parMERASA

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

D4.4 – Description of Tiny Automotive RTE Functions

Nature:	**R - Report**
Dissemination Level:	**PU - Public**
Due date of deliverable:	**30/09/2013**
Actual submission:	**30/09/2013**
Responsible beneficiary:	**UAU**
Responsible person:	**Theo Ungerer**

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
Christian Bradatsch, Florian Kluge	author	2013-09-20
Theo Ungerer	WP leader	2013-09-20
Theo Ungerer	coordinator	2013-09-20

Deliverable Summary

Deliverable D4.4 “Description of Tiny Automotive RTE Functions” concerns the function descriptions of the tiny automotive RTE implementation, which supports the execution of the parallelised hard real-time application from the automotive domain.

This deliverable comprises the integration of the tiny automotive RTE in the overall parMERASA system architecture. Furthermore, it illustrates the concept of the tiny automotive RTE. It also explains how configuration and generation of API functions lead to an application specific software stack.

Deliverable D4.4 summarises the work done in task T4.3 to fulfil the tiny automotive RTE specific parts of milestone 10.

Task description of T4.3 (m17::8m):
Tiny Automotive RTE Implementation and AUTOSAR Interface (see DoW, sect. 1.3.3, p. 48)

- Implementation of AUTOSAR OS subset on top of Multi-core RTOS Kernel.
- Driver interfaces for automotive applications and AUTOSAR MCAL (Micro Controller Abstraction Layer) wrappers to RTOS kernel.
- Implementation of AUTOSAR multi-core communication features.
- Report on experiences with tiny automotive RTE for multi-cores to the AUTOSAR Standardisation committee (see also D6.5 in WP6).
- Fulfilment of tiny automotive 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 a tiny automotive RTE for the parMERASA multi-core processor as extension of the RTOS Kernel from task 4.2 and a tiny automotive RTE Interface for the applications.

Conclusion of task T4.3:

All targets of task T4.3 and deliverable D4.4 have been reached.

Table of Contents

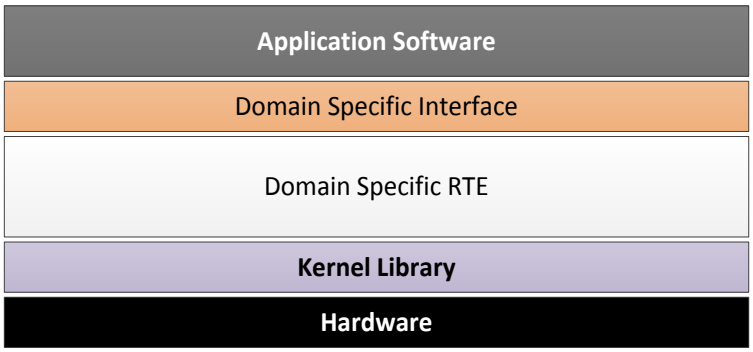

Figure 1: Conceptual Overview of the parMERASA System Architecture

1 Introduction

Work package 4 develops a 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 WCET analysability. Therefore, a domain specific runtime environment serves as base for the applications developed in work package 2.

The parMERASA system architecture as well as the concept of a *Kernel Library* for many-core processors was introduced in deliverable D4.1 and implemented in deliverable D4.2. The domain specific runtime environments were also specified in deliverable D4.1. A comprehensive overview of the parMERASA system architecture for cross-domain usage on embedded hard real-time many-core systems is presented in the attached paper, which will be published [1].

For the automotive domain, a subset of AUTOSAR, called tiny automotive RTE, and additional optional modules represent the domain specific runtime environment. Deliverable D4.3 includes the implementation of the tiny automotive RTE and its functionality is described in this document. The main goal of the tiny automotive RTE is to support the execution of parallelised hard real-time applications from the automotive domain. The tiny automotive RTE is based on the Kernel Library running on a simulated many-core processor. The Kernel Library was implemented in task T4.2 (cf. deliverable D4.2). Furthermore, the tiny automotive RTE is designed and implemented in a way that it fits into the parMERASA system architecture specified in task T4.1 (cf. deliverable D4.1).

Figure 1 gives a conceptual overview of the parMERASA system architecture. Each domain is based on the common *Kernel Library* that abstracts the underlying hardware and provides basic functionalities commonly used among all domain specific implementations. Building on this, the d*omain specific runtime environment (RTE)* follows. 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 (e.g. AUTOSAR). Closely coupled to the RTEs are the *domain specific interfaces*. On top, the particular industrial *application software* is executed.

Section 2 motivates the tiny automotive RTE and describes its main idea. Section 3 specifies the modules and their functions being part of the tiny automotive RTE as well as the generated API functions. Section 4 points to the implementation and gives an outlook to the future project months.

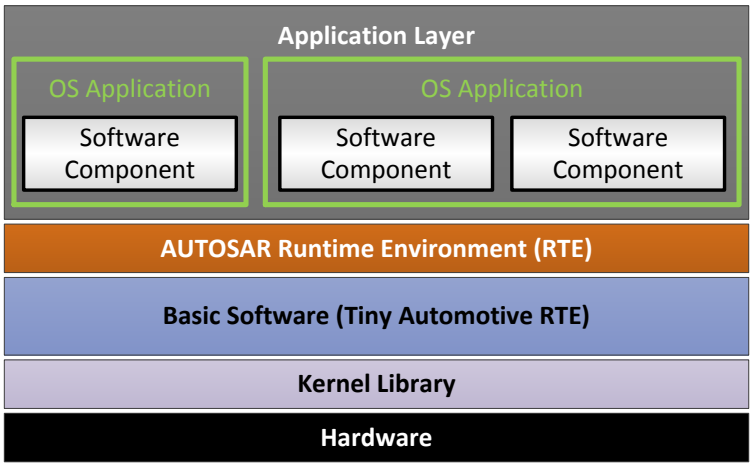

Figure 2: Integration of the Tiny Automotive RTE in the parMERASA System Architecture

2 Tiny Automotive RTE Overview

AUTOSAR is a worldwide development partnership of car manufacturers, suppliers and other companies from the electronics, semiconductor and software industry. It standardises a software architecture, services and a methodology including configuration and description methods. The software architecture is divided in three parts. The lowest layer with direct access to the Microcontroller is the Basic Software (BSW) [2], which is subdivided into several BSW modules. The Run-Time Environment (RTE) separates it from the application layer on top. With the AUTOSAR 4.0 specification [3], multi-core processors found their way into the standard. Over the last years, the number of processing cores per chip was constantly increasing. One reason for the usage of multi-cores in the automotive domain is more computing power for additional functions, but with less increased energy consumption.

The idea of the tiny automotive RTE is to define a *minimal* subset of the AUTOSAR software architecture, which builds an evaluation platform for *parallelized* and *in parallel executed* automotive applications. The intention of the tiny automotive RTE is to run on multi-/many-core processor systems. To achieve these goals, the tiny automotive RTE

- comprises only basic functionalities required on any core and
- provides basic communication and synchronization features for data transmission inside a multi-/many-core processor.

The tiny automotive RTE subset is composed of a minimal set of AUTOSAR BSW modules, which retains compatibility to the AUTOSAR API. The tiny automotive RTE BSW modules provide the basic functionalities extended by adapted communication and synchronization mechanisms.

Figure 2 illustrates how the tiny automotive RTE fits into the overall parMERASA system architecture. The tiny automotive RTE is divided in *AUTOSAR Runtime Environment* and *Basic Software*. The *Basic Software* corresponds to the *domain specific RTE* of Figure 1. The *AUTOSAR Runtime Environment* of Figure 2 relates to the *domain specific interface* and has nothing to do with the *domain specific RTE*. The a*pplication layer* can comprise one or more different user applications, called *OS Applications*.

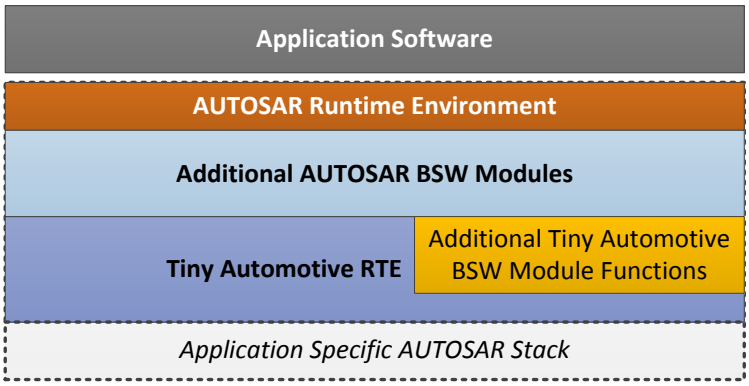

Figure 3: Structure of Application Specific AUTOSAR Stack

An application in turn consists of one or more *Software-Components (SW-Cs)*. The *AUTOSAR RTE* acts as a connector between SW-Cs and the *Basic Software (BSW)*. It provides standardized interfaces for accessing services of the BSW. For example, the BSW incorporates modules providing memory, communication, and the OS services. In the tiny automotive RTE, only the most necessary BSW modules are included. The *Kernel Library* abstracts from the underlying hardware and provides basic functionalities, for example many-core support amongst others. These functionalities reside on a lower level than those of the tiny automotive RTE.

3 Tiny Automotive RTE Specification

The AUTOSAR software stack comprises several BSW modules. Not every automotive application requires the functionality of the full software stack. The relevant modules are taken from the full stack according to a particular configuration and assembled to an application specific stack. Figure 3 shows the role of the tiny automotive RTE in an application specific AUTOSAR stack. The tiny automotive RTE defines a minimal subset of the full AUTOSAR software stack. For applications requiring additional functionality, any AUTOSAR BSW module can be appended. Some BSW modules belonging to the tiny automotive RTE are stripped down and do not include all functions to keep the code basis small. This is due to keep the tiny automotive RTE subset minimal. The missing functions are provided as additional modules and can optionally be added to an application specific software stack. In the end, the application specific AUTOSAR stack comprises the tiny automotive RTE, optional tiny automotive RTE BSW module functions, optional AUTOSAR BSW modules, and a generated AUTOSAR Runtime Environment.

For each application a set of configuration files defines, which modules are required and the particular module configuration options. Most times, these configuration files are the output of automotive development tools. By means of these configurations an application specific software stack is built. There are two kinds of module functions:

1. Regular **functions** with a consistent name and implementation.
2. **API functions** with parameterizable names and a particular implementation for each unique function name.

For the first type, the tiny automotive RTE specifies the modules and corresponding functions prototypes, which shall exist. The second type of functions refers nearly exclusively to the RTE module. The task of the AUTOSAR RTE generator is to generate these functions according to the configuration of the application. The tiny automotive RTE also specifies the API functions, which shall be supported by the tiny automotive RTE generator.

3.1 Standard Functions

The AUTOSAR specification release 4.1 revision 1 comprises 66 Basic Software modules [4]. In the tiny automotive RTE the number of BSW modules is reduced to 6 essential modules. Table 1 gives an overview of the tiny automotive RTE modules and the corresponding functions. They are explained below. Grey highlighted functions are part of the particular module in the AUTOSAR specification, but can be left out for the tiny automotive RTE. They may be added as additional functions to an application specific stack like other BSW modules.

The most important module is the Operating System module [3]. It also involves the functionality of the OSEK/VDX OS specification [5]. It is responsible for providing and managing execution contexts of applications and system services, which consist of tasks and/or interrupt service routines (ISRs). The OS module can be coarsely structured in the following system service groups: Task management is responsible for task execution and priority pre-emptive scheduling of them; likewise, interrupt handling for scheduling ISRs. Event control communicates binary information between tasks. Resource management is in charge of controlling accesses to shared resources. Counters, alarms, and schedule-tables enable timed and event triggered actions. Protection facilities allow memory, timing, and service protection and the request of access rights. All other services can be grouped to general operating system execution control. Several OS services are based on Kernel Library functionalities, for example scheduling and spatial/timing protection.

Some OS services depend on the hardware platform and not every automotive application needs each OS service. However, all OS functions are included in the tiny automotive RTE. A strip-down of the OS module makes no sense, because of the close entanglement of functions.

Similarly, the Micro Controller Unit (MCU) driver is also closely related to the hardware. It initializes MCU specific hardware and delivers status information. Most MCU driver functions depend on particular hardware support and can be omitted according to the MCU specification. In the tiny automotive RTE specification all functions are included, due to the fact that if a certain feature exists, it shall be supported by the MCU driver. Thus, functions of unsupported hardware features are excluded by configuration and not linked to the software stack.

The Electronic Control Unit (ECU) State Manager initializes and de-initializes the OS, the Schedule Manager and the BSW Mode Manager amongst other BSW modules. One major target of it is to manage the proper start-up of all cores and the shutdown of the ECU. In accordance with the AUTOSAR specification of the ECU State Manager, it is also responsible for the ECU sleep modes. This is not a necessary feature for application execution. Hence, it is left out in the tiny automotive RTE. For applications requiring this feature, the missing functions can be added to the application specific software stack.

The BSW Mode Manager is in charge of mode arbitration and mode control. “A mode is a certain set of states of the various state machines that are running in the vehicle that are relevant to a particular

entity, e.g. a SWC, a BSW module, an application, a whole vehicle." [6, p. 8] For example, the different states of the ECU State Manager are implemented as AUTOSAR modes and controlled by the BSW Mode Manager. All functions relying on BSW modules not incorporated in the tiny automotive RTE belong to the additional BSW module functions, e.g. CAN, LIN.

The RTE makes the application independent from a specific ECU. It serves as a separation between the layered architecture design of the BSW and the component-based design of the application. The RTE is the realization of Assembly-Connectors between Software-Components in the Virtual Functional Bus and their access to the BSW modules, including the OS and communication services. There are only few functions for starting and stopping the RTE and partitions, all others are API functions (see section 3.2). Likewise, the BSW Scheduler only includes functions for initialization and de-initialization.

3.2 Generated API Functions

Beside the mentioned modules and functions, there are further API functions, which are created by the RTE generator. The functions' implementations differ from application to application. Even the function names are parameterized and they can also differ in implementation inside of an entity. Most of these API functions are specified in the AUTOSAR Runtime Environment [7], but also in the Inter OS-Application Communicator (IOC) belonging to the OS module [3]. Also, the BSW Scheduler and its corresponding API functions are generated. Thus, in our opinion, no RTE, Schedule Manager, or IOC functions can be omitted. Table 2 lists the corresponding generator created functions supported by the tiny automotive RTE.

The RTE API functions primarily enable data exchange and function calls between different Runnables/SW-Cs and between SW-Cs and BSW modules through ports. For example, an application consisting of a SW-C providing a sender port and two SW-Cs each is requiring a receiver port. For each SW-C with receiver port the corresponding Rte_Receive function has another function name. If one SW-C with receiver port is located on another core, even the implementation of the Rte_Receive function might be different.

Similarly, the IOC module provides API functions for data exchange, but between OS-Applications. The BSW Scheduler also offers API functions for sending/receiving and function invocation, however for communication between BSW modules. Moreover, it comprises API functions for request to the BSW Mode Manager and for handling shared resources between BSW modules, comparable to the resource management functions of the OS module.

4 Conclusion

Since the automotive application of work package 2 does not use all functionalities specified in the tiny automotive RTE, only actually required modules and functions are implemented. Deliverable D4.3 contains a list of the implemented functions. It gives also an overview of the tiny automotive RTE generator implementation.

One task for the next period after month 24 is to parallelize the tiny automotive RTE. Therefore, some BSW modules will be distributed over several cores, inspired by the Factored Operating System (fos) [8] concept. In this course, the CAN functional cluster will be explored, which comprises all BSW

modules for communication over a CAN bus. Also, the usage of the Diagnostic Event Manager (DEM) in a fos orientated system software on a many-core system will be analysed.

Table 1: AUTOSAR Modules and Functions required by Tiny Automotive RTE

OS	
ActivateTask	ShutdownOS
TerminateTask	GetApplicationID
ChainTask	GetISRID
Schedule	CallTrustedFunction
GetTaskID	CheckISRMemoryAccess
GetTaskState	CheckTaskMemoryAccess
DisableAllInterrupts	CheckObjectAccess
EnableAllInterrupts	CheckObjectOwnership
SuspendAllInterrupts	StartScheduleTableRel
ResumeAllInterrupts	StartScheduleTableAbs
SuspendOSInterrupts	StopScheduleTable
ResumeOSInterrupts	NextScheduleTable
GetResource	StartScheduleTableSynchron
ReleaseResource	SyncScheduleTable
GetSpinlock	GetScheduleTableStatus
ReleaseSpinlock	SetScheduleTableAsync
TryToGetSpinlock	IncrementCounter
SetEvent	GetCounterValue
ClearEvent	GetElapsedValue
GetEvent	TerminateApplication
WaitEvent	AllowAccess
GetAlarmBase	GetApplicationState
GetAlarm	GetNumberOfActivatedCores
SetRelAlarm	GetCoreID
SetAbsAlarm	StartCore
CancelAlarm	StartNonAutosarCore
GetActiveApplicationMode	ShutdownAllCores
StartOS	ControlIdle

MCU Driver	MCU_GetResetReason
MCU_Init	MCU_GetResetRawValue
MCU_InitRamSection	MCU_PerformReset
MCU_InitClock	MCU_SetMode
MCU_DistributePllClock	MCU_GetVersionInfo
MCU_GetPllStatus	MCU_GetRamState

ECU State Manager
EcuM_GetVersionInfo
EcuM_GoDown
EcuM_GoHalt
EcuM_GoPoll
EcuM_Init
EcuM_StartupTwo
EcuM_Shutdown
EcuM_SelectShutdownTarget
EcuM_GetShutdownTarget
EcuM_GetLastShutdownTarget
EcuM_SelectShutdownCause
EcuM_GetShutdownCause
EcuM_GetMostRecentShutdown

EcuM_GetNextRecentShutdown
EcuM_GetPendingWakeupEvents
EcuM_ClearWakeupEvent
EcuM_GetValidatedWakeupEvents
EcuM_GetExpiredWakeupEvents
EcuM_SetRelWakeupAlarm
EcuM_SetAbsWakeupAlarm
EcuM_AbortWakeupAlarm
EcuM_GetCurrentTime
EcuM_GetWakeupTime
EcuM_SetClock
EcuM_SelectBootTarget
EcuM_GetBootTarget

Basic Software Mode Manager
BswM_BswMModeRequest
BswM_BswMModeSwitchNotification
BswM_BswMPartitionRestarted
BswM_GetVersionInfo
BswM_Init
BswM_Deinit
BswM_RequestMode
BswM_TriggerSlaveRTEStop
BswM_TriggerStartUpPhase2
BswM_CanSM_CurrentIcomConfiguration
BswM_CanSM_CurrentState
BswM_ComM_CurrentMode
BswM_ComM_CurrentPNCMode
BswM_Dcm_ApplicationUpdated
BswM_ Dcm_CommunicationMode_CurrentState

BswM_EcuM_CurrentState
BswM_EcuM_CurrentWakeup
BswM_EthSM_CurrentState
BswM_FrSM_CurrentState
BswM_J1939DcmBroadcastStatus
BswM_J1939Nm_StateChangeNotification
BswM_LinSM_CurrentSchedule
BswM_LinSM_CurrentState
BswM_LinTp_RequestMode
BswM_NvM_CurrentBlockMode
BswM_NvM_CurrentJobMode
BswM_Sd_ClientServiceCurrentState
BswM_ Sd_ConsumedEventGroupCurrentState
BswM_Sd_CurrentState
BswM_Sd_EventHandlerCurrentState
BswM_WdgM_RequestPartitionReset

RTE
Rte_Start
Rte_Stop
Rte_PartitionTerminated

Rte_PartitionRestarting
Rte_RestartPartition
Rte_Init
Rte_StartTiming

BSW Scheduler
SchM_Init

SchM_Deinit
SchM_GetVersionInfo

Table 2: AUTOSAR RTE Generator created Functions required by Tiny Automotive RTE

IOC	IocSend
IocRead	IocReceive
IocWrite	IocSendGroup
IocReadGroup	IocReceiveGroup
IocWriteGroup	IocEmptyQueue

RTE	
Rte_Ports	Rte_IRead
Rte_NPorts	Rte_IWrite
Rte_Port	Rte_IWriteRef
Rte_Write	Rte_IInvalidate
Rte_Send	Rte_IStatus
Rte_Switch	Rte_IrvIRead
Rte_Invalidate	Rte_IrvIWrite
Rte_Feedback	Rte_IrvRead
Rte_SwitchAck	Rte_IrvWrite
Rte_Read	Rte_Enter
Rte_DRead	Rte_Exit
Rte_Receive	Rte_Mode
Rte_Call	Enhanced Rte_Mode
Rte_Result	Rte_Trigger
Rte_Pim	Rte_IrTrigger
Rte_CData	Rte_IFeedback
Rte_Prm	Rte_IsUpdated

BSW Scheduler	SchM_Switch
SchM_Enter	SchM_Mode
SchM_Exit	Enhanced SchM_Mode
SchM_Call	SchM_SwitchAck
SchM_Result	SchM_Trigger
SchM_Send	SchM_ActMainFunction
SchM_Receive	SchM_CData

List of Figures/Tables

List of References

[1] 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.

[2] AUTOSAR, „Layered Software Architecture, Document Version 3.3.0,“ 2013.

[3] AUTOSAR, „Specification of Operating System, Document Version 5.0.0,“ 2011.

[4] AUTOSAR, „List of Basic Software Modules, Document Version 1.7.0,“ 2013.

[5] OSEK/VDX steering committee, “OSEK/VDX Operating System Specification Version 2.2.3,” 2005.

[6] AUTOSAR, „Requirements on Mode Management, Document Version 3.0.0,“ 2013.

[7] AUTOSAR, „Specification of RTE, Document Version 3.3.0,“ 2013.

[8] D. Wentzlaff und A. Agarwal, „Factored Operating Systems (fos): The Case for a Scalable Operating System for Multicores,“ *SIGOPS Operating Systems Review,* Nr. 43, pp. 76-85, April 2009.