on separate cores, we need to map the inter-partition communication onto the NoC. Therefore, we avoid copying messages between main memories of the individual partitions.

ARINC 653 specifies sample-based communication and queuing-based communication for inter-partition communication. The original ARINC interface for queuing and sampling ports specify that data be copied in and out of buffers internally in the library, when the data is sent and received, respectively.

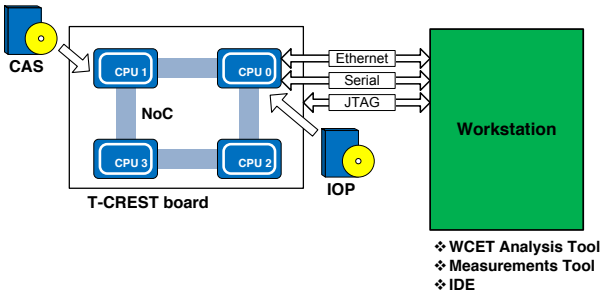


Figure 7. Example configuration of an avionic demonstrator

However, as this copying is not optimal for the T-CREST platform, we developed a zero-copying message-passing library [25]. The messages are created in local memory and sent automatically from there via the NoC. The message is not copied before transmission; therefore the current sending buffer can be reused only when the sending has finished. The same approach is used on the receiver side. A received message is not copied out of the local memory but processed in place. When the processing of the message is completed, the buffer for it is marked as free for re-use.

To enable overlapping of computation and communication, the library supports double buffering on the sender side and a queue of message buffers on the receiver side. The message-passing library is written in a WCET-analyzable way. As we avoid the copying of messages, the WCET of the individual library functions are independent from the message size.

## V. Evaluation

All demonstrators were originally IMA applications that communicated with other applications and external systems through queuing and sampling ports. These communication interfaces are originally based on buffers in main memory and, thus, are subject to heavy contention in a multicore processor. As part of the optimization to the T-CREST platform, the port interfaces used by the demonstrators were mapped to inter-core communication using the configurable Argo NoC. This optimization removes part of the contention for the main memory and therefore reduces the WCET of the demonstrators.

### A. Evaluation Configuration

An example configuration is shown in Figure 7. Both CAS and IOP are hosted on the T-CREST board, and a workstation is used to interface with the board and the demonstrators via both a serial and an Ethernet connection. The IOP demonstrator is deployed on core 0, so that it can access the Ethernet port, and the CAS application is loaded to core 1.

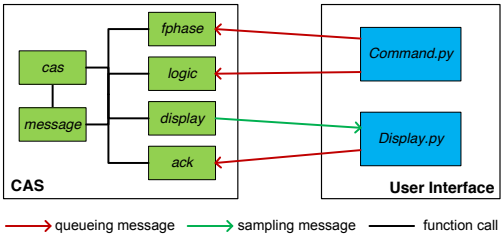


Figure 8. CAS original interfaces

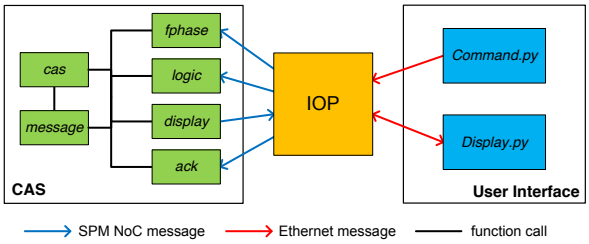


Figure 9. CAS interfaces in T-CREST

The IOP demonstrator was originally developed to manage I/O access in partitioned systems, where various partitions need to communicate with I/O devices in a safe, controlled manner. In all the avionics demonstrators, IOP's main duty is to manage the Ethernet interface. Applications trying to access the network will send data to the core, where the IOP is running using the Argo NoC. This communication is abstracted by the ARINC 653 queuing and sampling port interface. Once the data has been received, the IOP routes it according to its internal routing tables and sends it to the Ethernet network and data received by the IOP from the network is routed and sent to the applications via the NoC.

The CAS demonstrator is composed of the display unit and the processing unit. For the user to visualize messages, the display unit corresponds to a graphical interface. The display unit is simulated on the workstation.

The core of the CAS application is the processing part that is executed on the T-CREST platform. It can be decomposed in 6 subcomponents: (1) management of CAS messages, (2) interaction with the display, (3) setting of the flight phase, (4) message activation logic, (5) message acknowledgement, and (6) CAS main function for periodically calling the other CAS components.

At the workstation, the user can send one of two commands to the CAS processing element: (1) change of the flight phase and (2) activation of a specific message.

In the original application, sampling and queuing ports were used for communicating between the processing unit and the display unit (see Figure 8). In the T-CREST system, communication between the processing unit and the display unit was replaced by the Ethernet connection between the platform and the user workstation (see Figure 9). The arrows in these figures show the direction of the message exchange.

The IOP demonstrator serves as the middleman for CAS to communicate with its graphical interface at the user workstation (see Figure 9). In this case, the initial sampling and queuing ports are mapped to NoC-based ports that associate the various CAS processing components to IOP.

### B. Evaluation Results

The main objective of the avionics evaluation was to demonstrate that, given a configuration of the T-CREST platform, independently obtaining the WCET of any application is possible, regardless of other software executing on the platform. This temporal independence between applications is a cornerstone in the IMA concept, which is difficult to obtain in current multicore systems. The lack of analyzable multicore platforms hampers their adoption in the aerospace market, preventing the potential benefits in terms of higher system integration that can lead to more cost-efficient avionics systems.

To validate application independence, a demonstrator was set up in which each core hosts a different application (AOC, IOP, CAS) that would, in a typical IMA system, be a stand-alone partition. This asymmetric distribution of applications on cores was used to show that the timing of each application depends solely on its own execution and the hardware configuration.

To provide further insight over the behavior of the T-CREST platform, several test cases were setup by varying the number, configuration, and distribution of the avionics applications. For the different test cases, we estimated the WCET of selected tasks and, in some cases, compared the WCET against a measurement of the average-case execution time. We obtain the measured execution time by reading the cycle-accurate timer of Patmos at specific points in the source code.

The following tables show the WCET results of individual tasks of the avionics demonstrators. The WCETs are obtained for variations of the hardware (4 or 9 cores) and variations of the application setup. The hardware configuration has an impact on the WCET because more cores access the shared main memory. The application setup shows almost no influence on the WCET of tasks.

As Tables I and II show, we were able to obtain WCET estimations for every avionics application. Table I shows that the main factor influencing the WCET estimation is the number of cores available in the platform. In contrast, variations in concurrently executing applications, here appearing as different test cases, have no noticeable impact on the WCET estimation. The WCET estimation for the `cas_loop` entry point displays a twofold increase when changed from a quad-core platform to a nine-core platform.

Some applications, listed in Table I, exhibit a very small variance in the WCET estimation for different application setups while using the same number of cores. This small variation is due to different configurations used in different

Table I
WCET RESULTS FOR SELECTED TASKS OF AVIONICS DEMONSTRATORS

| Analysis entry | Test case | Cores | WCET (in ms) |
|---|---|---|---|
| `cas_loop` | CAS+IOP | 4 | 284 |
| | AOC+CAS+IOP | 9 | 619 |
| `AGPAOCReceiveMainLoop` | AOC+IOP | 9 | 5.41 |
| | AOC+CAS+IOP | 9 | 5.45 |
| `AirplanePAOCMainTask` | AOC+IOP | 9 | 1.92 |
| | AOC+CAS+IOP | 9 | 1.94 |
| `decoderLoop` | AOC+IOP | 9 | 210 |
| | AOC+CAS+IOP | 9 | 210 |
| `AOCAlertMainLoop` | AOC+IOP | 9 | 2.72 |
| | AOC+CAS+IOP | 9 | 2.74 |

Table II
WCET ESTIMATIONS AND AVERAGE-CASE EXECUTION TIME MEASUREMENTS FOR IOP: PRE_DISPATCHER AND POS_ROUTER ENTRY POINTS

| Analysis entry | Test case | Cores | WCET est. (in ms) | Timing meas. (in ms) |
|---|---|---|---|---|
| `pre_dispatcher` | CAS+IOP | 4 | 6.24 | 0.952 |
| | AOC+IOP | 9 | 14.64 | 0.239 |
| | AOC+CAS+IOP | 9 | 23.57 | 1.110 |
| `pre_router` | CAS+IOP | 4 | 6.42 | 0.121 |
| | AOC+IOP | 9 | 15.35 | 0.120 |
| | AOC+CAS+IOP | 9 | 27.07 | 0.151 |

test cases. Some of these configurations (e.g., the number of ARINC 653 ports) have a small impact over loop bounds and, hence, over the estimated WCET. This effect is also present on the WCET estimation results from the IOP application presented in Table II.

All WCET estimations are, as expected, higher than the average case measurements. However, as average case measurements usually do not trigger the worst-case execution path, the big difference between WCET estimates and measurements has no real meaning, Triggering the worst-case execution path is practically impossible for non-trivial application, as this path is usually unknown, providing another argument for static WCET analysis of tasks in safety-critical applications.

### C. Comparison with LEON

LEON processors [26] are very common in real-time systems, especially in the space domain. LEON is also supported by the aiT WCET analysis tool. We selected the IOP application, as an example, to compare the T-CREST platform against a LEON 3 processor.

Alongside the compiled IOP executable for LEON 3, a manually composed AIS annotations file is used as input to AbsInt's WCET analysis tool for obtaining the estimated WCET values for the LEON processor. These values are then compared with the values obtained, using the same source code compiled for the T-CREST platform. However, this comparison is feasible only for a single core, as determining the WCET for multicore configurations of the LEON processor are impossible (the problem is unbounded).

Table III
COMPARISON OF WCET RESULTS BETWEEN PATMOS AND LEON FOR THE IOP APPLICATION

| Analysis entry | Cores | Target CPU | WCET (in ms) |
|---|---|---|---|
| `pre_dispatcher` | 1 | Patmos | 2.20 |
| | | LEON | 2.32 |
| | 4 | Patmos | 5.75 |
| | | LEON | 45.28 |
| `pre_router` | 1 | Patmos | 2.21 |
| | | LEON | 2.15 |
| | 4 | Patmos | 6.06 |
| | | LEON | 41.81 |

Nonetheless, an approximate value of the expectable WCET, for multicore LEON, is estimated by factoring single-core WCET values with a maximum interference multiplier representative of a LEON multicore processor. This maximum interference multiplier is extracted from the literature [27], namely a European Space Agency-funded study aimed at characterizing the NGMP processor (quad-core LEON 4). This study found that inter-core interference could increase the execution time of a given code segment up to twenty times compared to its single-core value.

Table III presents the comparison between LEON 3 and T-CREST/Patmos. From the comparison in Table III we can conclude that, in single-core configurations, Patmos and LEON have similar results with one alternately exhibiting a marginal reduction ($<5$ %) in the WCET bound over the other, depending on the specific task analyzed.

In multicore configurations, the difference is more significant; Patmos, as part of the T-CREST platform, can be directly targeted by static WCET analysis techniques. Such analysis is unfeasible in multicore versions of the LEON processor, such as the NGMP. Being analyzable in terms of WCET behavior offers the T-CREST platform a key advantage over the LEON multicore processor. When comparing the WCET values obtained in multicore T-CREST with those empirically estimated for the LEON, we see the T-CREST platform yielding a seven times lower WCET bound. Nonetheless, we cannot use the LEON values presented to build a safety case around the software, because they are rough estimates derived from empirically obtained interference patterns.

## VI. RELATED WORK

As standard processors do not fit for future avionics applications, research on time-predictable architectures is gaining momentum. This section compares related research with the T-CREST platform.

A bus-based multicore processor has been developed within the MERASA (Multi-Core Execution of Hard Real-Time Applications Supporting Analysability) [28] project. In contrast to MERASA, we developed a network-on-chip based multicore architecture. We also tackled the time predictability challenge by developing a new processor architecture, with a WCET optimized instruction set, and the supporting compiler.

The FP-7 project PREDATOR studied the factors that influence time-predictability in processors [29]. Within the T-CREST project we followed the design principles of the PREDATOR project.

The precision-timed (PRET) machine [30], [31] is a processor that supports repeatable timing. The initial version of PRET implements one version of the ARM instruction set [32], [33]. PRET supports several threads in execution via chip-multithreading. Scratchpad memories are used instead of instruction and data caches. FlexPRET [34] extends PRET to support two different thread types, hard and soft real-time threads, directly in the hardware. This extension is intended for mixed-critically systems. The main difference between our proposal and PRET is that we focus on time predictability [11], [12] and PRET on repeatable timing.

Fernandez et al. [35] explore the task interference in a LEON 4-based quad-core processor. The processor is a conventional multiprocessor that has been developed with critical real-time systems in mind. The study shows that interference may cause the execution time of a task to increase by a factor of 2-9 times when executing on the multicore configuration, compared to a single core configuration. This result is a strong argument in support of time-predictable multicore architectures such as the one we have built with the T-CREST platform.

## VII. Conclusion

Avionics applications need to be certified for the highest criticality standard. In this paper we presented avionics applications that have been ported to the time-predictable T-CREST multicore processor. We showed that the avionics applications can be analyzed for their worst-case execution time when executing on the T-CREST multicore processor, although multicore processors are usually a very hard target for worst-case execution time analysis. Therefore, we conclude that the T-CREST platform is a good choice for high criticality real-time systems.

## Acknowledgements

We would like to thank all T-CREST partners for their support, the successful building of the T-CREST platform, and successful execution of the EC funded T-CREST project.

This work was partially funded under the European Union's 7th Framework Programme under grant agreement no. 288008: Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST), and by the Danish Council for Independent Research | Technology and Production Sciences under the project RTEMP, contract no. 12-127600

*Source Access*

Most parts of the T-CREST implementation are available in open source: https://github.com/t-crest. Further information can be found at the project web site: http://www.t-crest.org and the web site for the processor and the compiler: http://patmos.compute.dtu.dk/

## References

[1] M. Schoeberl, C. Silva, and A. Rocha, "T-CREST: A time-predictable multi-core platform for aerospace applications," in *Proceedings of Data Systems In Aerospace (DASIA 2014)*, Warsaw, Poland, June 2014.

[2] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, A. Rocha, C. Silva, J. Sparsø, and A. Tocchi, "T-CREST: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 449–471, 2015.

[3] M. Schoeberl, P. Schleuniger, W. Puffitsch, F. Brandner, C. W. Probst, S. Karlsson, and T. Thorn, "Towards a time-predictable dual-issue microprocessor: The Patmos approach," in *First Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES 2011)*, Grenoble, France, March 2011, pp. 11–20.

[4] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki, "A statically scheduled time-division-multiplexed network-on-chip for real-time systems," in *Proceedings of the 6th International Symposium on Networks-on-Chip (NOCS)*. Lyngby, Denmark: IEEE, May 2012, pp. 152–160.

[5] E. Kasapaki, M. Schoeberl, R. B. Sørensen, C. T. Müller, K. Goossens, and J. Sparsø, "Argo: A real-time network-on-chip architecture with an efficient GALS implementation," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. PP, 2015.

[6] J. Garside and N. C. Audsley, "Prefetching across a shared memory tree within a network-on-chip architecture," in *System on Chip (SoC), 2013 International Symposium on*, Oct 2013, pp. 1–4.

[7] M. Schoeberl, D. V. Chong, W. Puffitsch, and J. Sparsø, "A time-predictable memory network-on-chip," in *Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, Madrid, Spain, July 2014, pp. 53–62.

[8] M. D. Gomony, B. Akesson, and K. Goossens, "Architecture and optimal configuration of a real-time multi-channel memory controller," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, 2013, pp. 1307–1312.

[9] P. Puschner, R. Kirner, B. Huber, and D. Prokesch, "Compiling for time predictability," in *Computer Safety, Reliability, and Security*, ser. Lecture Notes in Computer Science, F. Ortmeier and P. Daniel, Eds. Springer Berlin / Heidelberg, 2012, vol. 7613, pp. 382–391.

[10] R. Heckmann and C. Ferdinand, "Worst-case execution time prediction by static program analysis," AbsInt Angewandte Informatik GmbH, Tech. Rep., [Online, last accessed November 2013].

[11] M. Schoeberl, "Time-predictable computer architecture," *EURASIP Journal on Embedded Systems*, vol. vol. 2009, Article ID 758480, p. 17 pages, 2009.

[12] ——, "Is time predictability quantifiable?" in *International Conference on Embedded Computer Systems (SAMOS 2012)*. Samos, Greece: IEEE, July 2012.

[13] P. Puschner, "Experiments with WCET-oriented programming and the single-path architecture," in *Proc. 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, Feb. 2005.

[14] P. Degasperi, S. Hepp, W. Puffitsch, and M. Schoeberl, "A method cache for Patmos," in *Proceedings of the 17th IEEE Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC 2014)*. Reno, Nevada, USA: IEEE, June 2014, pp. 100–108.

[15] S. Abbaspour, F. Brandner, and M. Schoeberl, "A time-predictable stack cache," in *Proceedings of the 9th Workshop on Software Technologies for Embedded and Ubiquitous Systems*, 2013.

[16] B. Huber, S. Hepp, and M. Schoeberl, "Scope-based method cache analysis," in *Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, Madrid, Spain, July 2014, pp. 73–82.

[17] A. Jordan, F. Brandner, and M. Schoeberl, "Static analysis of worst-case stack cache behavior," in *Proceedings of the 21st International Conference on Real-Time Networks and Systems (RTNS 2013)*. New York, NY, USA: ACM, 2013, pp. 55–64.

[18] J. Sparsø, E. Kasapaki, and M. Schoeberl, "An area-efficient network interface for a TDM-based network-on-chip," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13. San Jose, CA, USA: EDA Consortium, 2013, pp. 1044–1047.

[19] E. Kasapaki and J. Sparsø, "Argo: A time-elastic time-division-multiplexed noc using asynchronous routers," in *Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, May 2014, pp. 45–52.

[20] J. Garside and N. C. Audsley, "Investigating shared memory tree prefetching within multimedia noc architectures," in *Memory Architecture and Organisation Workshop*, 2013.

[21] E. Lakis and M. Schoeberl, "An SDRAM controller for real-time systems," in *Proceedings of the 9th Workshop on Software Technologies for Embedded and Ubiquitous Systems*, 2013.

[22] F. Brandner, S. Hepp, and A. Jordan, "Criticality: static profiling for real-time programs," *Real-Time Systems*, pp. 1–34, 2013.

[23] *AVIONICS APPLICATION SOFT-WARE STANDARD INTERFACE PART 1: REQUIRED SERVICES*, 3rd ed., AERONAUTICAL RADIO, INC., 2551 RIVA ROAD, ANNAPOLIS, MARYLAND 21401-7435, November 2010, aRINC SPECIFICATION 653P1-3.

[24] G. Francois, "Study on I/O in time and space partitioned systems," National Aerospace Laboratory NLR, Tech. Rep. NLR-CR-2008-531-PT-1, November 2008.

[25] R. B. Sørensen, W. Puffitsch, M. Schoeberl, and J. Sparsø, "Message passing on a time-predictable multicore processor," in *Proceedings of the 17th IEEE Symposium on Real-time Distributed Computing (ISORC 2015)*. Aukland, New Zealand: IEEE, April 2015, pp. 51–59.

[26] J. Gaisler, "A portable and fault-tolerant microprocessor based on the SPARC v8 architecture," in *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2002, p. 409.

[27] F. J. Cazorla, R. Gioiosa, M. Fernandez, and E. Quiñones, "Multicore OS Benchmark," European Space Agency (ESA) and Barcelona Supercomputing Center (BSC), Tech. Rep. RFQ- 3-13153/10/NL/JK, 2012.

[28] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quiñones, M. Gerdes, M. Paolieri, and J. Wolf, "Merasa: Multi-core execution of hard real-time applications supporting analysability," *Micro, IEEE*, vol. 30, no. 5, pp. 66–75, 2010.

[29] L. Thiele and R. Wilhelm, "Design for timing predictability," *Real-Time Systems*, vol. 28, no. 2-3, pp. 157–177, 2004.

[30] S. A. Edwards and E. A. Lee, "The case for the precision timed (PRET) machine," in *DAC '07: Proceedings of the 44th annual conference on Design automation*. New York, NY, USA: ACM, 2007, pp. 264–265.

[31] S. A. Edwards, S. Kim, E. A. Lee, I. Liu, H. D. Patel, and M. Schoeberl, "A disruptive computer design idea: Architectures with repeatable timing," in *Proceedings of IEEE International Conference on Computer Design (ICCD 2009)*. Lake Tahoe, CA: IEEE, October 2009.

[32] I. Liu, J. Reineke, D. Broman, M. Zimmer, and E. A. Lee, "A PRET microarchitecture implementation with repeatable timing and competitive performance," in *Proceedings of IEEE International Conference on Computer Design (ICCD 2012)*, October 2012.

[33] I. Liu, "Precision timed machines," Ph.D. dissertation, EECS Department, University of California, Berkeley, May 2012.

[34] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee, "FlexPRET: A processor platform for mixed-criticality systems," in *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS)*, Berlin, Germany, April 2014.

[35] M. Fernández, R. Gioiosa, E. Quiñones, L. Fossati, M. Zulianello, and F. J. Cazorla, "Assessing the suitability of the ngmp multi-core processor in the space domain," in *Embedded Software (EMSOFT)*. Tampere, Finland: ACM, 2012, pp. 175–184.