parMERASA

Multi-Core Execution of Parallelised Hard Real-time Applications Supporting Analysability

D6.6 – Report on Experiences with Tiny Avionic RTE for Multi-core to the Standardisation Committees

Nature:	**R - Report**
Dissemination Level:	**RE - Restricted**
Due date of deliverable:	**September 30, 2013**
Actual submission:	**September 30, 2013**
Responsible beneficiary:	**BSC**
Responsible person:	**Eduardo Quiñones**

Grant Agreement number:	FP7-287519
Project acronym:	parMERASA
Project title:	Multi-Core Execution of Parallelised Hard Real-Time Applications Supporting Analysability
Project website address:	http://www.parmerasa.eu
Funding Scheme:	STREP SEVENTH FRAMEWORK PROGRAMME THEME ICT – 2011.3.4 Computing Systems
Date of latest version of Annex I against which the assessment will be made:	June 20, 2012
Project start:	October 1, 2011
Duration:	36 month
Period covered:	2012-07-01 to 2013-09-30

Project coordinator name, title and organisation: Prof. Dr. Theo Ungerer, University of Augsburg
Tel: + 49-821-598-2350 Fax: + 49-821-598-2359 Email: ungerer@informatik.uni-augsburg.de

Release Approval

name	role	date
Theo Ungerer	coordinator	30 – September – 2013
Pavel Zaykov	WP2 leader	30 – September – 2013
Eduardo Quiñones	WP5 leader	30 – September – 2013

TABLE OF CONTENTS

1 INTRODUCTION

The Integrated Modular Avionics (IMA) is an airborne network of safety-critical computers capable of supporting multiple applications of different criticality levels. The IMA allows multiple applications to share the same computing resources, leading to overall lower size and weight, reduced power needs, and lower maintenance costs compared to conventional federated approach, in which each computer is fully dedicated to one application.

IMA enables *incremental qualification* by providing *robust space and time partitioning* to avionics applications, under which the application functional and timing behaviour are isolated with respect to other applications. Time isolation rests on the hypothesis that the system exhibits the property of *time composability* meaning that the timing behaviour of an application, determined in isolation by the timing analysis, is not affected by the presence or the absence of other applications in the system.

The ARINC 653 standard [1] specifies the requirements and the Application Programming Interface (API) for robust time and space isolation. The current ARINC 653 standard fits the needs for the aerospace industry considering single-core systems. Within the current report, we identify the minimum ARINC 653 extensions towards multi-/many-core architectures. More specifically, we identify those extensions in the scope of the FP7 parMERASA project on the many-core processor architecture proposed within the project and modelled through simulation techniques. It is important to remark that the extensions presented can be applied to any many-core processor architecture fulfilling the properties defined in this deliverable. In this sense, we consider the parMERASA architecture to be a representative of architectures, such as Kalray MPPA 256 many-core processor [2].

Our plan to approach the standardization authorities is as follows: 1.The current document was produced in collaboration between HON and BSC. 2. The document will be sent to Benoit Triquet from airbus (parMERASA IAB member) for a review; 3. The document will be introduced to the ARINC 653 working group through the Honeywell International USA (working group member).

The rest of the report is organized as follows: In Section 2, we identify the problems of using ARINC 653 in multi-core processors. In Sections 3, 4 and 5, we introduce the main contributions to ARINC 653 standards from a software and hardware perspective. Section 6 introduces the suggested ARINC 653 API extensions towards multi-/many-core architectures. We conclude the report with Section 7.

2 ARINC 653 AND SINGLE-CORE AVIONICS SYSTEMS

Time isolation in IMA is mainly driven by the ARINC 653 standard, which encapsulates avionics applications into software partitions (SWP), each composed of functions named A653 processes (or simply processes). For each SWP, ARINC 653 assigns a *time capacity*, which defines the amount of CPU given to satisfy its processing requirements. To do so, a *time window* is allocated for each partition during which the system exclusively executes processes belonging to the SWP. It is important to remark that SWP provide time isolation at application level but not at process level. In other words, the SWP acts as a ”bubble” that ”protects” the functional and timing behaviour of processes associated to a SWP from others SWP’ processes, but not from the possible interactions of processes belonging to the same SWP.

Overall, the time partitioning imposed by ARINC 653 ensures that, in single-core execution models, the access to shared processor resources, such as the communication bus, memory controller or peripherals, is exclusive per partition, which allows SWPs to execute without affecting the timing behaviour of other SWPs.

Unfortunately, this is not the case for multi-core execution models. The simultaneous execution of multiple parallel applications makes processor resources shared at the same time by multiple processes belonging to different SWPs. This makes that the timing behaviour of one application can be affected by other applications due to interferences accessing hardware shared resources, and so breaking the robust time partition property required to provide incremental qualification. Overall, SWPs are not prepared to parallel execution, i.e. neither the simultaneous execution of applications nor the execution of parallel applications.

3 PROVIDE TIME PARTITIONING IN MANY-CORE AVIONICS SYSTEMS

When moving towards parallel execution on multi-core processors, A653 processes that form an application can be executed in parallel. Moreover, multiple parallel applications can run simultaneously. As a result, In order to be IMA-compliant, the time isolation property must remain the same as in single-core execution, i.e. *the execution of a parallel application cannot affect the timing behaviour of other parallel applications. By doing so, we can guarantee time composability so the timing analysis of parallel applications can be done in isolation and will not be affected by other applications.*

We introduce a new concept in the ARINC 653 standard to allow parallel execution on multi-core processors that extends the functionality of SWPs: the *parallel software partition* (pSWP). pSWP, described in Deliverable D5.3 from Milestone MS13, guarantees that parallel processes belonging to one application cannot affect the timing (and functional) behaviour of parallel processes belonging to other applications. Moreover, parallel processes belonging to the same application that do not require fulfilling the robust partitioning, and so within a pSWP, parallel processes can interference among them. Such interactions must be considered at WCET analysis time.

pSWP supports the communication methods defined in ARINC 653 standard, i.e. intra-partition and inter-partition:

- Communication among parallel processes belonging to the same application is performed through shared memory using intra-partition communication methods. Such communication must occur within partition boundaries in order to guarantee time isolation properties.
- Communication among applications is performed through message passing methods using inter-partition communication methods.

These two communication methods are performed through two separated logical memory regions: A *private memory region* accessible only by parallel processes belonging to the same pSWP through intra-partition communication methods, and a *shared memory region* accessible only by the source and destination pSWP through inter-partition communication methods.

Moreover, pSWPs provide methods to consider the impact of inter-partition communication requests at system integration time, as shown in the following equation:

$$WCET_{integration} = WCET_{isolation} + \Delta_{inter} ,$$

where $WCET_{integration}$ is the final WCET estimate of the application after system integration, $WCET_{isolation}$ the WCET estimate of the application computed in isolation, i.e., assuming that no other application is running in the system, and Δ_{inter} the maximum increment that $WCET_{isolation}$ can suffer due to inter-partition communications. See Deliverable D5.3 from Milestone MS13 to further details.

pSWP imposes requirements in the many-core processor architecture design. Next section describes them.

4 THE HARDWARE PLATFORM REQUREMENTS

pSWP relies on a new hardware feature to guarantee time isolation properties: The *Guaranteed Resource Partition* (GRP). GRP defines an execution environment composed of a cluster of processor resources, including cores, NoC resources, memory, etc., in which pSWPs run, providing the desirable time isolation properties as defined above. In other words, a GRP is a representation of ARINC 653 pSWP at architectural level. Deliverable D5.3 from Milestone MS13 provides further details on the hardware implementation of GRP. Next we summarise the main properties:

- In order to provide the time isolation properties to pSWPs, intra-partition communication request cannot exceed GRPs boundaries. To that end, in a GRP, cores communicate through the private GRP memory using the ARINC 653 intra-partition API, without interference from other GRPs. Moreover, the NoC design must guarantee that it does not exist a path from cores to memory that exits GRP boundaries.
- Cores from different GRPs communicate by writing data to each other's GRP memories using the ARINC 653 inter-partition API. In order to consider the impact that inter-partition communication may have on its WCET estimate (as expressed in the equation above), the memory device and the NoC must provide mechanisms to *freeze* intra-partition communication requests, and let inter-partition communication requests to proceed.

In Figure 1, we provide an example of the parMERASA architecture composed of four GRPs. Each GRP is composed of four processor cores and a memory (M) private to the cores, connected through own local Network-on-Chip (NoC), a tree. The GRPs are interconnected through global bus that also connects different GRP memories among them.

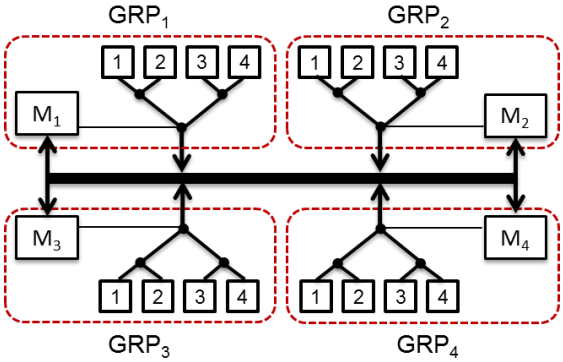

Figure 1. parMERASA architecture composed by four GRPs, each with four cores

A fundamental property of GRPs is to deliver time-predictability, i.e., to have bounded delays. The parMERASA architecture has been developed to provide such a time predictable behaviour. Below, we list and comment those shared architectural resources:

1) **Network-on-Chip** (**NoC**) – We differentiate two types of NoC with respect to the NoC location in the architecture: i) a tree, which connects cores inside of a cluster; ii) a bus, which is shared among the clusters.
2) **Memory Controller** – Data memory inside of a GRP is accessible by other GRP. In case of multiple concurrent (intra- and inter-GRP) memory accesses, the memory controller first grants the intra-GRP memory requests. To do so, two separated request queues are implemented in the memory controller.
3) **Input/output** (**I/O**) **interface** (Not visualized in Figure 1). The parMERASA architecture implements the Smart Interrupt Controller (SIC) and Tiny Interrupt Controller (TIC) to which I/O devices are connected.

We consider the parMERASA architecture to be a representative of many-core architectures, such as Kalray MPPA 256 many-core processor [2].

Overall, we recommend the architectural support for **GRPs** on many-core processor for bounding the contentions over shared resources. Such an architectural support would alleviate time-isolation analysis and easy the re-certification efforts of existing applications.

5 THE SOFTWARE PLATFORM REQUREMENTS

From software perspective, each processor core in a GRP has access to a single instance of the application binary, positioned in the GRP memory. Furthermore, we assume that there is one master (e.g., Core#0) and many slave/worker cores in a GRP. The master core is responsible for application/platform initialization. In Figure 2, we provide an exemplary code-snippet of an application running on the parMERASA architecture.

```
#define MASTER_CORE    0
#define CLUSTER_APP_1  0
APEX_UNSIGNED const core_id = core_id();
APEX_UNSIGNED const cluster_id = GRP_id();

if ((core_id == MASTER_CORE) && (cluster_id ==  CLUSTER_APP_1))
        {/*code for the Master core*/}
else
        {/*code for the Worker cores*/}
```

Figure 2. A code-snippet of exemplary application running on the parMERASA architecture

In Table 1, we summarize the set of assumptions for the parMERASA architecture. For sake of simplicity, we assume that the number of GRPs and the number of cores per GRP can be an arbitrary number, satisfying platform and application needs, respectively. For the scope of this project, we map each application on a pSWP, running inside of a single GRP. Furthermore, the mapping between ARINC 653 processes and processor cores in the GRP is also 1:1. In case the number of pSWP and ARINC 653 processes is higher than the number of GRPs and processor cores, we need to analyse the mapping possibilities in a way that the real-time requirements are fulfilled. In this line, it is worth to mention that the parMERASA architecture is prepared to support concurrent execution of multiple pSWPs among GRPs, as well as guarantees time isolation within a GRP.

Table 1. Set of assumptions for the parMERASA architecture

Assumptions	parMERASA architecture
Number of GRPs	N (depends on platform needs)
Number of cores per GRP	M (depends on application needs)
Application : A653 partition	1:1
Application : GRP	1:1
A653 process : core	1:1
Support concurrent execution of clusters	YES
Guarantee time & space isolation within a cluster	YES
Data Memory	Partition the memory space. Each GRP has one single-port data memory.
Network-on-Chip scheduling policy	Round-Robin

6 THE ARINC 653 API EXTENSIONS

The existing single-core ARINC 653 standard is composed of two basic blocks – an XML schema and an API. The ARINC 653 XML schema contains the number and execution sequence of the partitions, and the potential inter-partition communication. The ARINC 653 API encapsulates the coding standards for the avionics applications such as the type of variables and API calls.

Based on our experience on the applicability of a subset (referred as tiny avionic RTE and described in deliverable D4.6 from Milestone MS13) of the single-core ARINC 653 standard on multi-core architectures, we suggest preserving the existing API. Furthermore, we propose an extension of the XML schema and API, missing in the original standard to support the definition of pSWP and multi-core execution. Those newly proposed functionalities are related with the system initialization and application mapping on the parMERASA architecture.

Below, we propose the list of augmentations/recommendations towards multi-core ARINC 653 extension:

1) **Extension of the SWP functionality to support pSWP**. Although the A653 partition related APIs are not modified, its implementation must change to support the properties of pSWP as defined in Section 2.
2) pSWPs relies on the processor architecture supporting GRPs. Therefore, ARINC 653 requires the architecture to implement GRP.
3) **Mapping** of ARINC 653 processes and pSWP to processor cores and GRPs. The mapping shall be performed during the initialization phase and shall be part of the ARINC 653 XML-schema. An API extension might be considered as well. The mapping of ARINC 653 processes to master and worker cores might be achieved through the help of ***core_id()*** and ***GRP_id()*** functions, which return the identification of the current core and cluster, respectively.
4) **Port & virtual channel initialization** functionality connects two ports and creates a virtual channel between the sending and receiving pSWP through the NoC. The port and virtual channel initialization shall be part of the ARINC 653 XML-schema with a

corresponding API extension. Such a port communication is also related to pSWP-to-GRP mapping.

5) **Barrier** functionality stalls the execution of the cores at the end of the initialization phase, waiting for all processor cores from all pSWPs to be initialized. The necessity for the barrier functionality arises in case of an inter-process communication is initiated, before the designated process running on a different core completes its initialization. The barrier functionality might be implemented with the help of the existing ARINC 653 API.

7 CONCLUSIONS

In this report, we introduce the parMERASA many-core architecture, we list the platform assumptions, and we envision the extensions/recommendations for the upcoming multi-core ARINC 653 standard.

REFERENCES

1. "Avionics Applications Software Standard Interface - part 1 (required services)," ARINC-Specification 653P1-3, technical report, Aeronautical Radio, 2010.
2. "KALRAY Programmable multi-purpose processor array (MPPA 256)," www.kalray.eu, 2013.