This Project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement: 688860

Project title:	High-Performance Real-time Architectures for Low-Power Embedded Systems
Acronym:	HERCULES
Project ID:	688860
Call identifier:	H2020 - ICT 04-2015 - Customised and low power computing
Project Coordinator:	Prof. Marko Bertogna, University of Modena and Reggio Emilia

D4.2: Lightweight RTOS (final)

Document title:	Lightweight RTOS - final
Version:	1.1
Deliverable No.:	D4.4
Lead task beneficiary	EVI
Partners Involved:	EVI
Author:	Claudio Scordino, Paolo Gai, Nicola Serreli
Status:	Final
Date:	30/6/2018
Nature[1]:	S

Dissemination level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services – CS & IAB)	
RE	Restricted to a group specified by the consortium (including the IAB)	
CO	Confidential to consortium (including CS & IAB)	

[1] **For deliverables: R** = Report; **P** = Prototype; **D** = Demonstrator; **S** = Software/Simulator; **O** = Other
For milestones: O = Operational; **D** = Demonstrator; **S** = Software/Simulator; **ES** = Executive Summary; **P** = Prototype

This Project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement: 688860

Document history:

Version	Date	Author	Comments
0.1	2018-06-18	EVI	First version
0.2	2018-06-26	PIT	Internal review by PIT.
1.0	2018-06-28	EVI	Integrated review comments
1.1	2018-06-30	UNIMORE	Final revision

This Project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement: 688860

Table of contents

List of figures

Glossary

Item	Description
ECU	Electronic Control Unit
ELF	Executable and Linkable Format
EMF	Eclipse Modeling Framework
GA	Grant Agreement
HVAC	Heating, Ventilation and Air Conditioning
NoC	Network On Chip
OIL	OSEK Implementation Language
RTE	AUTOSAR Run Time Environment, a middleware generated from an AUTOSAR XML specification.
RTOS	Real-Time Operating System
SoC	System On Chip

1. EXECUTIVE SUMMARY

Deliverable D4.4 is a software package containing the final version of the lightweight RTOS for the reference platform, developed in Task 4.2 of the HERCULES project. This document aims at illustrating the code developed, in particular related to the ERIKA Enterprise v3 RTOS, and to the AUTOSAR RTE Generator developed in order to integrate MM's automotive use case.

The developed source code of the RTOS is publicly available on EVI's GitHub account [Erika3-code] and is also contained in the software package provided along with this document. The package also contains the demo provided to MM for easing the integration step of their own use-case. Due to confidentiality concerns, the source code of the RTE is not provided as part of the deliverable, but it is maintained on EVI's private servers and available to the European Commission and to the reviewers upon request.

2. INTRODUCTION

2.1. Objectives

As illustrated in the GA, task 4.2 of HERCULES *"focuses on designing and developing a lightweight RTOS based on ERIKA Enterprise for executing the most time-critical activities in a timely fashion and at a lower power consumption than using the Linux OS"*.

As part of this task, EVI carried out the design and development of two different software components:

- EVI finished the redesign of the **ERIKA Enterprise kernel** and ported the RTOS onto the reference platforms chosen in the HERCULES Project. Concerning this activity, the document in particular summarizes the characteristics of the new version of ERIKA Enterprise, and the hardware platforms on which it has already been ported. Most of the technical information (e.g. how to install and use the software) is already available on public webpages [Erika3-nvidia, Erika3-xilinx].

- EVI designed and developed an **AUTOSAR Run Time Environment (RTE) generator** to be able of integrating the code of the pilot coming from MM's automotive use case. Note that, as further explained in Section 4, this activity was not originally planned in the GA, and it has been needed for properly supporting the use-case by partner MM.

3. ERIKA ENTERPRISE 3

ERIKA Enterprise [Erika] is a RTOS designed and certified for the automotive domain (i.e., AUTOSAR standard and OSEK/VDX certification), especially targeting multi-core devices. Such RTOS, developed by partner EVI in the course of several years, is released as Open-Source software on a GitHub repository [Erika3-code].

ERIKA Enterprise has been already used in production by various companies operating in the automotive and white goods market. Among those, we can cite Magneti Marelli, Piaggio & C Spa, Ariston, using the kernel on applications like Diesel and gasoline injection, robotic gears shifts, electric bikes, and HVAC systems.
The kernel has been also used (thanks to its open-source nature) by various other European research projects (including AMALTHEA, ARAMIS, eDAS, PROXIMA, SAFURE, P-SOCRATES, ENABLE-S3 and various others) as an environment for testing research results in the automotive market.

The original multi-core support in ERIKA Enterprise was designed for hardware architectures that did not have a uniform memory region, such as Janus [Fer00]. In those architectures, each core had its own local memory, and, most importantly, the view of the memory as seen by the various cores was different (i.e., the same memory bank was available at a different address on each core). This imposed a custom copy (i.e. separate ELF file) of the RTOS for each core. Other architectures had a uniform memory space, but the visibility of some memory regions was prevented by the Network on Chip (NoC). On Altera Nios II, for example, addresses differentiated by only the 31st bit referred to the same physical address with or without caching. This, again, implied the need for separate images (in particular, you can refer to the work done during the FP6 project FRESCOR, D-EP7 [Frescor]). More modern architectures like Freescale PPC and Tricore AURIX allowed the possibility of single-ELF, but the previous multi-ELF scaled relatively well due to the small number of cores, reducing the need for single-ELF versions of the system.

This approach, however, showed its limits when dealing with many-core architectures such as the Kalray MPPA; the high number of cores, in fact, required to avoid code duplication to not waste memory. For this reason, in the P-SOCRATES FP7 project, partner EVI started a complete rewrite of the RTOS to have a single ELF image shared among multiple cores. The development of the new version of the RTOS, called “ERIKA Enterprise 3”, announced at Embedded World 2017 and through a new website [Erika3], has been finalized within the HERCULES project. In the next paragraphs, we will summarize the characteristics of the new version of the RTOS.

3.1 New features

New data structures

The initial version of ERIKA Enterprise used a set of global data structures (basically, C language arrays of scalars) allocated in RAM and Flash. Each core had its own copy of the data structures, with the same name. Data which was shared among the cores was defined and initialized in one core referred to as the *first/master* core. The other cores were called *slave* cores.

The main idea in the compilation process was that the master core was self-contained, and could be compiled and statically linked first. Afterwards, when compiling the slave cores, the locations of the shared data (allocated in the master core image) were appended to each core’s linker scripts (see also [Fer00]). Figure 1 shows the structure of the two ELF files, highlighting the first core (master), which had everything defined, and the subsequent slave cores, which had the shared symbols addresses appended in the linker script.

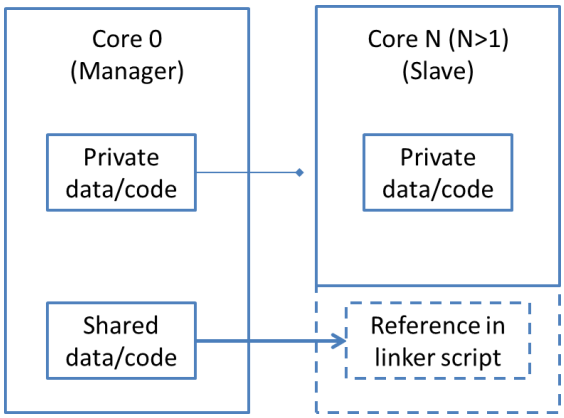

Figure 1: Structure of the multicore images in the original ERIKA Enterprise structure.

The new many-core support required instead that all the compilation process should be resolved in a single statically linked image (i.e., "single-ELF" approach) and therefore required a complete restructuring of the binary image. In particular, the main change has been related to a redesign of the data structures, which are now designed to separate information, letting the cores access relevant per-CPU data using an additional indirection.

The main guidelines used when designing the data structures follow:

- All data are shared among all cores.
- The code must be able to know on which core it is running. This is typically done using a special register of the architecture that tells the CPU number.
- Given the CPU number, it is possible to access "private" data structures for each core (see Figure 2). Note that those "private" data structures can be allocated in special memory regions "near" to each core (for example, they could be allocated in sections which can be pinned to per-core caches).
- Clear distinction between Flash Descriptor Blocks (named *DB) and RAM Control Blocks (named *CB). This way, the reader can have a clear idea of the kind of content just by looking at the name of the data structure.

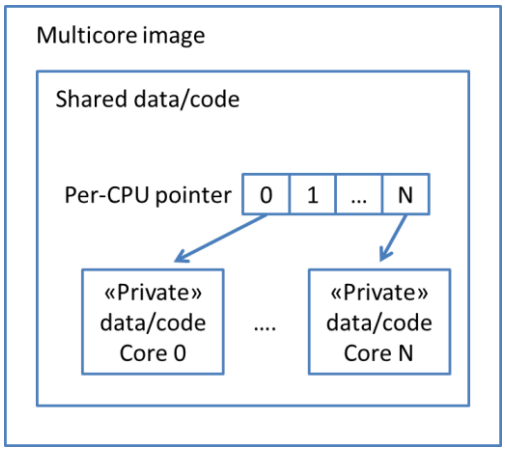

Figure 2: Structure of the Single-ELF image produced by ERIKA Enterprise.

IRQ handlers as tasks

The original version of ERIKA handled interrupts in the most efficient way in the case of no memory protection among tasks. The AUTOSAR standard, however, has introduced interrupt-service routines that run in application space (i.e. ISR2). The new version of ERIKA, therefore, has introduced a special fast handler called upon IRQ arrival, with the duty of activating the "interrupt task". Handling these ISRs as special tasks has also brought the advantage of simplifying the codebase by allowing simpler context change primitives.

File Hierarchy

For the new version of ERIKA we have adopted a new file hierarchy which aims to a simplify the codebase. In particular, these are the main changes of the new codebase:

- In the old version, CPU (the specific instruction set, such as PPC, Cortex-MX, …), MCU (the peripherals available on a specific part number), Boards (code related to the connections on the PCB) were stored in directories under the "pkg" directory. With the growing number of supported architectures, this became a limitation which also made the compilation process longer. The new version of the codebase includes MCUs and Boards under the CPU layer, making the dependencies in the codebase clearer.
- We have adopted a local self-contained flat (single directory) project structure instead of a complex hierarchy. All needed files are copied once in the project directory at compilation time, leading to simpler makefiles. Moreover, each project can, once created, be moved easily as it does not retain any dependency with the main ERIKA Enterprise source tree.
- We have maintained the RTOS code separated from the Application configuration. This is very useful to allow the deployment of pre-compiled libraries, moreover it allows partial compilation of the code.
- Files have been written in a way to allow the generation of per-project documentation using Doxygen.

RT-Druid code generator

For the new version of ERIKA we also completely rewrote the RT-Druid code generator. This part is described for completeness together with the RTE generator part in the next Chapter.

3.2 Supported platforms

In the HERCULES project, the new version of the ERIKA Enterprise RTOS has been ported on the following hardware platforms:

- Cortex-A57 of Nvidia Jetson TX1 and TX2 [Jetson]
- Cortex-A53 of Xilinx Ultrascale+ [Xil]

For the Nvidia platforms, partner EVI has been also in charge of porting the hypervisor on the specific SoC, releasing the support to the public domain [JailEvi]. The support developed by EVI has been already integrated into the mainline Jailhouse [JailTX1, JailTX2].

On both platforms, the RTOS has been ported on top of the Jailhouse hypervisor [Jailhouse]. For the second one, EVI exploited the same Ultrascale+ board [AX-Board] resulting from the AXIOM H2020 project (EVI was also a member of the AXIOM consortium) for a cross-project exploitation.

To ease the exploitation activities, partner EVI also released:

- A YouTube video showing a demo of ERIKA Enterprise on the Nvidia TX1 platform [YouTX1].

- A VirtualBox [VBox] virtual machine containing the ERIKA Enterprise RTOS and the RT-Druid development environment already installed for the TX1 and TX2 platforms. Such virtual machine is publicly available for download [Erika3-VM].

3.3 Usage

The make process is divided into the following phases:

1. The RT-Druid code generator generates the erika/pull and out/ directories.
2. The erika/pull/ directory contains a makefile with all the information needed to copy the kernel source code inside the inc/ and src/ directories.
3. Once copied, the (local) compilation process can start, producing the compiled library, and afterwards the binary image which is put inside the subdirectory out/.

The same compilation process can be configured from the Eclipse graphical environment.

In-depth instructions for using ERIKA on the HERCULES reference platforms are available on the following public pages:

- Nvidia Jetson TX1 and TX2:
  http://www.erika-enterprise.com/wiki/index.php?title=Nvidia_Jetson_TX1_and_TX2
- Xilinx Ultrascale+:
  http://www.erika-enterprise.com/wiki/index.php?title=Xilinx_ZCU102

The source code of the RTOS is available on Evidence's GitHub public account [Erika3-code] and also in the erika-rtos/ directory contained in the software package provided along with this document.

4. RT-DRUID AND THE AUTOSAR RTE GENERATOR

4.1 Introduction

During the analysis of the requirements of the Automotive use case, it became clear that the development process of partner MM was highly depending on the AUTOSAR infrastructure. In particular, there is a direct dependency on the usage of the software component model and of the AUTOSAR Run Time Environment (RTE), in the sense that the software of the use case is developed by MM as a set of AUTOSAR Software components that have to be compiled and integrated in the HERCULES Software Framework. On the other hand, partner EVI was, at the beginning of the project, only in charge of activities related to porting and customization of the RTOS, without including any RTE generator. This indeed was an issue, because there was no way to connect the software components (described in AUTOSAR XML format) to the RTOS primitives

Given that, the HERCULES consortium had two possibilities:

1. Require partner MM to provide an integration by hand of their software components into the ERIKA Enterprise RTOS developed by partner EVI;
2. Require partner EVI to somehow provide means for connecting the components to the kernel, possibly using an ad-hoc tool.

After the analysis of the pros and cons, the consortium decided to ask partner EVI to develop a custom tool that would be used to connect the Software components provided by MM to the ERIKA Enterprise v3 operating system. The subset of the RTE specification has been selected in a way to limit the additional effort to the minimum. The development activity has been inserted as part of Task 4.2 and is described in this document.

The development of an additional tool has indeed another positive aspect: it allows the consortium to optimize the AUTOSAR Runtime Environment for the specific case of HERCULES. In fact, this allows the consortium to fine tune:

- The communication between the software components (with the related data exchange, and the tuning on the memory bandwidth required to have predictability, see Deliverable D3.4);
- The communication between the various hypervisor's guests, that in this case could be directly handled in a transparent way by the code generator;
- The communication with the external world (in particular by directly wrapping the CAN and Ethernet protocols on the board).

All this happened at the same time of the demise of the OSEK/VDX consortium (in fact, as of today, the website http://www.osek-vdx.org is no longer available), which caused on the other hand commercial issues to the partner EVI (i.e. the OSEK/VDX certification of ERIKA Enterprise v2 is less valuable once the standard has disappeared).

Given those advantages, and given the disappearing status of the OSEK/VDX standard, partner EVI decided to start the development of an AUTOSAR RTE Generator. In order to do that, EVI started from the Artop plugins [Artop], a set of Eclipse plugins that provide updated support for the AUTOSAR XML metamodel, adding then a code generator written in Acceleo [Acc], which is then described in the following paragraphs.

As a final note, we would like to note that the usage of Artop required EVI to participate to the AUTOSAR consortium. For this reason, since May 2017, EVI has become an official AUTOSAR Development member, joining the forces and providing its expertise in the newly created working group on AUTOSAR Multicores. Partner EVI believes that this is a great opportunity for growth, which is exposing the company to the European level, and which allows to provide a set of AUTOSAR officially-compliant components. Finally, it provides an opportunity for the HERCULES consortium to influence the existing AUTOSAR standard, as it will

be possible for partner EVI to propose to the AUTOSAR community some enhancements and ideas coming from the project.

4.2 RT-Druid Description

ERIKA Enterprise has been always distributed in bundle with a configuration tool able of converting the configuration files (based on the OIL format, and, more recently, on the AUTOSAR XML format) into kernel data structures used to build the RTOS.

In particular, the old version of RT-Druid had an internal data structure which was inherited from a previous version of the tool (which was designed in year 2004 to provide a schedulability analyzer), and adapted to perform code generation. As a result, the data structure used was rather complex, and the various per-CPU-type generators were written mainly in Java language, making them difficult to be maintained, with large code duplications.

With the complete rewrite of the codebase, it became clear the opportunity to restart from scratch with a better design. In particular, EVI started a complete redesign of the RT-Druid tool, which has now the following features:

1. It is completely written using Ecore [EMF], XText [Xtext] and Acceleo [Acc], which are the standard Eclipse plugins used to implement model-to-model and model-to-text converters and code generators, as well as advanced text editors.
2. It allows the separation of the engine responsible for displaying, reading and generating code, from the actual rules used in the code generation. As a result, the configuration code generator can be completely written in Acceleo directly by the kernel developers, cutting down the complexity of the maintenance of the code generation infrastructure.
3. It allows the separation of the “OS configuration” from the “Application configuration”. This is quite important in large setups, because in this way the kernel can be compiled in a library that can be afterwards moved or reused, while the application code reuses the available library.

Concerning the second point, the tool is composed by some parts that do not depend on the specific release of ERIKA v3 and some others that are strictly related to a specific release of ERIKA v3. As example, the capability to read and edit a file, written using the OIL standard, is not related to a specific release of ERIKA v3; on the other hand, the list of parameters needed to configure ERIKA v3 and the content of generated source files may change between two distinct release of ERIKA v3. Thus, the “independent” part can follow its own development schedule, while the ERIKA-related part will now be included as part of ERIKA v3, and modified as well by the ERIKA v3 developers independently from the RT-Druid developers.

Figure 3 shows all components of RT-Druid, dividing graphically the parts which are dependent from ERIKA v3 and the parts which are independent:

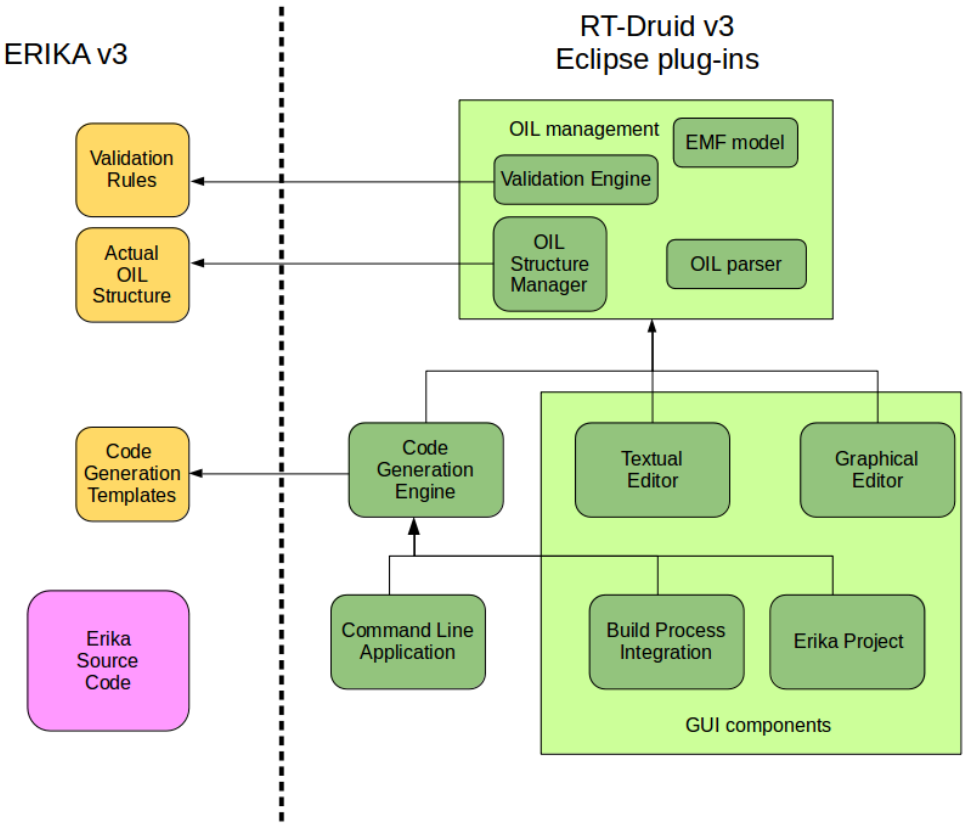

Figure 3: Internal structure of the RT-Druid Code Generator.

The most important component is the one able to contain all data: the *EMF model*. As the name says, it is based on the Eclipse Modeling Framework [EMF], and used by all the other RT-Druid components.
To be able to read, write and validate an OIL file, RT-Druid contains an *OIL parser* and a *validation engine*.
Then, considering the OIL file format, we can see that an OIL file is composed of two parts: an Implementation Definition and an Application definition:

- The Implementation Definition is in charge to define all valid parameters and their properties. It can be thought as a type declaration where, for example, we declare that a task has an attribute “priority”. As it can be easily imagined, this part declaring the attributes existing into a project is strictly related to a specific version of ERIKA v3, and for that reason it is distributed with ERIKA v3 (in the component *Actual OIL Structure*). This part is handled in RT-Druid by the *OIL Structure Manager* component.
- The Application Definition then contains the equivalent of a variable definition, which is at the end the actual value of each parameter defined in the Implementation Definition (following the example above, a Task named task10ms has priority 2). The Application Definition must be provided by each project in order to configure the kernel.

The *Validation Engine* performs both syntactic and semantic checks. As an example, it checks that the Actual OIL Structure declares a parameter provided by the user, and that its value is compliant with all declared parameter properties. A more complex example is: “check if the total amount of required memory can fit on a specific board”. Again, some of the Validation Rules are strictly related to a specific release of ERIKA v3.

The most visible part of RT-Druid are all GUI components that allow the user to edit the OIL file and to integrate the configuration of ERIKA v3 with the classical Eclipse build process. Currently, these components are mainly:

- Specialized text editors (for editing OIL text). These editors provide line numbering, and automatic completion of the OIL parameters.
- Special options in the properties menu.
- Special graphical editors.

A special note has to be done on the graphical editors. These editors are generated using the EFORMS framework developed by EVI (before the HERCULES Project) [EFORMS]. Basically, the graphical editors are created automatically starting from a description composed of the following files:

- a data structure description;
- a text description of the graphical representation of the widgets on the panel;
- a description of the verification and consistency rules;
- a description of the configuration code to be generated (in Acceleo).

This allows a fast implementation of graphical plugins that otherwise would have required months to be implemented.

Finally, the *Code Generation Engine* is based on a set of templates that may change between different release of ERIKA v3.

4.3 Command line execution of the code generator

To allow the integration of RT-Druid on a production tool-chain, there is also a component to execute RT-Druid from command line to generate ERIKA v3 configuration source code.

In this way, it is possible to execute the same build commands issued from the graphical Eclipse interface in a command line, useful to script the complete build process.

4.4 RTE Generator

The RTE generator is being developed as part of the HERCULES Project by EVI, and it will support a subset of the RTE specification, enough to allow partner MM to port their application on top of the HERCULES software stack.

The main features to be implemented were agreed with partner MM in a meeting held in Pisa in April 2016 as follows (“H” means high priority while “L” stands for lower priority):

- (H) The RTE will take advantage of Artop, provided by MM.
- (L) MISRA Compliancy. Code generated by the RTE does not need to be MISRA compliant.
- (H) The RTE generator will generate code from an AUTOSAR XML produced by DaVinci Vector.
- (H) The RTE generator will be launched either from command line or from the GUI (the more convenient for EVI to implement).
- (H) The RTE subset implemented will include inter- and intra- task communication. Inter-partition communication will not be supported.
- (L) Inter-ECU communication will be supported in a custom way since it will involve the hypervisor and the external Tricore architecture. Actually, the RTE exposes a communication API based on the AUTOSAR standard. Then, a custom Linux daemon is in charge of forwarding the messages from/to the external AURIX board.

- (H) The RTE will support the implicit communication.
- (L) Explicit communication (using send and receive functionalities) will be handled, but without queuing support.
- (L) The RTE will not support communication timeouts. The final version may support it, to be decided.
- (H) The mapping of runnables to tasks will be provided in the AUTOSAR XML.
- (L) The RTE code generated will support C++ 11.
    - MM will provide an example including the C++ functionalities needed.
- (H) The RTE generator will provide "reasonable" error checking, for the most trivial checks.
- (L) Complex consistency checks will be discussed in a second moment.
- (H) No support for Measurement and Calibration and for A2L files. Everything is put in the init procedure.
- (L) EVI will provide for each release the list of RTE requirements that will be supported.

More in-depth requirements have been agreed during a few face-to-face meetings organized at MM's premises in Venaria.

The Communication between different cores on the Jailhouse hypervisor has been implemented by the means of a library developed by partner EVI in the context of the RETINA EUROSTARS project [Retina]. This library provides a subset of the AUTOSAR COM standard API, so the RTE supports the Inter-ECU communication following the AUTOSAR standards.

We now have a version of the RTE working on the Jetson TX2 platform. In the next weeks, partner EVI will coordinate with partner MM for checking the very last missing features (e.g., endianness, padding, multicast, C++) before starting the integration of its use-case.

4.5 Proof of concept (PoC)

The proof of concept of EVI's RTE has been shown through a video uploaded onto the YouTube platform [YouRTE]. The video shows the complete compilation path for an AUTOSAR application composed of two runnables (see Figure 4).

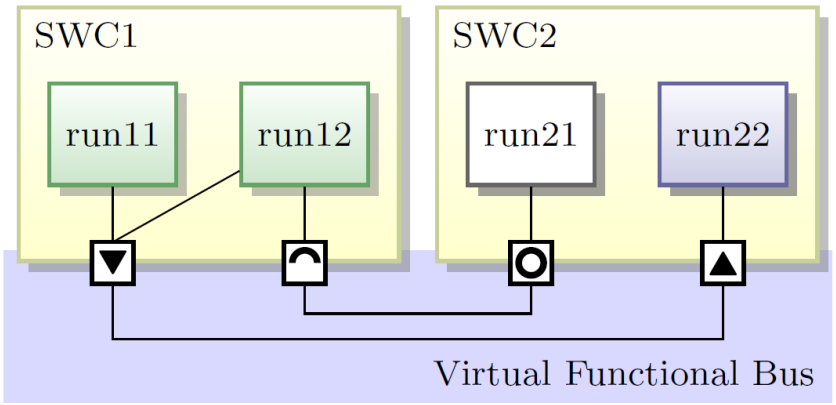

Figure 4: The structure of the AUTOSAR PoC generated with the RT-Druid RTE generator.

The following settings have been applied to the system:

- Runnable run11 is triggered by a timing event every 100ms.
- Runnable run12 is triggered by another timing event every 50ms.
- Runnable run22 is triggered by a timing event every 50ms.
- Runnable run21 is triggered by the invocation of a server call at the server port.

- Runnable run11 and run12 are mapped to Task1.
- Runnable run22 is mapped to the Task2.
- Runnable run21 is not mapped to any task and it can be invoked concurrently.

In the video, we built and run an AUTOSAR system starting from the runnables and the AUTOSAR XML description already created, through the following steps:

- We used Artop to edit/view the AUTOSAR file;
- We used the RT-Druid RTE generator to generate the RTE code;
- We use the RT-Druid code generator to generate the ERIKA Enterprise v3 configuration code;
- After that, the system is compiled using the GCC cross-compiler for AVR;
- The binary image is then flashed into the Arduino Uno board using a command line tool;
- The application finally runs in the Arduino Uno board, basically sending a set of messages to the serial console in correspondence of the execution of the various runnables (see Figure 5).

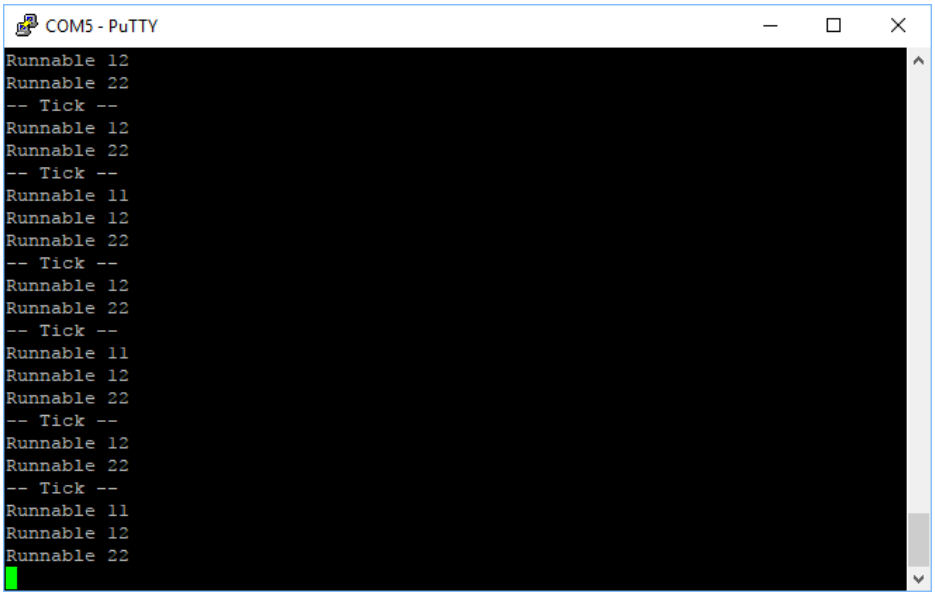

Figure 5: Running the AUTOSAR PoC on the Arduino Uno.

4.6 Demo

To help partner MM to get familiar with RT-Druid and especially with the RTE generator, partner EVI has developed a simple demo for the Nvidia TX2 board. The demo, shortly described by the following diagram, implements a simple "loop-back" mechanism. It consists of the following actors:

- An application running on Linux, generating an integer value sent to the first ERIKA core and showing the value read back from such core.
- The first ERIKA core, running 2 Software Components (SWC), forwarding data from Linux to the other core running ERIKA and the other way around.
- The second ERIKA core, running an SWC which sends data back to the first ERIKA core.

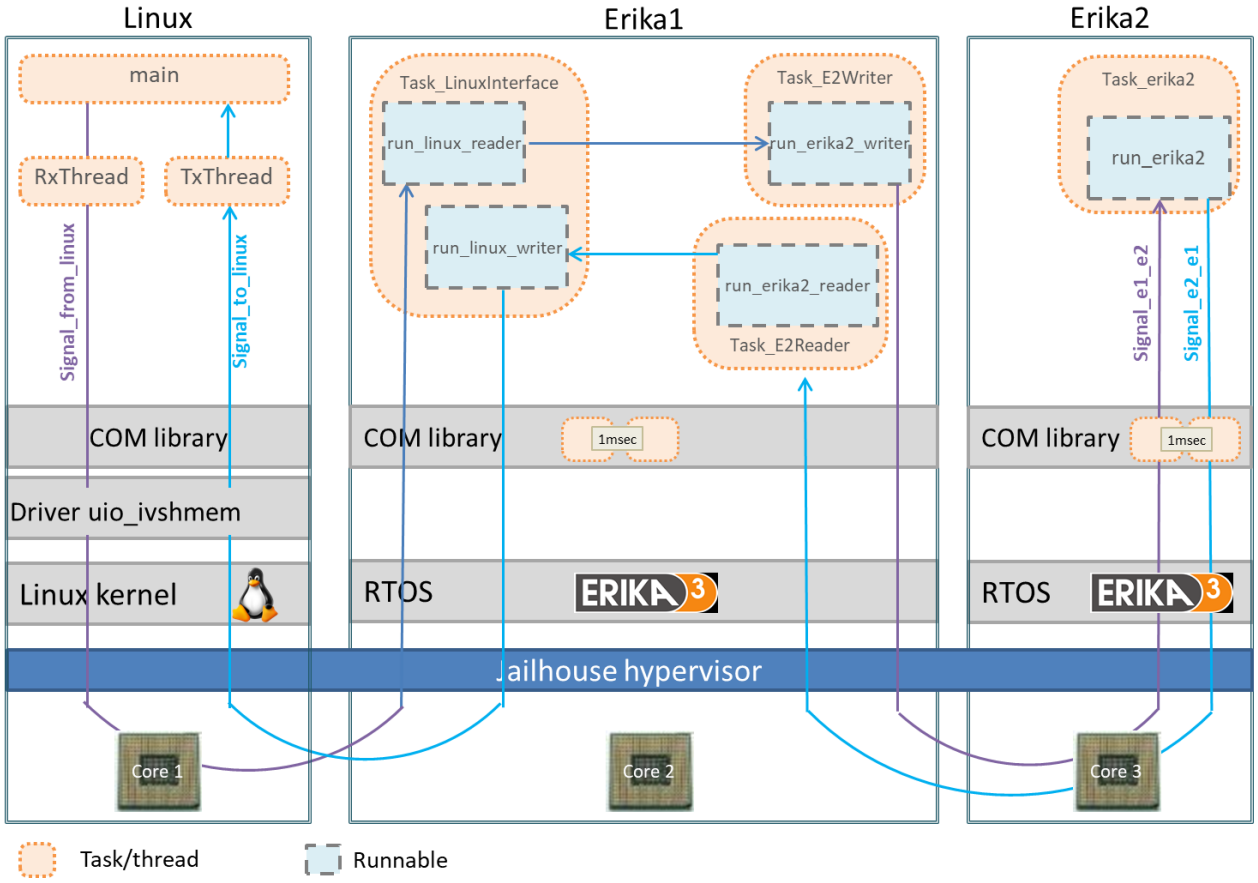

Figure 6: Diagram of the RTE demo.

This demo stresses both:

- the communication between Software Components in the same core (also called intra-ECU communication);
- the communication between Software Components on different cores (also called inter-ECU communication).

In fact, we can see the integer moving from the Linux application to the Erika2 core, and then back to Linux. In both directions, the Erika1 core acts as a router, forwarding data.

The actual Round Trip Time (RTT) is strictly related to the scheduling of each component. In the demo, we have used the following settings:

- as described by AUTOSAR COM standard, the COM library has 2 periodic tasks that send and receive data with a period of 1ms;
- all Software Components are event-driven, i.e. they are activated when the waited data is available.

The resulting RTT is mostly affected by the COM tasks period.

The steps for generating the code for the TX2 platform starting from the given ARXML files are summarized in Figure 7. The ARXML files and the PowerPoint presentation describing all the needed steps that EVI has given to MM are contained within the demo-rte-tx2/ directory in the software package provided along with this document. Please, refer to the PowerPoint presentation for in-depth information about the steps and the generated files.

The demo has been provided within a VirtualBox [VBox] virtual machine with all the development tools already installed, and it is expected to ease and shorten the integration activity of the use-case by MM.

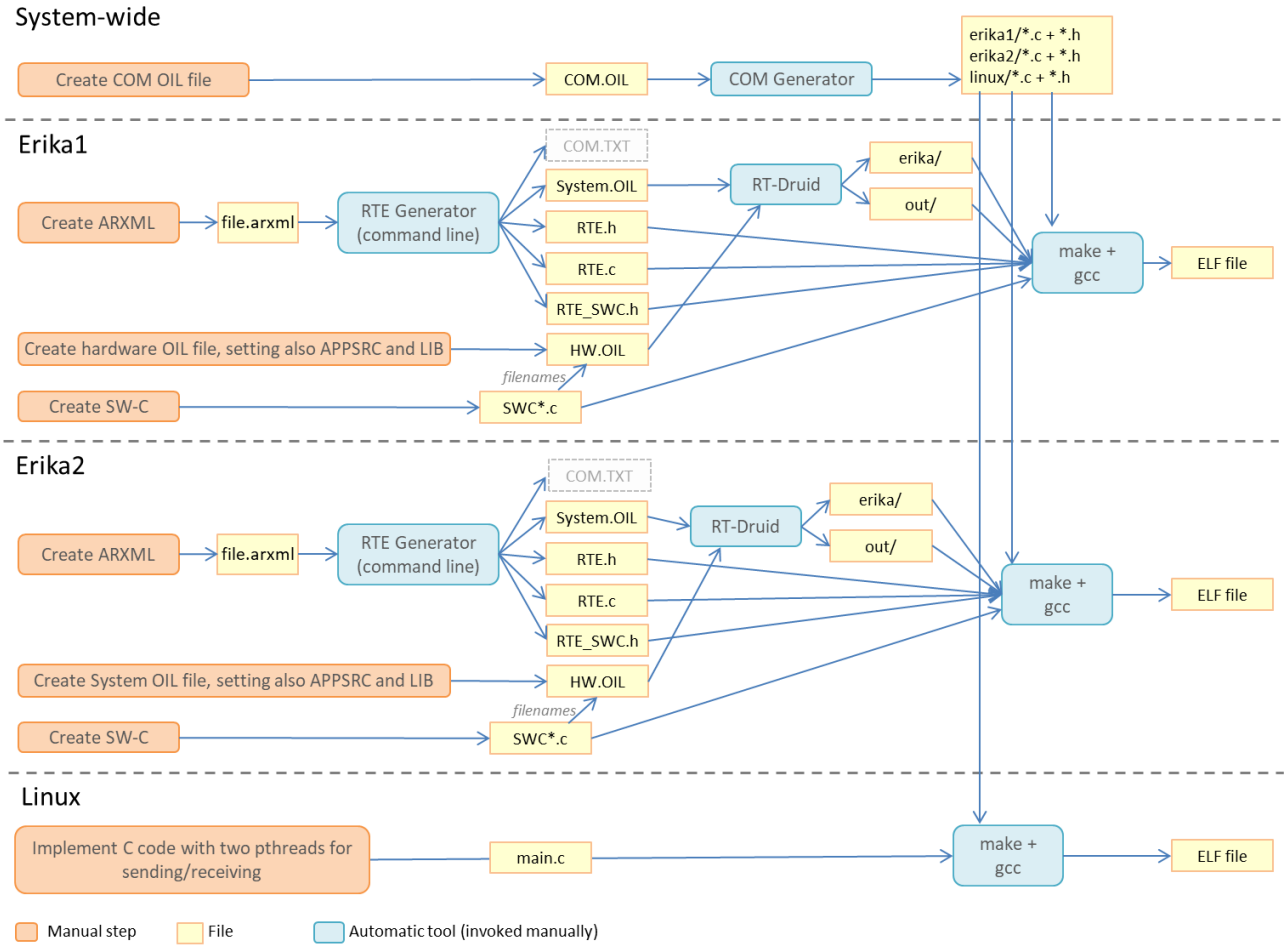

Figure 7: Generated files for the RTE Demo.

5. CONCLUSIONS

This document illustrated the main features of the new version of the ERIKA Enterprise RTOS that has been finalized in HERCULES, the hardware platforms supported and the AUTOSAR RTE generator. The work carried out has been promoted through websites [Erika3] and YouTube videos [YouTX1] and [YouRTE]. Part of the work done (i.e., ERIKA Enterprise, the Jailhouse hypervisor) has been publicly merged in the respective open-source projects.

6. REFERENCES

[Acc] Eclipse community, Acceleo, https://www.eclipse.org/acceleo/

[Artop] AUTOSAR Tool Platform (Artop), http://www.artop.org

[AX-Board] The AXIOM board, http://www.axiom-project.eu/2017/02/the-axiom-board-has-arrived/

[EEReq] Evidence, ERIKA Enterprise version 3 requirement document, available on the ERIKA Enterprise website, http://erika.tuxfamily.org/drupal/content/erika-enterprise-3

[EFORMS] EFORMS Framework for fast implementation of graphical plugins in Eclipse. http://www.evidence.eu.com/products/eforms.html

[EMF] Eclipse Modeling Framework (EMF), https://eclipse.org/modeling/emf/

[Erika] Evidence Srl, ERIKA Enterpise, http://erika.tuxfamily.org

[Erika3] Evidence Srl, ERIKA Enterprise 3, http://www.erika-enterprise.com/

[Erika3-code] Evidence Srl, ERIKA Enterprise 3 source code, https://github.com/evidence/erika3

[Erika3-nvidia] Evidence Srl, ERIKA Enterprise 3 for NVIDIA Jetson TX1 and TX2, http://www.erika-enterprise.com/wiki/index.php?title=Nvidia_Jetson_TX1_and_TX2

[Erika3-VM] Evidence Srl, VirtualBox virtual machine for TX1 and TX2, http://www.erika-enterprise.com/index.php/download/virtual-machines.html

[Erika3-xilinx] Evidence Srl, ERIKA Enterprise 3 for Xilinx ZCU102, http://www.erika-enterprise.com/wiki/index.php?title=Xilinx_ZCU102

[Fer00] A. Ferrari, S. Garue, M Peri, S. Pezzini, L.Valsecchi, F. Andretta, and W. Nesci, *"The design and implementation of a dual-core platform for power-train systems"*. In Convergence 2000, Detroit (MI), USA, October 2000.

[Frescor] FRESCOR FP6 D-EP7v2, available at http://www.frescor.org/ and also http://erika.tuxfamily.org/wiki/index.php?title=Altera_Nios_II

[JailEvi] Evidence Srl, Jailhouse for default Linux kernel of Nvidia Jetson platforms, https://github.com/evidence/linux-jailhouse-jetson

[Jailhouse] Siemens, Jailhouse hypervisor, https://github.com/siemens/jailhouse

[JailTX1] Evidence Srl, Jailhouse support for Nvidia Jetson TX1, https://github.com/siemens/jailhouse/blob/master/configs/arm64/jetson-tx1.c

[JailTX2] Evidence Srl, Jailhouse support for Nvidia Jetson TX2, https://github.com/siemens/jailhouse/blob/master/configs/arm64/jetson-tx2.c

[Jeston] Nvidia Jetson, http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html

[PSOC] P-Socrates EU project, http://www.p-socrates.eu/

[Retina] RETINA EUROSTARS project, http://www.retinaproject.eu/

[VBox] Oracle VirtualBox, https://www.virtualbox.org/

[Xil] Xilinx, Ultrascale architecture, https://www.xilinx.com/products/technology/ultrascale.html

[XText] Eclipse community, XText, https://eclipse.org/Xtext/

[YouRTE] Evidence Srl, *Compiling an AUTOSAR application using RT-Druid RTE Generator and ERIKA Enterprise 3*, https://youtu.be/SXK8aiEG-04

[YouTX1] Evidence Srl, *Multi-OS Demo on Nvidia Jetson TX1 with JailHouse Hypervisor, ERIKA Enterprise 3 and Linux*, https://www.youtube.com/watch?v=skIcAkXfNWQ