parMERASA

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

D4.6 – Description of Tiny Avionic RTE Functions

Nature:	**R - Report**
Dissemination Level:	**PU - Public**
Due date of deliverable:	**30/09/2013**
Actual submission:	**30/09/2013**
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

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	WP leader	2013-30-09
Theo Ungerer	coordinator	2013-30-09
Miloš Panić	main contributor	2013-30-09
Pavel Zaykov	contributor	2013-30-09

Deliverable Summary

Deliverable D4.6 “Description of Tiny Avionic RTE Functions” summarises the implementation work done within the scope of task T4.4 to provide system software for the aerospace evaluation demonstrator that is due in project month M36. More precisely, we introduce a subset of the ARINC 653 standard [1] and a few new functionalities referred hereafter as tiny avionic RTE. In the sections to follow, we provide details of the tiny avionic RTE integration, i.e., application specific software stack, to the parMERASA system architecture, and other aerospace activities related to task T4.4. The deliverable fulfils the tiny avionic RTE specific parts of project milestone 10.

Task description of T4.4 (m17::8m):
Tiny Avionic RTE Implementation and AUTOSAR Interface (see DoW, sect. 1.3.3, p. 49)

- Implementation of tiny avionic RTE and a subset of the ARINC 653 API services on top of Multi-core RTOS Kernel.
- Device drivers required for avionics support.
- Board supporting package development for running avionics applications on COTS multi-core hardware platform.
- Report on experiences with tiny avionic RTE for multi-cores to the IMA Standardisation Committee (see also D6.6 in WP6).
- Fulfilment of tiny avionic RTE requirements defined in task 4.1 and system software interface defined in task 2.2 will be checked in month 24.
- Targets after month 24 are a specification and implementation of an tiny avionic RTE for the parMERASA multi-core processor as extension of the RTOS Kernel from task 4.2 and a commercial COTS multi-core platform that support the avionics case study.

Conclusion of task 4.4:

Targets of task 4.4 are successfully reached. In the scope of D4.5, we invested effort to provide device drivers (implemented as part of the tiny avionic RTE library) for the parMERASA platform. The functionality of the tiny avionic RTE library was already successfully exercised with the help of small benchmark and the 3D Path Planning application. For further details, please refer to Prototype Description deliverable. The effort in task T4.4 anticipated also some minor activities for preparation of execution platform based on COTS multi-core hardware. In the scope of these activities we analysed COTS hardware platforms – based on Freescale P4080 and Intel i7 processors. Some of our preliminary findings with the P4080 have been presented at the parMERASA workshops and published in [1].

For the aerospace demonstrator, due in M36, we selected a multi-core COTS platform based on Intel’s i7 processor. Board supporting package (BSP) has been delivered and an operating system has been selected and deployed to form an execution platform we will exercise with for the aerospace evaluations and comparison study.

We have a functional tiny avionic RTE library working with avionics application on the parMERASA system architecture and COTS platform (Intel i7) with fully functional BSP and RTOS.

Table of Contents

1 Introduction

The parMERASA work package 4 develops common system architecture for a many-core processor suitable for the three application domains automotive, avionic, and construction machinery. The system architecture covers all layers from simulated hardware to application layer. The main objective is to support the execution of parallelized hard real-time applications and their Worst-Case Execution Time (WCET) analysability. Therefore, a domain specific runtime environment (RTE) serves as base for the applications developed in parMERASA work package 2.

The parMERASA system architecture as well as the concept of a *Kernel Library* for many-core processors was introduced in deliverable 4.1 and implemented in deliverable 4.2. The domain specific RTEs were also specified in deliverable 4.1. A comprehensive overview of the parMERASA system architecture for cross-domain usage on embedded hard real-time many-core systems is presented in [2].

The avionics domain-specific RTE is presented by the ARINC 653 standard [2], referred as tiny avionic RTE. In this deliverable, we describe the implementation of the tiny avionic RTE (working prototype contained in deliverable 4.5). The main goal of the tiny avionic RTE is to support the execution of parallelised hard real-time applications from the avionics domain. The tiny avionic RTE is based on the Kernel Library running on a simulated many-core processor. The Kernel Library was implemented in task 4.2 (deliverable 4.2). Furthermore, the tiny avionic RTE is designed and implemented in a way that it fits into the parMERASA system architecture specified in task 4.1 (deliverable 4.1).

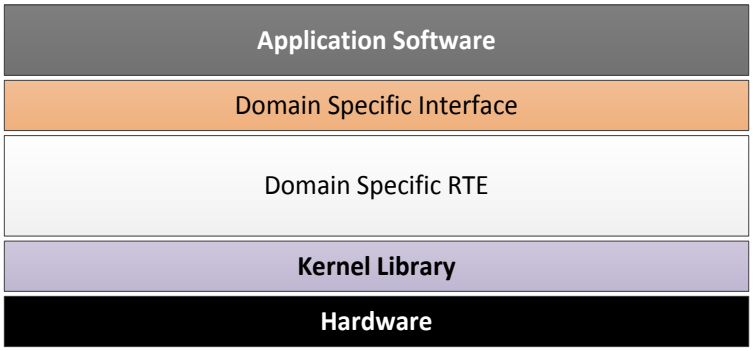

Figure 1: The parMERASA system architecture - conceptual overview

In Figure 1, we introduce a conceptual overview of the parMERASA system architecture. Each application domain is based on the common *Kernel Library*, which abstracts the underlying hardware and provides basic functionalities commonly employed among all domain specific implementations. Kernel library provides an abstraction layer to the d*omain specific runtime environment (RTE)*. For each of the three application domains, a separate RTE is implemented to provide a reasonable subset of the application programming interface (API) defined in the domain specific standards such as ARINC 653). Closely coupled to the RTEs are the *domain specific interfaces* described in the domain specific standards. The industrial *application software* are positioned at the top of the software stack and implemented with the domain specific interface.

The rest of the deliverable is organized as follows. In Section 2, we introduce the tiny avionic RTE motivation and we outline the main concepts. In Section 3, we specify and explain the parts of ARINC 653 API which the tiny avionic RTE covers and the extensions required that allow the execution in the targeted many-core architectures.

2 Tiny Avionic RTE Overview

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. Furthermore, the 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.

The ARINC 653 standard [3] specifies the requirements and the Application Programming Interface (API) for robust time and space isolation by encapsulating the avionics applications into separate software partitions. Since the existing ARINC 653 standard targets single-core processes, within the current report, we identify the "must-have" ARINC 653 extensions towards multi-/many-core architectures. More specifically, we identify those extensions in the scope of the FP7 parMERASA project on custom many-core architecture (i.e., simulator) proposed within the project.

The idea of the tiny avionic RTE is to implement a minimal subset of the ARINC653 API required by our avionics case studies. Beyond that, the tiny avionic RTE is capable of running on many-core processor systems featuring clustered architectures. Due to that it has to provide mechanisms for assignment partitions to clusters and processes to cores as well as intra- and inter-partition communication and synchronisation features to exchange data between cores.

In Figure 1, we introduce an overview of the parMERASA system architecture. More precisely, Figure 2 illustrates how the tiny avionic RTE fits into the overall system architecture. It is divided in *application layer*, *tiny avionic Runtime Environment*, *Kernel Library*, and the target *hardware*. The

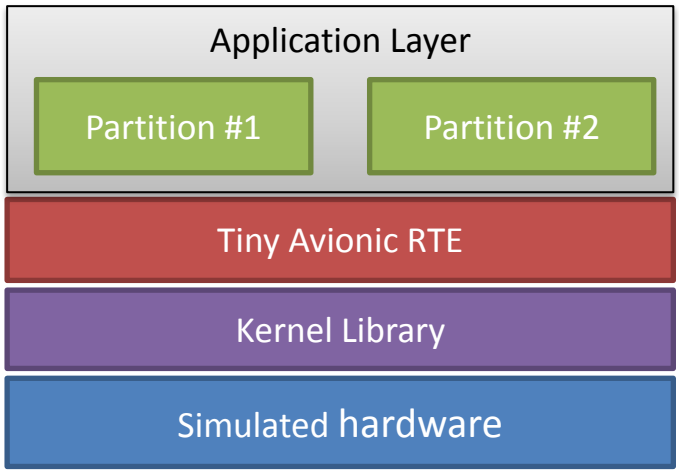

Figure 2 - Integration of the tiny avionic RTE to the parMERASA system architecture

application layer consists of one or more software partition. The tiny avionic RTE covers the necessary subset of the ARINC 653 API and XML schema. The tiny avionic RTE builds upon Kernel Library (deliverable 4.2), which abstracts the simulated hardware and provides required functionality for implementing the tiny avionic RTE functions.

3 Tiny Avionic RTE Specification

As stated in the previous section, the existing ARINC 653 standard fits the needs for the aerospace industry considering single-core systems only. 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. In our avionics case study, the functionality of the XML schema is implemented by custom functions, which initialize the partitions manually, at the beginning of their execution. The ARINC 653 API encapsulates the coding standards for the avionics applications such as the type of variables and API calls. The tiny avionic RTE preserves the existing ARINC653 API, and adds the additional functions to support execution on many-core platforms.

APEX_BUFFER
CREATE_BUFFER
SEND_BUFFER
RECEIVE_BUFFER
GET_BUFFER_ID
GET_BUFFER_STATUS

APEX_PARTITION
GET_PARTITION_STATUS
SET_PARTITION_MODE

APEX_PROCESS
CREATE_PROCESS
SET_PRIORITY
SUSPEND_SELF
SUSPEND
RESUME
STOP_SELF
STOP
START
DELAYED_START

LOCK_PREEMPTION
UNLOCK_PREEMPTION
GET_MY_ID
GET_PROCESS_ID
GET_PROCESS_STATUS
BIND_PROCESS_TO_CORE

APEX_QUEUE
CREATE_QUEUING_PORT
CONNECT_SRCDST_QUEUING_PORTS
SEND_QUEUING_MESSAGE
RECEIVE_QUEUING_MESSAGE
GET_QUEUING_PORT_ID
GET_QUEUING_PORT_STATUS

APEX_SEMAPHORE
CREATE_SEMAPHORE
WAIT_SEMAPHORE
SIGNAL_SEMAPHORE
GET_SEMAPHORE_ID
GET_SEMAPHORE_STATUS

Table 1 - Considered ARINC 653 API and proposed extension in the tiny avionic RTE implementation

3.1 Tiny Avionic RTE Functions

The list of functions that the tiny avionic RTE implements (marked with black), is shown in Table 1. The entries marked with grey present ARINC653 functions that are not implemented (our avionics case studies do not require them). The ones marked with red are not part of ARINC653 API and present the proposed extension to support many-core execution.

Our implementation of the ARINC 653 presents the following categories:

- **Partition management:** APEX_PARTITION provides time isolation. A partition contains at least one APEX_PROCESS. The tiny avionic RTE supports only 2 modes for partitions. A partition is created in IDLE mode for the initialization phase. After the initialization phase in

which processes, buffers and queues are created, the partition mode is set to NORMAL with SET_PARTITION_MODE function. GET_PARTITION_STATUS returns a structure that contains identifier, period, operating etc. of a partition.

- **Process management:** APEX_PROCESS is a unit of execution in the system. It implements the functionality of the system. CREATE_PROCESS allocates the space for the process, creates the stack, sets period, priority, deadline, etc. Once created, the process has to be assigned to a specific core by calling BIND_PROCESS_TO_CORE. When the mode of the partition that contains the process changes to NORMAL, all processes of that partition can start the execution by calling START function. GET_MY_ID and GET_PROCESS_STATUS return PROCESS_ID and PROCESS_STATUS structures respectively.

- **Inter-partition communication:** The communication between processes in different partitions is done through APEX_QUEUES. A queue is created with CREATE_QUEUING_PORT call that allocates a simple FIFO structure to which processes read and write by calling RECEIVE_QUEUING_MESSAGE and SEND_QUEUING_MESSAGE respectively. Ports are unidirectional and can be SOURCE or DESTINATION. During the initialization of the system CONNECT_SRCDST_QUEUING_PORTS is called that binds a source to a destination port. GET_QUEUING_PORT_ID and GET_QUEUING_PORT_STATUS return the id and status structures of the port.

- **Intra-partition communication:** The communication between processes that belong to the same partition is done through APEX_BUFFER. The buffers are created with CREATE_BUFFER call that allocates the necessary space. The tiny avionic RTE supports only FIFO discipline. RECEIVE_BUFFER and SEND_BUFFER are read/write operations that are exclusive (implemented via ticket locks from Kernel Library). GET_BUFFER_ID returns the identifier of the buffer.

- **Synchronisation primitives:** Synchronization is done through APEX_SEMAPHORE that are implemented with abovementioned ticket locks. For the creation of a semaphore, CREATE_SEMAPHORE is used. A lock is obtained by calling WAIT_SEMAPHORE and released with SIGNAL_SEMAPHORE. GET_SEMAPHORE_ID returns the identifier of the semaphore.

- **Time management** and **error handling** are not covered by the current implementation of the tiny avionic RTE.

List of Figures/Tables

List of References

[1] „parMERASA – Multi-Core Execution of Parallelised Hard Real-Time Applications Supporting Analysability," in *DSD*, 2013.

[2] C. Bradatsch, F. Kluge and T. Ungerer, "A Cross-Domain System Architecture for Embedded Hard Real-Time Many-Core Systems," in *11th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC-13), accepted for publication*, China, 2013.

[3] „ARINC Specification 653: Avionics Application Software Standard Standard Interface, Part 1 and 4, Subset Services," ARINC Inc., 2012.