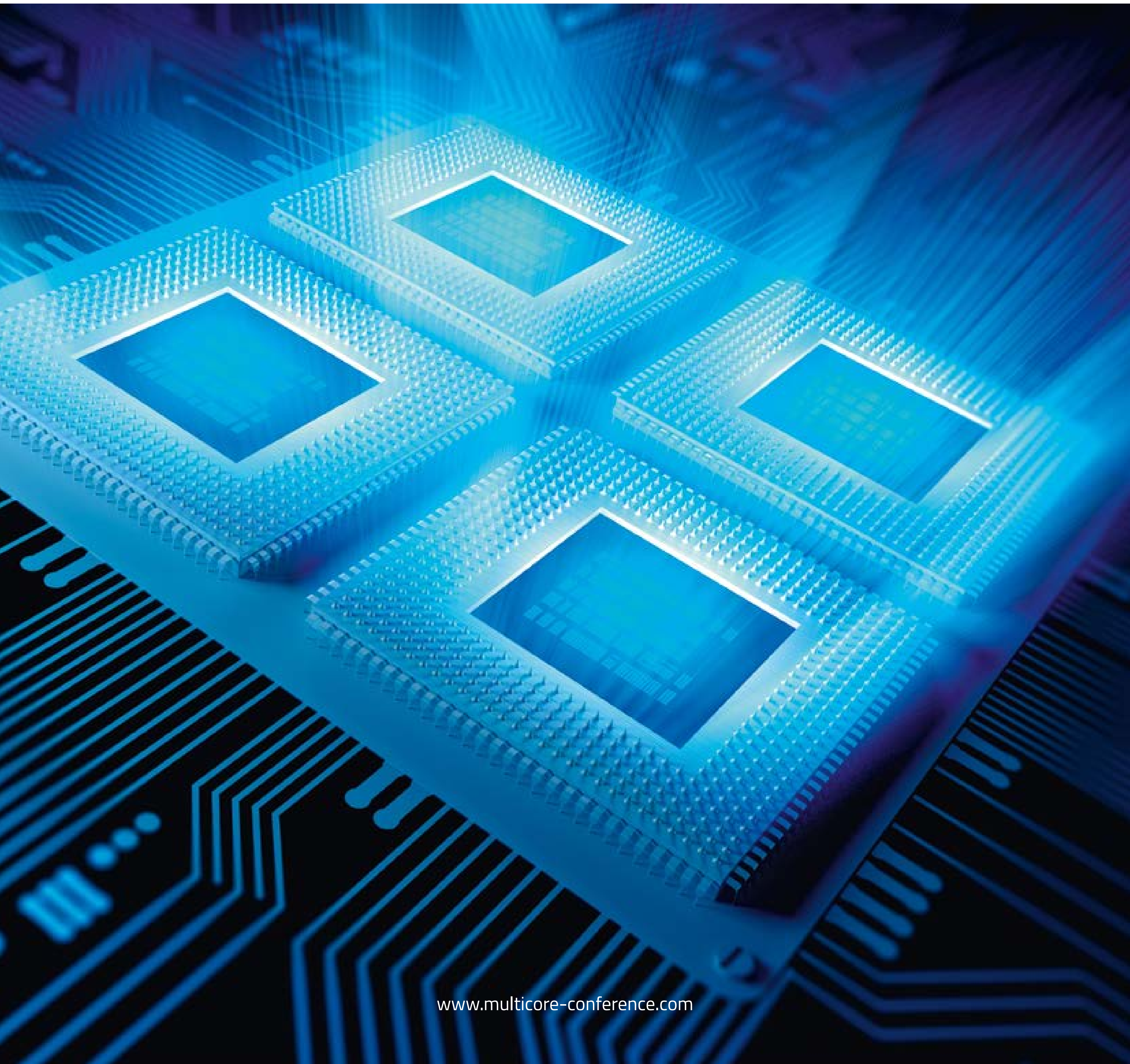
Embedded Multi-Core Conference

EMCC

Meet the experts on multi-core development



Speed up your automotive control unit – a synchronized tool chain for efficient multi-core development

There is a simple market myth for Electronic Control Units (ECUs) in the automotive market: multi-core processors offer better performance than single-core ECUs. Does this always hold true? The answer is no! Major efforts can be required to get the processor performance actually on the road, considering the peculiarities of hard real time software and the usual constraints, e.g., the overall memory consumption of the application.
This white paper describes the workflow and tool chain for optimal software component distribution, combined with an iterative cyclic engineering process.

Figure 1 illustrates the interaction of three partner tools:

- TA Tool Suite
- EB tresos Studio
- iSYSTEM winIDEA

The goal is to provide software developer and integrator with an integrated solution for perfect software mapping on an Infineon AURIX multi-core microcontroller.

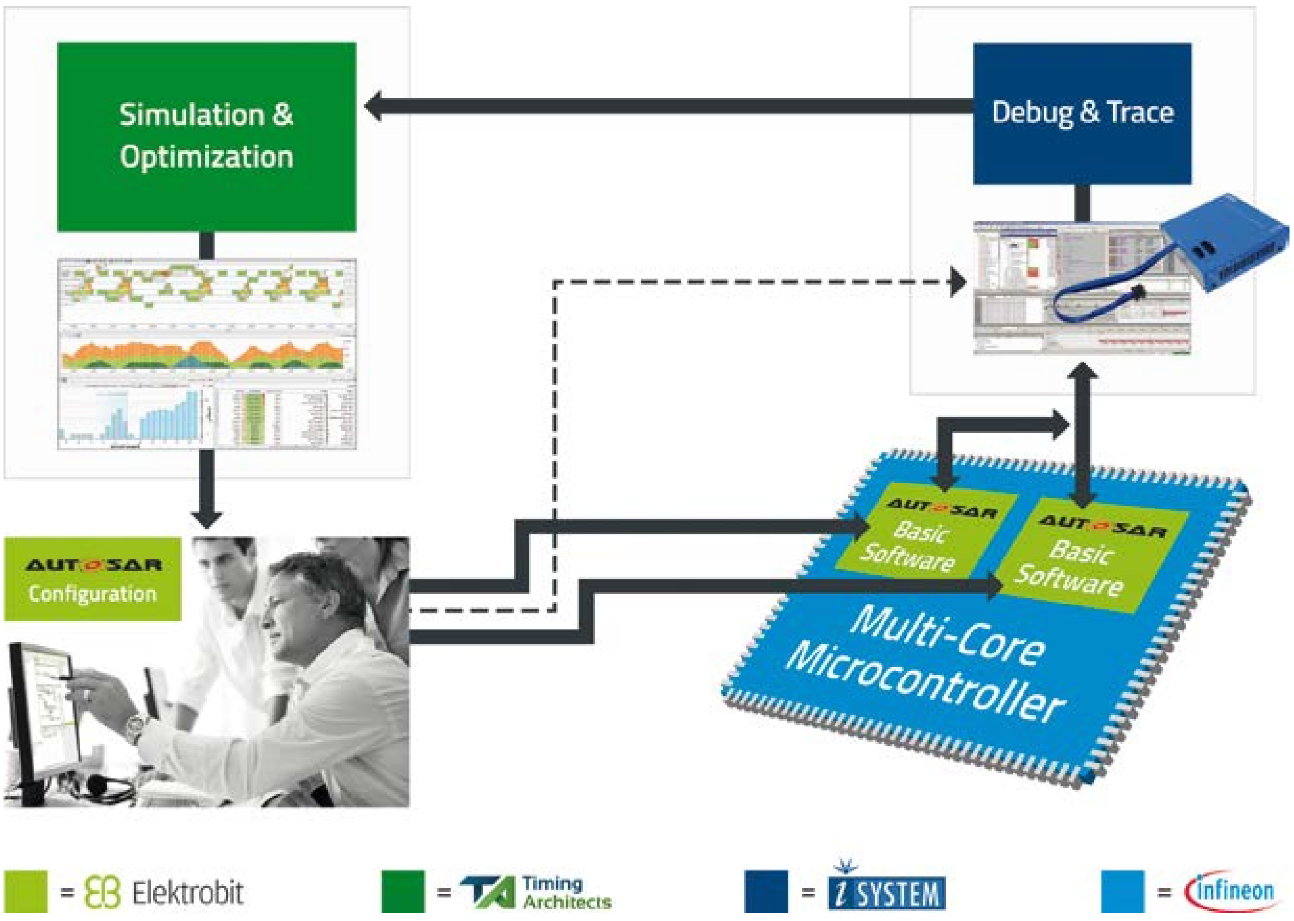


Figure 1: Tool chain: configuration, debug & trace, simulation & optimization of ECU software on a multi-core microcontroller

This article is structured in four chapters. Timing-Architects (TA), Elektrobit (EB), and iSYSTEM describe the tools' capabilities and the interfaces to the partner tools. Infineon Technologies as chip manufacturer provides a detailed insight in the memory usage.

TA Tool Suite: Automated optimization and validation

A key challenge in the development of embedded multi-core systems is the distribution of multiple software components to the available processor cores. Data exchange between components on different cores requires significantly more computational resources than the same operation would require on a single core, i.e. inter-core operations are more expensive than intra-core operations.

To achieve optimal system performance, it is important to allocate an application in a way that minimizes the overhead generated by inter-core communication. This is a challenging task, in particular for systems that were not designed for multi-core in the first place, e.g. legacy applications for single-core microcontrollers.

Timing-Architects offers the TA Tool Suite, a collection of tools that facilitates the development of embedded multi-core systems.

To solve the problem, the TA Optimizer, one module of TA Tool Suite, provides a system design that minimizes inter-core communication overhead and optimizes real-time behavior and performance automatically. Based on a timing model of the application, the TA Optimizer computes promising system configurations. The timing model includes a hardware part, a software part, and an OS part. The most promising configuration results can be compared with respect to relevant metrics, such as average processor utilization or communication delay as shown in Figure 2. Thus, the user can determine the optimal system design.

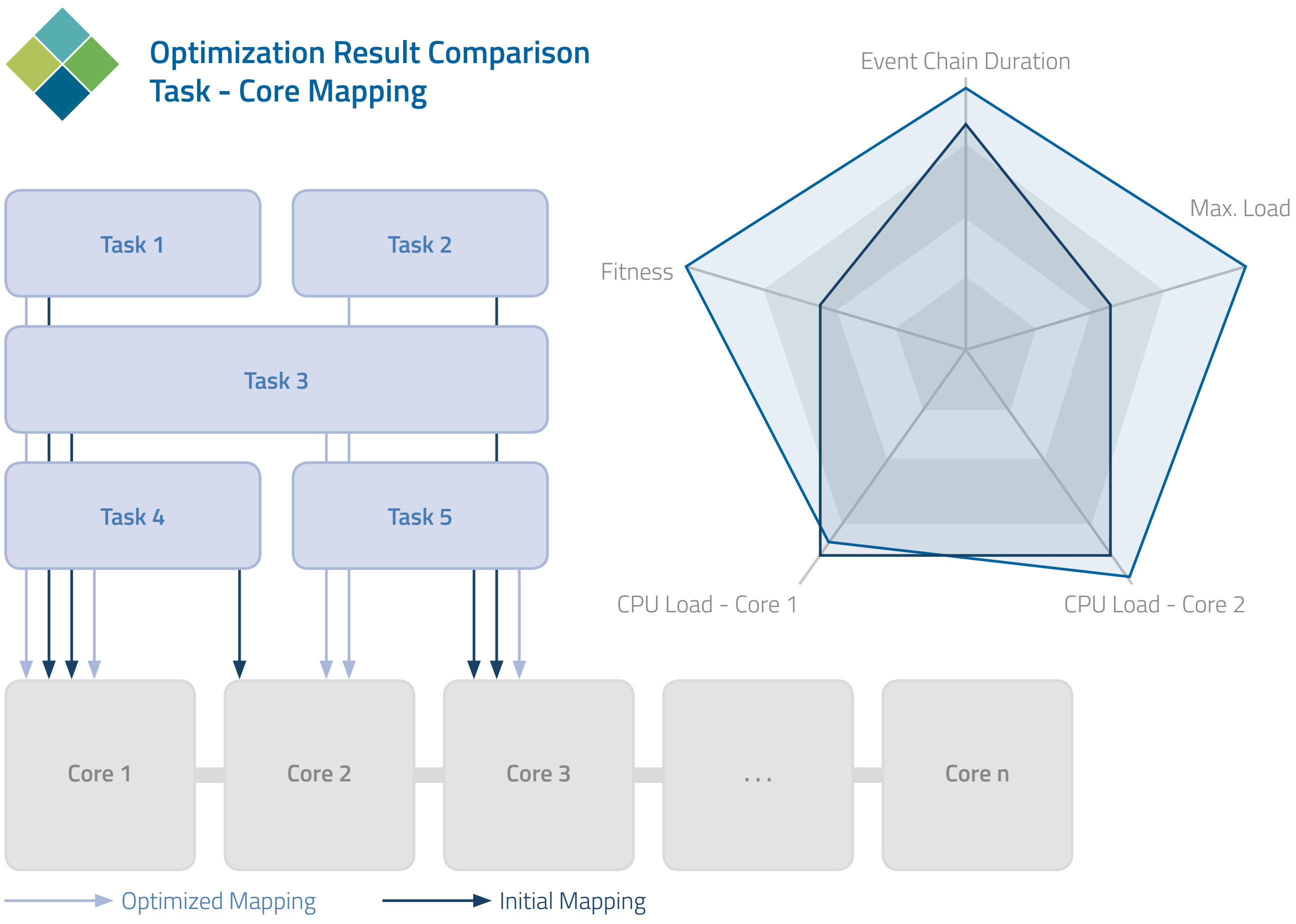


Figure 2: The TA Optimizer improves the design of embedded multi-core applications automatically, for example, by optimizing the task-to-core allocation. Based on a variety of real-time and performance metrics, the user can select the optimal configuration with respect to specified requirements.

The TA Optimizer uses the TA Simulator to calculate the metrics for a specific configuration.
First, the application is simulated, based on the timing model and the current configuration. Then, the metrics are calculated with respect to the simulated run-time behavior.

Simulation accuracy depends on the operating system and the hardware platform used by the application. For most accurate results, it is critical to consider OS and hardware-specific parameters for the simulation. Thanks to close collaboration with Elektrobit and Infineon, the TA Simulator can take account of the subtleties in the behavior of the Elektrobit operating systems and of the Infineon AURIX processor family.

Once an optimal application design has been created, the system can be tested on the actual target. Via the TA Tool Suite Export, the AUTOSAR system description and the ECU configuration can be exported and imported by EB tresos Studio to generate OS vendor-specific code.

The following section of this article covers the operating systems that are most suitable for usage in multi-core systems.

EB tresos software & tools: Operating systems and run-time environment (RTE) configuration

One of the challenges for the current and next generation of automotive embedded systems is efficient, safe, and secure use of the available cores on a single chip. Some years ago, selecting an operating system was quite simple for automotive ECUs. The choice was between an OSEK OS and an AUTOSAR OS.
Today there are multiple options.
For ECUs with low or no Automotive Safety Integrity Level (ASIL) requirements:
- Single-core AUTOSAR OS with simple scheduler
- Multiple instantiation of single-core AUTOSAR OS
- Multi-core AUTOSAR OS

For ECUs with high ASIL requirement or ECUs used as integration platform:
- One or multiple instantiations of single-core Safety OS (can be combined with options 1-3)
- Safety OS multi-core

The Safety OS and Safety OS multi-core are featured as SEooC[1], providing safety integrity functions like partitioning, context protection, stack protection, and monitored microkernel startup to support safety case argumentation for freedom from interference in spatial domain.

In addition to the obvious key criteria for OS selection, e.g., safety, performance, interaction with AUTOSAR RTE, and tool support for configuration, criteria like tracing, simulation, and optimization are getting more and more important due to the increased complexity of hardware and software: more cores, more functions, more interaction. A multiple-OS solution requires a core-to-core (C2C) communication module (Figure 3) which is not part of the AUTOSAR standard specification.

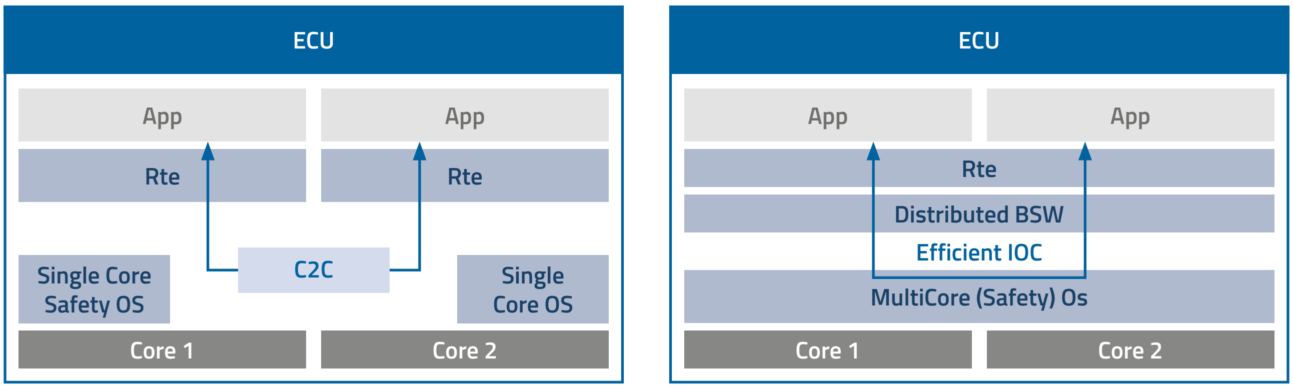


Figure 3: Basic architecture of different approaches: Multi-Core ECU with two Single Core operating systems (left side) vs. Multi-Core ECU with a Multi-Core Safety OS (right side)

[1]SEooC - Safety Element out of Context, a concept described in the ISO 26262 standard

The multi-core OS solution comes with a partitioned RTE which abstracts the inter-core communication from the application. The inter-core communication is delegated to the RTE and operating system that implements an efficient IOC[2]. The configuration of the communication is completely handled by the RTE based on the ECU system description.

The advantages of the multi-core OS approach include:

- No user-specific C2C module
- Generated inter-core communication
- Configuration support by wizard
- Reuse of existing tracing solutions (iSYSTEM) and optimization solutions (TA Tool Suite) without specific adaptation

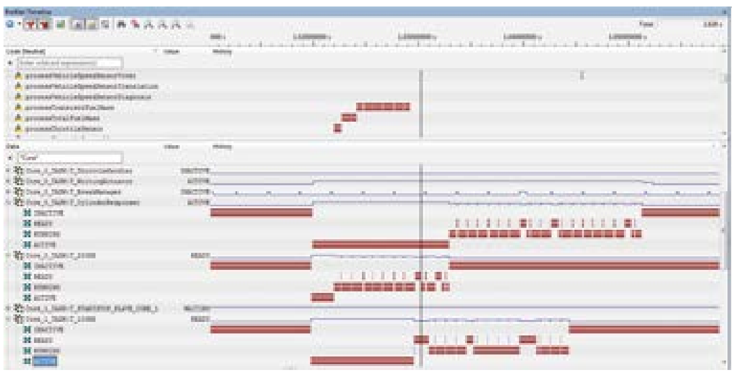
The interface between EB tresos Studio and the TA Tool Suite consists of two components.

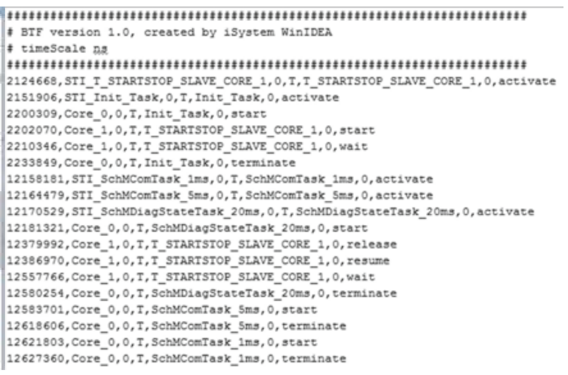
- The first component is the user visible part. The importer and the exporter of both tools need to be able to exchange data. The data exchange is based on the AUTOSAR formats but requires some adaptations to consider vendor-specific parameters.
- The second component of the interface is the model of the underlying operating system and RTE. For example, a task switch in an AUTOSAR OS has a different run-time compared to a task switch in a microkernel based Safety OS. This specific value has to be provided to TA´s simulator and optimizer tool in a specific format by EB´s operating system and RTE.

The interface between EB tresos and the iSYSTEM winIDEA functionality is based on the ORTI file for OS elements such as tasks and interrupts. To handle Runnables, we created a special EB tresos Studio plugin that provides Runnable information in a specific format, which can be used by iSYSTEM winIDEA to identify start and stop events of Runnables. In the future, standardized formats like AUTOSAR system description or RTE configuration will be used for tracing Runnables.

iSYSTEM winIDEA: Using trace for run-time analysis of an AUTOSAR application

The motivation of performing a trace recording on the actual ECU hardware is to verify the optimization results of the TA Tool Suite which are based on a software model simulation. For a meaningful correlation, the trace recording must provide the same type of run-time information as the simulation. In this case, the trace recording has to provide correct timing information about the status of every AUTOSAR OS task and Runnable execution on all processor cores.





```
##########################################################################
# BTF version 1.0, created by iSystem WinIDEA
# timeScale ns
##########################################################################
2124668,STI_T_STARTSTOP_SLAVE_CORE_1,0,T,T_STARTSTOP_SLAVE_CORE_1,0,activate
2151906,STI_Init_Task,0,T,Init_Task,0,activate
2200309,Core_0,0,T,Init_Task,0,start
2202070,Core_1,0,T,T_STARTSTOP_SLAVE_CORE_1,0,start
2210346,Core_1,0,T,T_STARTSTOP_SLAVE_CORE_1,0,wait
2233849,Core_0,0,T,Init_Task,0,terminate
12158181,STI_SchMComTask_1ms,0,T,SchMComTask_1ms,0,activate
12164479,STI_SchMComTask_5ms,0,T,SchMComTask_5ms,0,activate
12170529,STI_SchMDiagStateTask_20ms,0,T,SchMDiagStateTask_20ms,0,activate
12181321,Core_0,0,T,SchMDiagStateTask_20ms,0,start
12379992,Core_1,0,T,T_STARTSTOP_SLAVE_CORE_1,0,release
12386970,Core_1,0,T,T_STARTSTOP_SLAVE_CORE_1,0,resume
12557766,Core_1,0,T,T_STARTSTOP_SLAVE_CORE_1,0,wait
12580254,Core_0,0,T,SchMDiagStateTask_20ms,0,terminate
12583701,Core_0,0,T,SchMComTask_5ms,0,start
12618606,Core_0,0,T,SchMComTask_5ms,0,terminate
12621803,Core_0,0,T,SchMComTask_1ms,0,start
12627360,Core_0,0,T,SchMComTask_1ms,0,terminate
```

Figure 4: iSYSTEM multi-core AUTOSAR profiling and BTF file export

[2]IOC - Inter-OS Application Communication

Traditionally, originating from OSEK-times, profiling an AUTOSAR application is done based on the information provided by the ORTI file. The ORTI file contains all required information for OS profiling, i.e., run-time analysis on task level and on ISR level. However, it does not provide any information about Runnables as they are implemented above the RTE level of the AUTOSAR architecture. This level is typically beyond the scope of the ORTI file exporter utility of the AUTOSAR tool.

Typically, only the currently running task is recorded during a trace run, while the detailed state of each task is not recorded, i.e., whether the task is activated, running, or suspended. However, metrics such as task activation delays are important for the overall system performance analysis.

Thus, the first challenge for a trace tool is detailed trace recording and profile reconstruction of the task and Runnable states. The second challenge is the export of the profiler data for seamless import into TA Tool Suite for correlation.

The EB tresos AutoCore OS maintains a static data structure which holds OS state information, such as the currently running task, and also the state of each individual task. For a realistic AUTOSAR application, the tool needs to monitor the state of about 30 to 50 tasks, or more, by data tracing. As a sufficient number of data trace channels need to be supported, this is quite a challenge for the trace tool as well as for the on-chip trace logic of the microcontroller.

Fortunately, the Infineon AURIX processor family implements very powerful trace features which allow recording of write access to a large number of memory locations.

Based on the trace recording, the iSYSTEM Analyzer reconstructs the status of all OS tasks of all cores. The reconstructed task states follow the task state model of the so-called Best Trace Format (BTF) specification. The BTF specification is a result of the AUTOSAR tool interoperability efforts within the AMALTHEA project (amalthea-project.org). BTF specifies state models for tasks as well as for ISRs and Runnables. In addition, it defines a trace export file format. iSYSTEM as well as TA support this file format, which ensures a seamless data exchange between the tools.

Tracing of Runnables can be performed in two ways, either via program flow trace or by means of data access trace. In this project, the data trace based method is used due to its much lower trace bandwidth requirements. However, it requires so-called code instrumentation, i.e., instructions have to be added to the application code to mark the entry points and exit points of the Runnables by writing unique ID values to a specially allocated static variable.

The question now is, where is the instrumentation code to be added and where do the runnable IDs come from?

The EB tresos Studio allows the automatic generation of Runnable start and return hook functions as part of the Virtual Functional Bus (VFB) tracing capabilities. In addition, it produces a C-header file containing unique IDs (e.g., macro definitions) for each Runnable. Within the hook functions, a simple write access to the assigned status variable needs to be added, using the IDs defined in the generated Runnable ID header file. The C-header file file is also imported into the iSYSTEM Analyzer. The analyzer uses the definitions to map the Runnable IDs, recorded via data tracing of the status variable, back to the actual Runnable names as defined in the source code.

The approach presented here is the first step towards an automatized multi-core AUTOSAR run-time analysis which includes both OS objects, such as tasks, and application-level objects like Runnables. Within this cooperation, the concept will be further refined and extended to also support other and future AUTOSAR configurations.

Infineon AURIX: Hardware configuration for optimal performance

AUTOSAR strives to reach a high degree of independency of a specific microcontroller. However, in particular for multi-core microcontrollers and depending on the application, knowing the performance impact of specific memory access may be critical. The Infineon part of the workshop focused on the hardware-dependent performance impact for communication on a multi-core microcontroller.

Before addressing the multi-core specific performance analysis, it is to be checked whether the standard settings are properly applied, e.g., clock system, flash wait states, code and data caches.

- Access to core local PMI for code and core local DMI for data
- Access to a remote PMI or DMI (for code or data)
- Access to LMU RAM
- Access to PMU (flash banks)

The user must know the applied linker script. Using a default linker script, as for single-core microcontrollers, may not be sufficient. For generating a first impression about the application performance, it may be helpful to check the instruction count and cycle count per core. Similar counters are available for other architectures as well. For example, the TC1.6P core on the AURIX TC27x device, features 3 parallel pipelines, see figure 5.

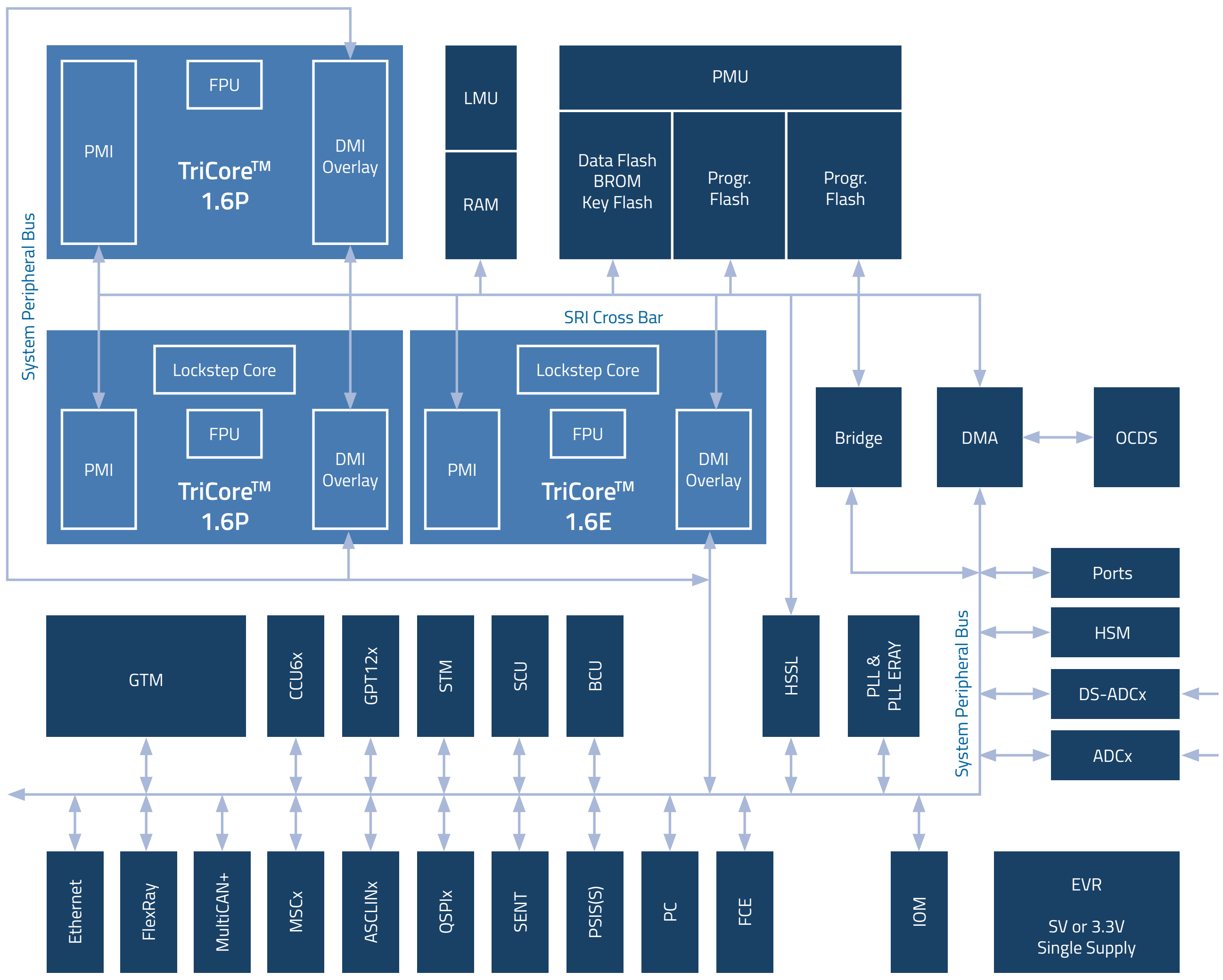


Figure 5: Architecture of the TriCore™ AURIX™ TC27x device

Often in practice, not all pipelines can be filled. An average performance of about one instruction per cycle indicates a reasonable system configuration. An IPC significantly lower than 1, for example 0.1, often indicates an issue with the system configuration.

With the correct overall configuration ensured in respect of the clock system, flash wait states, and cache enabling, we recommend the following steps to optimize the multi-core performance:

- Try to keep the code in the local PMI and the data in the local DMI
- Avoid arbitration by separating the code which is accessed from different cores to different flash banks.
- Avoid arbitration when accessing the DMI from different cores
- Be aware of the performance impact of atomic operations, e.g., the swap instruction that is typically used for implementing mutexes, spinlocks, and semaphores.

Summary

With the cooperation of three tools, combined with the detailed hardware knowledge provided by Infineon, the developer gets a detailed insight in the dynamic part of his ECU architecture. The performance problems of most projects can be detected in an early development stage, tracked, and managed. The result of the cooperation is a ready-to-use infrastructure for seamless tool interaction. Tool interaction is provided by the tool vendors and no longer part of a project.

The partner companies' experts currently extend the presented workflow to implement more detailed timing models, enable the support of different variants of operating systems and their combinations, and various tracing possibilities provided by the hardware. Are you curious about how? Meet us at

EMCC | **Embedded Multi-Core Conference**
28.-30.06.2016, Munich, Germany