KRONO-SAFE

A technology leap to accelerate the development of complex, reliable and scalable embedded systems

ASTERIOS

- An improved ENGINEERING PROCESS
- Fast prototyping and scalable MULTICORE performance
- Shortened testing and SAFETY certification

www.krono-safe.com

WHITE PAPER

Vincent DAVID

CTO at KRONO-SAFE

Contents

Introduction

Real-time embedded systems are present in every innovative industry like aerospace, automotive, industrial, rail transportation, nuclear and medical devices. Breakthroughs in artificial intelligence, industrial IoT, autonomous vehicle, robotics and smart automation are driving exponential growth in the number and complexity of real-time embedded systems. For example, an S-class Mercedes car now runs with 20 million lines of code: the embedded systems that are omnipresent in the car already account for 30% of total car cost and are expected to reach 50% by 2030. These electronic systems now drive the value of smart hardware across industries, but also are at the core of the safety and reliability of equipment.

As the growth in number and complexity of these real-time embedded systems accelerates and regulations raise the bar on reliability of critical systems, another perfect storm is brewing: the rapid adoption of multi-core processors, though necessary to support the computing power requirements of real-time systems, makes it almost impossible to integrate software and hardware using existing software engineering methods while taking advantage of the increased computer power.

Software development organizations not only have to cope with dramatic increases in the cost and time necessary to design, code, integrate and test real-time embedded systems, but they are at a loss how to take advantage of the new multi-core processors to meet the performance and quality requirements of today's and tomorrow's embedded systems.

The development of a real-time embedded system is being challenged on four levels today: the limitation of the current development process to address the complexity of these new real-time embedded systems, the difficulty and cost to meet new performance constraints, the hurdle of meeting stringent safety regulations, and the lack of support for rapid prototyping.

With this document, we start a series of four episodes to show how KRONO-SAFE addresses such problems with its ASTERIOS innovative technology:

Episode 1 – An improved engineering process: *how to simplify and accelerate the software development cycle by eliminating the need for iterative, implicit micro-design and build a system that meets the specifications,*

Episode 2 – Fast prototyping and scalable multicore performance: *how to verify as quickly as possible that the system will offer the expected functionalities, how to size the system accurately,*

Episode 3 – Shortened testing and certification: *how to generate a deterministic system to dramatically improve its quality and accelerate its testing and certification, i.e. how to build a safety-critical system with the right level of trust,*

Episode 4 – Demonstration on a use case: *The Flight Control System (coming soon).*

Episode 1 - An improved engineering process

What are the recurring problems of process engineering for real-time embedded systems?

Micro-design vs integration by parts: macro-delays and macro-costs

In the last 20 years, we have witnessed the development of new engineering methods supported by tools that offer agile, iterative development with continuous integration and integration by parts of software components. However, these methods apply only to software projects without strong integration constraints with a target processor (e.g. information systems). In other words, these methods hold their promise when the integration problems boil down to a **software / software integration** that does not depend on the target hardware (e.g. ignoring processing and communication time).

For a critical real-time embedded system, these methods do not apply because the performance constraints require conceiving a software design by making many assumptions that condition the integration on the target processor.

The Agile software engineering methods usually rely on an abstraction of the execution platforms (virtual machine, design formalism) with regards to processing time (assumptions of zero-time execution, of ideal performance, of synchronization in case of parallelism). These methods apply to systems with little or no constraints: the validation of the integration on the abstracted execution platforms is enough to show that the system is correctly integrated and totally functional (the integration by parts is horizontal and focused on functional parts). This work is complex for a critical embedded real-time system that is constrained by nature (dynamics of the environment to control/manage), and therefore inevitably approximate with the aforementioned methods: assumptions made in design can only be validated during hardware/software integration testing of the system where actual performance measurements can be performed. It is at this stage that erroneous assumptions are revealed, and their corrections will require a re-design (e.g. processing times, communication times, synchronization times, etc.) and therefore iterations must be applied to the development process. These re-design phases are costly. These iterations are difficult to quantify and plan. This hampers the possibility of assigning several development teams in parallel to the parts, and thus hinders the economic efficiency of development in terms of cost and time.

In summary, the process of **software / hardware integration** of a critical real-time embedded system naturally requires iterations on the design, integration and integration testing, which leads to lengthy and costly approaches related to the complexity of the projects to be carried out (complexity of the functional specifications, complexity of the computer and its constraints of use, the constraints of performances and necessary optimizations). Since critical on-board real-time systems are usually multi-timescale, the software design method in use today to build the dynamic architecture of the software is to split each

functional model (a task on its period) into several sub-models. This is done in order to assemble all these small and numerous model fragments according to a sequence that guarantees both functional consistency and performance constraints for integration on the specific processor: this is often called micro-design. The stronger the constraints are, the more it is necessary to iterate, often having to carry out a new design and / or redrawing of the sub-models.

Key points:

- Long, expensive engineering, margin(s) not necessarily well mastered
- Need methods and tools to facilitate iterations design / development of the executable / integration and integration tests
- Need methods and tools that do not use micro-design approach (the designer describes the constraints to be respected, and does not draft an over-constrained solution)
- Need development methods and tools that automate the process of continuous integration and integration by parts
- Need to automate as much as possible the generation of a dynamic architecture of the compiled code
- Need to guarantee as soon as possible that a dynamic architecture satisfies real-time functional and integration constraints
- Need to re-use existing features (legacy code)

Multi-core: more computing power ... but harder to take advantage of

Multicore architectures make it even more difficult to design the dynamic software architecture because the necessity to allocate the functional parts to the different cores creates dependencies on the execution across cores due to synchronizations between tasks. Not only is inter-core synchronization (writing and reading in shared memory through cache memories and cross-bars, hardware synchronization to keep the sequencing of memory operations, etc.) less efficient than intra synchronization, but a task on one core can now prevent a task on another core from running, avoiding it to perform any other computation. This is how multi-core systems may be less efficient than single-core systems!

Key points:

- Need an engineering process integrating parallelism concepts to facilitate multicore target integration
- Need to guarantee efficient single or multi-core execution, and only rely on non-blocking mechanisms for writing and reading shared data

Timing and synchronization over time

A solution to solve these types of problems is to rely on a design method where synchronizations are no longer expressed between tasks, but only by synchronizing the production and consumption of data with respect to the actual time flow. The dynamic architecture is directly described with respect to the real **time** (Time-Triggered System), independently of the underlying architecture (execution times, number of tasks and sequencing). Of course, the synchronization costs on a multi-core architecture remain, but the problem is simply divided in two: on one hand the respect of the timing constraint of the writing sequencing of shared data (i.e. before the instant of synchronization defined in the dynamic architecture), and on the other hand the waiting for the synchronization time before reading shared data. Intuitively, everything runs in the system like a music score performed by an orchestra, regardless of the number of cores (or musicians), regardless of the producer and the time of production. In the end, they all still produce the same symphony!

Key points:

- Need methodologies and tools allowing at design time to specify production and consumption of data on temporal synchronization points (TSP)
- Need development methodologies and tools that integrate into a process of continuous integration and integration by parts, not only functionally but within the dynamic architecture of real-time software components
- Need to automate as much as possible the generation of the executable code respecting the synchronization points of the dynamic architecture

What are KRONO-SAFE's solutions for process engineering?

A formal language to express parallelism, communication, cadencing and synchronizations

The constraints of real-time performances, communications and synchronization of a real-time multitasking system need to be specified unambiguously, in addition to functional descriptions (pre-existing when necessary). This objective is achieved with the PSY formal language - for Parallel SYnchronous - which allows to formally express the constraints of the dynamic architecture of the real-time multitasking system, and which is based on a design model clocked by the system time (ST): the tasks and the data shared for production and consumption are all linked to the same time referential, the real time.

This model, historically called OASIS (ref. [1]), is the foundation of the ASTERIOS technology, and is at the origin of a simplified version in the context of automotive standardization (AUTOSAR) through the concept 33 (ref. [4]) and more recently, the 640 concept (ref. [3]).

Figure 1 illustrates the specification of the temporal constraints of a task clocked by a time source named "realtime". First, two clocks are declared hMS and 5MS to define instants respectively spanned by one millisecond and five milliseconds. The task is then declared using the “agent” keyword while its temporal behavior is represented by the timeline below using the keyword “advance.” This keyword makes it possible to define the next synchronization point to be reached on the referenced clock. For example, the last “advance” of the code means to start the next processing at the 1st tick of the 5MS clock.

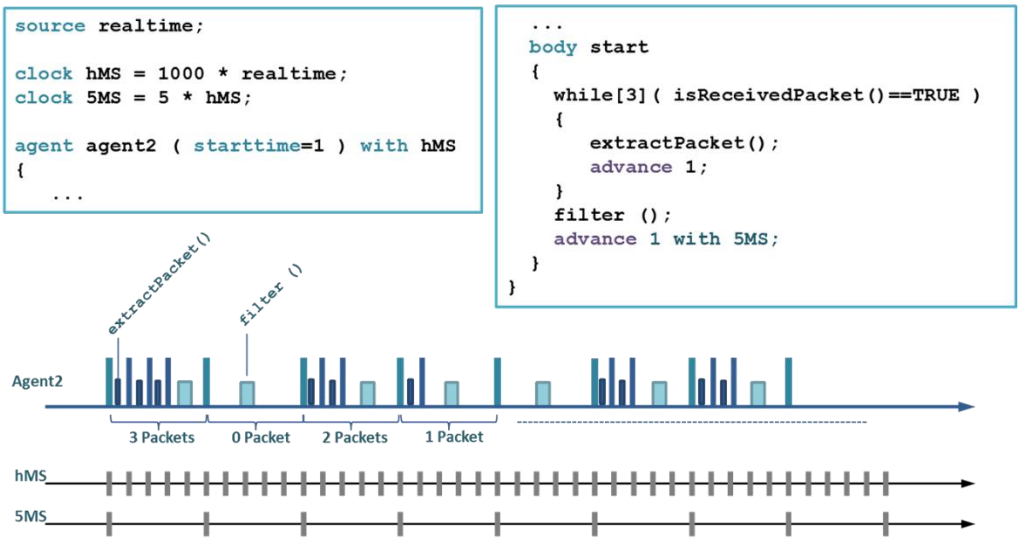

FIGURE 1. FORMAL EXPRESSION OF PSY AGENT CADENCING

This method, supported by the PSY language, makes it possible to specify all the elements necessary for a real-time embedded system: an executable component is a time-clocked task, producing and consuming

data synchronized according to the temporal sequencing formally defined during design, the time being formalized in an unambiguous and unique way. This property guarantees a deterministic behavior for all executable components grouped to form the real-time system.

This determinism therefore ensures only one possible behavior for a set of input data for the multi-tasking real-time system, but also ensures that the automatic analysis (i.e. compilation) for the computation of the dynamic architecture satisfies the required timing constraints. For this purpose, one only needs to know the task execution time estimate (budget) then an optimal dynamic architecture can be identified by (automated) calculation from the time budgets. The executable component is a task (called agent in PSY), and micro-design is no longer necessary. A correct dynamic architecture is guaranteed by design from the provided budgets. This independence of the behavior of a task (functional and dynamic components) also solves the important problem of the integration by parts: each component completely and uniquely describes the timing of its behavior vis-a-vis the synchronized data.

A suite of automatic code generation tools

ASTERIOS offers a suite of tools for automatically analyzing, compiling and generating the dynamic architecture, and automatically generating executables for any single-core or multi-core target. In particular, the production of the executable code for the specific implementation of the dynamic architecture - in the form of a Repetitive Sequence of Frames (RSF) - is based on a patented technology (ref. [2]). An RSF describes the multi-scale timeframe sequencing necessary for the execution of the tasks respecting all their temporal constraints. This replaces very advantageously a micro-design with a preemptive system taking into account, at compilation time, the material cost of a preemption on the execution time of the task (and therefore on the budget).

In this context, an iteration of the software / hardware integration process does not require a design modification: it only requires a simple recompilation / generation of code with the new assumptions about execution times (budgets). As many compilations as necessary can be done to achieve the best integration, allowing for complete optimization. Each iteration preserves the independence of the functional validation that remains correct under the deterministic principle stated above, guaranteeing the integration by parts of real-time components (the tasks).

This production of RSF by the ASTERIOS tools works both on single-core and multi-core processors. For multi-core, an RSF is generated for each core, each RSF respecting the temporal synchronization constraints necessary for the exchange of data between cores: RSFs describe at runtime absolutely all synchronization constraints necessary for the correct operation of the system, including for multi-core targets. Each iteration of the process can be single or multi-core depending on the engineering needs.

There are actually two sets of RSF intervals:
1.the first set is called the ***Head*** and is executed only once, when the application starts;
2.then the remaining intervals are executed in loop, indefinitely: these intervals are the RSF ***Loop***.

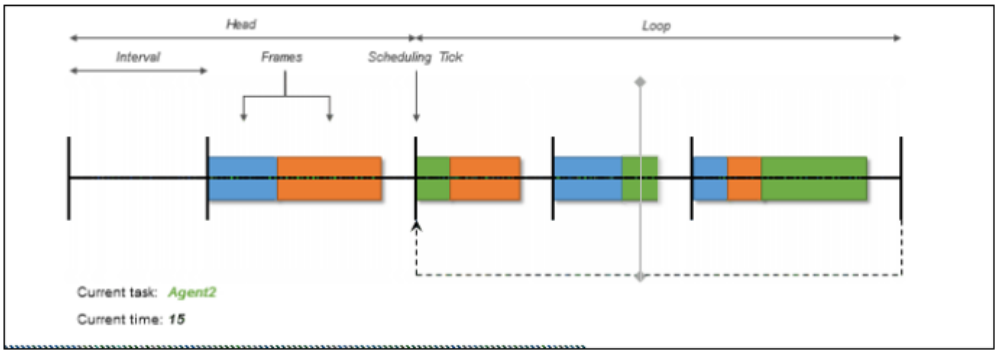

FIGURE 2. EXAMPLE OF RSF FOR THE SEQUENCING OF THREE AGENTS

Conclusion

In this episode, we have discussed the recurring problems of process engineering for real-time embedded systems. We have seen that the current process of writing software for critical real-time embedded systems requires iteration through an implicit design-coding-integration-testing cycle, with manual calculation of timing and scheduling of individual tasks. As systems typically operate in a multi-timescale, this traditional software design methodology is based on micro-design: each functional model is split into sub-models and each of these is sequenced in a way that guarantees functional, performance, and safety constraints. As systems grow in complexity, the implicit, iterative micro-design approach becomes exponentially complex: there are just too many models, sub-models and dependencies to handle. We have seen also that multi-core architectures make the task even more difficult since the necessity to allocate the functional parts to the different cores creates dependencies on the execution across cores due to synchronizations between tasks.

In order to address such complexity, KRONO-SAFE took a radically different approach to the problem, with the main goals of simplifying and accelerating the development cycle by eliminating the need for iterative, implicit micro-design. Such innovative approach relies on a design method where synchronizations are no longer expressed between tasks, but only by synchronizing the production and consumption of data with respect to the actual time flow. The PSY formal language have been developed to directly describe the dynamic architecture with respect to the real time (Time-Triggered System), independently of the underlying architecture (execution times, number of tasks and sequencing, number of core). Only one possible dynamic behavior is ensured, for a set of input data, for the multi-tasking real-time system (determinism by construction). Then, one only needs to know the task execution time estimate (budget) and an optimal dynamic architecture can be identified by (automated) calculation from the time budgets. Integration by parts is simplified by the fact that each component completely and uniquely describes the timing of its behavior; micro-design is no longer necessary.

In the next episode, we will show how the KRONO-SAFE approach helps doing early systems validation and the gain that it will imply; we will also point out the impact that the KRONO-SAFE approach has on system performances.

Episode 2 – Fast prototyping and scalable multicore performance

In episode 1 of our series “A technology leap to accelerate the development of complex, reliable and scalable embedded systems”, we have seen how the ASTERIOS approach efficiently addresses the continuous integration and delivery process for critical embedded software, with the main goals of simplifying and accelerating the development cycle by eliminating the need for iterative, implicit micro-design. In episode 2, we will see how the continuous integration and delivery process for critical embedded software take benefits of the ASTERIOS approach also when the concerns involve topics such as fast prototyping and system performance.

In fact, the engineering of critical embedded software requires **fast prototyping and simulation** to validate the proposed designs at an early stage. This prototyping and simulation phase needs to be accurate enough to prevent costly redesigns at a later stage. The problem with current engineering prototyping and simulation processes is that they are disconnected from the rest of the development process (often based on micro-design), causing problems of realism and faithfulness and leading to costly re-engineering cycles. Micro-design process does not work well in conjunction with prototyping.

On the other hand, meeting the **performance** requirements for critical embedded software is increasingly difficult as they grow in complexity. Taking advantage of multi-core processors looks like one potential solution. However, while multi-core architectures provide developers with the necessary increase in computing power, they make micro-design even less efficient as the parallelism of task execution requires to synchronize tasks across cores and manage shared memory and hardware. Because of the inherent inability to take full advantage of multi-core architectures, it is not unusual that multi-core systems using micro-design process end up being less efficient than single-core systems.

What are the problems related to fast prototyping for real-time systems?

Current engineering prototyping process is causing problems of realism

Again, as we stated in episode 1, the need for continuous integration and integration by parts is very important since not all software components are available at the same time. For an Embedded Real Time System (ERTS), this exercise obviously implies a certain faithfulness in the dynamic behaviors or at least a certain realism of these: pre-validated elements at this stage should not be challenged later to avoid

starting from scratch again. The micro-design process being long and expensive, prototypes are usually created using more flexible technology (e.g. MATLAB®) but with the disadvantage that there is no direct link with the dynamic architecture adopted in the later part of the development process. Consequently, the prototypes are not realistic and faithful to the final software, especially for dynamic behaviors.

Key points :

- Current engineering prototyping process is causing problems of realism / faithfulness as it is disconnected from the rest of the development process
- Need methodologies and tools for rapid, faithful prototyping
- Need methodologies and tools for faithful simulation
- Need "unified" prototyping and design methodologies and tools that are part of a continuous integration and integration by parts process

These needs can be addressed at two different levels:

- The first level corresponds to a software architecture exploration work by allocating elementary functional blocks to the executable components (the tasks), while respecting all the functional constraints and all the real-time constraints (e.g. End-to-End). This work can gain tremendously from automation because the many constraints of a real-time system make it difficult to find a valid architectural solution manually and quickly (at this level the problems of micro-design are very apparent, and so is the very complex problem of resource allocation theory).

- The second level concerns the need to quickly characterize the evaluated architectures to pre-validate the selected architecture to continue development. Validation techniques are based on simulations that allow for the validation of the functional elements, specifically for an ERTS, but also the pre-validation of the selected architecture in terms of performance, hoping to be as close as possible to the performance of the final processor-specific integrated system. For this, efficiency comes from the faithfulness of the architecture model compared to the actual final architecture. Unfortunately, there is often a disconnect between high-level modeling techniques (e.g. SysML) used in fast prototyping and the development process where software specifications are being written without any formal link or traceability toward the simulation models.

What are KRONO-SAFE's solutions for fast prototyping?

Determinism

From this point of view, the method supported by the PSY language makes it possible to describe all the elements necessary for a real-time embedded system such as temporal synchronization and data exchanged, with a very powerful range of expressions and flexibility. The same medium can and should be used to unambiguously describe performances, communications and synchronization constraints as early as the prototyping phase. Using the formal PSY deterministic language, only one dynamic behavior is possible by definition, its computation and automatic analysis being automatically generated. It is then very easy to execute this dynamic behavior on any simulation platform, as the model for fast prototyping guarantees a very high faithfulness and realism of the results obtained. To this end, it is possible with the ASTERIOS tool suite (which includes a simulator) to quickly execute on a PC the dynamic architecture model in an exact and deterministic fashion, without being on the target hardware, accomplishing what is called an exact **execution in simulated time**. Only time is simulated at this stage of fast prototyping, the dynamic architecture model being actually executed.

A seamless process

A seamless process between the prototyping and the design phase is achieved thanks to the exact execution in simulated time (same PSY language), with all the benefits from all the properties of the engineering model based on the independence of task behavior (functional component and real-time): integration by parts and faithful simulation. Each iteration preserves the independence of the functional validation that remains consistent (determinism). The analysis, compilation and generation of the dynamic architecture are done automatically as soon as prototyping starts. The automatic generation of executables (with the ASTERIOS tool suite) for a simulation (which is just a particular execution target) is easily and quickly performed at each iteration of the architecture exploration phase, as each iteration is just a recompilation. The dynamic architecture is identical between the prototype and the real final system.

What are the current problems of performance engineering?

Poor programming model and source of errors

Ensuring a high level of performance for an ERTS requires considering the implementation principles adopted, especially with regard to the behavior of the RTOS (Real-Time Operating System) / RTK (Real-Time Kernel) used to support the orchestration of the dynamic architecture, communications and synchronizations of the application layer.

The performance of an RTK is linked to the execution time of the system primitives (or RTK services) which must be short, bounded and known. In particular, these execution times can be variable for the application layer because they depend on the atomic sections implemented to guarantee consistency of access to shared data. The typical solutions based on locks and semaphores of traditional multi-task asynchronous systems therefore cause real difficulties for an ERTS and complicate enormously the analysis of the schedulability of a real-time system. They reduce margins, not to mention risk of deadlock in case of design errors. Regardless, and this is not the least important priority, the emergence of multi-core processor architectures requires that the solutions implemented remain operational on multi-core processors, which poses many performance issues with conventional synchronization solutions (i.e. lock, semaphore).

It should be noted that the design choices of asynchronous RTOS do not meet this objective, and that conventional RTKs apply the same schemas as the RTOS although with less services and therefore less burden (all more or less implement a POSIX or similar philosophy, including ARINC 653).

Key points :

- Need to reduce RTK user interfaces to limit design errors
- Need to separate the service layers in the RTK that are linked to functional needs of the tasks performing scheduling and task management
- Need to have a fast preemptible RTK service layer at all times
- Need for synchronization methods based on atomic sections as short as possible
- Need a service with short, bounded, and known execution times, regardless of the state of the tasks being executed
- Need for scalable multi-core performance

Specificities related to multi-core architectures

The impact of multi-core (MC) processor architectures creates a very particular set of issues, because the performance requirements of a modern processor architecture imply a memory hierarchy with cached memories, which violate a priori the "von Neumann hypotheses": the atomicity of an instruction is no longer guaranteed in certain cases (i.e. multiple access to the memory bus), and the writing order in shared memory is no longer guaranteed by the sequential order of a series of instructions on a core.

Key points :

- Need to preserve the atomicity and execution order of writes / reads on shared data

What solutions does KRONO-SAFE provide for performance management?

No blocking instructions and constant time execution

The design principles of the ASTERIOS RTK are disruptive in order to best resolve all these problems, but also to provide a level of service perfectly aligned with the concepts required to have an ERTS engineering process as defined above. An executable component is a time-sequenced task, producing and consuming synchronized data according to the formally defined temporal sequencing: the time is unambiguously and uniquely formalized, real-time sequencing at runtime having to comply with automatically generated RSFs (Repetitive Sequence of Frames).

The ASTERIOS RTK has no lock, no spin lock, no semaphore (and therefore no possible deadlock) which guarantees that no sequence of instructions can be blocked by the wait for a resource. Instead, communication services (for data production and consumption) are fully implemented based on a "lock-free / wait-free" data structure that allows writing and reading on communication buffers without any blocking instruction. The real-time system is preserved from any possible inconsistency in the data exchange between tasks because all data are implicitly linked to the formal moments of the real-time sequencing of the tasks: the "lock-free / wait-free" data structures implement time-stamped communication buffers and are accessible at any time in parallel by the task responsible for the production of given data and by as many tasks consuming the data. The read and write services of the RTK are 100% preemptible. The only necessary atomicity is the atomicity of an instruction that is guaranteed by any single-core processor (for multi-core aspects see below).

Given this vision, the RTK is thus separated into a service layer called "system layer" having the above properties, and a micro-Kernel (mK) implementing the scheduling and task management service. The mK is then the only atomic section of the RTK, necessary to carry out the switching of the tasks, and with extremely short execution time. It realizes the temporal allocation of the CPU (election of the task to execute) in constant time because it is described in an RSF built at compilation time: there are no more lists of tasks to be processed, nor any fluctuation based on the number of tasks, the execution state of the tasks, or the system layer. Many optimizations are then possible, and the mK code is very short with only about 300 instructions, providing an unprecedented level of real-time performance. All the data handled by the mK are pre-calculated offline and provided in automatically generated runtime tables, which further increases the performance of the ASTERIOS RTK and the operational reliability as everything can be verified before execution.

A major result of having created a system layer without lock, spin-lock or semaphore is that it avoids situations of unacceptable performance on a multi-core architecture. However, the main benefit of having a fully implemented system layer based on a lock-free / wait-free data structure is that when switching to multi-core, it implicitly specifies what needs to be checked in terms of atomicity and by extension, preserves the operations on the lock free / wait free structures "in order". In the ASTERIOS RTK, the required atomicity is preserved by writing and reading a descriptor only on the size of a word of the aligned memory bus, and by adding instructions of forced synchronization of read and write operations on these descriptors in shared memory. Regardless of the behavior of cached memories, the consistency of access

to wait-free / lock-free data structures is guaranteed "in order" just like a sequential execution on a single processor, regardless of the number of cores. In addition, these elements are verifiable by analysis and formally demonstrable on the RTK code: this is a fundamental differentiator of this architecture because testing multi-core software can be very combinatorial and very difficult to do in practice.

Conclusion

The revolutionary concepts introduced by KRONO-SAFE allow developers to explicitly define the requirements and constraints, and then let the ASTERIOS tool suite automatically produce an optimal dynamic architecture that can be pre-validated on a simulator (hosted on a PC). The results obtained during prototyping are guaranteed to be faithful to those that will be obtained on the real hardware, building trust in the system as early as possible without questioning the validated elements. This guarantees a better control (and reduction) of development time because it makes it possible to take advantage of a software / hardware integration, pre-validating the dynamic architecture, without having the final hardware.

The level of performance achieved by the ASTERIOS RTK is unique, and it is compatible with the requirements of new technologies requiring fast control loops (electric motors 100 us or ms). The ASTERIOS RTK is thus able to easily support 100 us tasks, allowing the conception of very reactive real-time systems thanks to a very small system overhead of RTK, even in a multi-core environment. As said previously, this is achievable thanks to the innovative design principles included in the ASTERIOS RTK: no lock, no spin lock, no semaphore (and therefore no possible deadlock), which guarantees that no sequence of instructions can be blocked by the wait for a resource. Instead, communication services (for data production and consumption) are fully implemented based on a "lock-free / wait-free" data structure that allows writing and reading on communication buffers without any blocking instruction. Also, the mK is the only atomic section of the RTK, necessary to carry out the switching of the tasks, and with extremely short execution time.

In the next episode, we will show how the KRONO-SAFE technology facilitates the certification process against increasingly stringent safety standards. We will see how it helps to reproduce, diagnose and fix software defects identified during the testing cycle and then to accelerate time for obtaining credits by certification authorities.

Episode 3 – Shortened testing and certification

Critical embedded systems need to be certified against increasingly stringent safety standards. The current programming model for parallel execution leads to non-deterministic solutions, meaning that the software cannot guarantee that given the exact same inputs, the system will always generate the same output. This not only creates faulty behaviors that might prevent compliance with industry standards and certification, but it requires very expensive and time-intensive testing cycles without ensuring to verify exhaustively the system and by consequence pushing to market products with hidden defects. Moreover, this makes it very hard to reproduce, diagnose and fix software defects identified during the testing cycle.

In this episode, we will discuss how KRONO-SAFE's approach generates deterministic systems that dramatically improve the quality of the system and accelerate the testing cycle and certification.

What are the features of KRONO-SAFE's technology for operational reliability?

Developing fully deterministic parallel systems by design

The operational safety of a system pre-supposes the prevention of errors, their detections, and finally the mitigation of the effects of residual errors, and a justification of the methods used with respect to the expected objectives.

The elements of the ASTERIOS technology introduced in previous episodes already contribute to error prevention and detection. As previously explained, an Embedded Real Time System (ERTS) designed and developed with ASTERIOS is inherently correct by design, including for a dynamic architecture, which is characterized by a compliant execution, high performance and built-in reliability.

It should also be noted that the determinism guaranteed by the design and implementation principles supported by ASTERIOS's concepts (no synchronization / asynchronous communication, no interruption, RTK without lock / spinlock / semaphore, etc.), is a requirement sine qua none for any safety critical system. Without determinism, the test coverage to be performed would be very low compared to the objectives to be achieved for a critical system, because if a software defect is systematic by nature (it is present or not at the origin, but it does not appear by chance as the execution continues over time), the effects of a defect can be systematic (e.g. division by zero) or non-systematic (e.g. inconsistency on data shared between parallel tasks). If the system is not deterministic, the tests are of a statistical nature, and in this case it is misleading to expect to do better than a system failure rate of the order of 10^{-3} / hour (which already represents a random, intensive system test for 40 days), while for a safety critical system the goal is 10^{-7} or 10^{-9} / hour. It becomes absolutely necessary to have a systematic and deterministic approach to

testing in order to achieve such a level of confidence by test, and thus the system needs to be deterministic with respect to test scenarios. ASTERIOS is a disruptive technology in this fundamental aspect, because the technology makes it possible to build deterministic parallel systems by design, which means that a multi-task real-time system developed with ASTERIOS is tested as a sequential system, and therefore everything is testable, in accordance to a safety approach. The validity of a test is not dependent on the actual moment of occurrence of the test stimulus. With ASTERIOS there is no need to take into account the combination of instants to which we must stimulate the system under test.

Guaranteed time and space partitioning

The mitigation of failures in a modern system relies on an isolation of the behavior of the various ERTS components through spatial partitioning hardware mechanisms (e.g. MPU or MMU) and temporal partitioning (e.g. watchdog / timer), with the granularity often achieved for a group of tasks. Tasks in the same group are not isolated from each other, meaning that anomalies could occur without detection at runtime, whether in real conditions or in test phase (see below). With the growing complexity of software, this risk increases because more and more components are integrated on the same processors. Therefore, if nothing is done, increasing integration leads to a growing risk of failures, combined with a growing impact.

Key points :

- Need for spatial and temporal partitioning at the task level to isolate anomalies
- Need to implement a Restricted Access Rights (MILS) restriction strategy to detect and prevent anomalies

Temporal partitioning is already guaranteed at the level of each task thanks to the RSFs as presented earlier. Indeed, the micro-Kernel only has to allocate the frames to the tasks as planned in the RSF provided in the configuration tables, and has to ensure that there is no overtaking, which is naturally performed (by design of the kernel).

The ASTERIOS solution completely defines the access rights strictly necessary for each part of each task (executable code and data) according to a MILS approach (ref. [5]). ASTERIOS tools are then able to automatically generate the binaries memory image organized so that each part will automatically be assigned an access right according to a strategy of maximum restriction according to the execution context (e.g. differentiate a producer's rights to write a consumer's reading rights on a shared buffer). The system layer of the RTK follows the same rules of partitioning and access restriction, which means that a failure in the system layer will be detected and confined in the same way, protecting within the RTK the micro-Kernel of the system layer (ref. [5]).

These access rights are calculated automatically at the creation of the binary links and are stored in a configuration table of the MPUs or MMUs that will be used by the micro-Kernel at runtime. It is during this phase that the parts are arranged in such a way as to obtain additional specific properties such as stack

overflow detection (Buffer OverFlow -BOF- or Buffer UnderFlow –BUF-). This approach is associated with a specific stack management within the RTK, each task having a stack for the application layer and a stack for the system layer. The micro-Kernel has its own stack and is associated with an execution privilege restriction: only the micro-Kernel needs privileged execution rights (i.e. supervisor) and it is the only one authorized to manage the access rights for the different execution contexts.

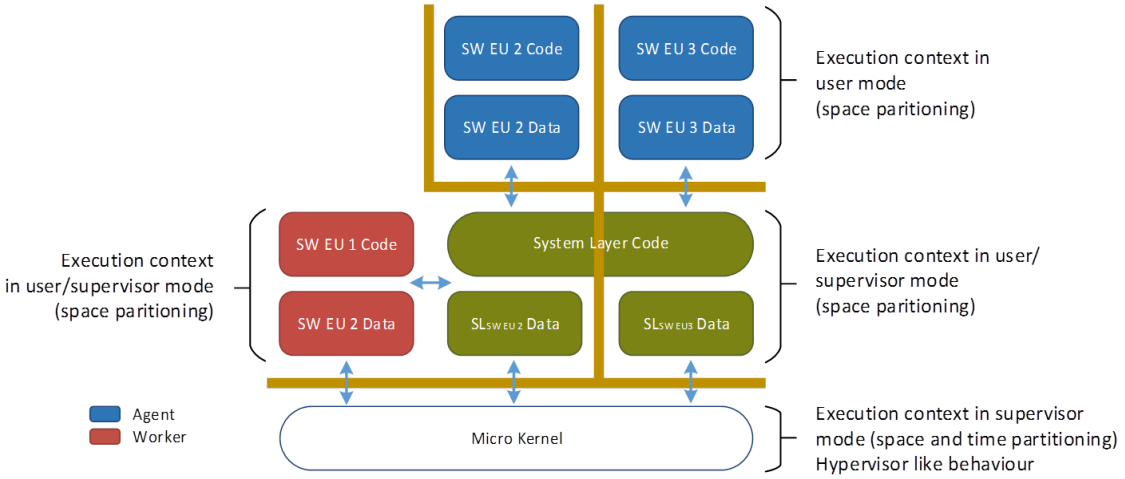

FIGURE 1. PARTITION ORGANISATION WITH RESPECT TO BINARY EXECUTABLE

It becomes impossible for a task (application or system layer) to violate the defined access policy, automatically and transparently (including for the problems of BOF). As a result, the real-time system is robust in the case of code generation errors, compilation errors, and faulty executable codes. This means that the real-time system can withstand this type of defect even if they were intentional... The implementation of safety concepts in ASTERIOS intrinsically encompasses the required properties for the safety of systems (MILS, no BOF, no execution privilege required for the system layer, micro-Kernel separated as in a hypervisor...). The system layer of the RTK lacks the classic primitives, and it just contains a single-entry point for a task to report that it has reached a node in its control graph: the operations to be performed at this node of the graph are stored in the automatically generated configuration table (without any possible error).

The testability is still greatly improved by these mechanisms of anomaly detection / containment because the real-time system is deterministic, even in case of errors. The patented ASTERIOS technology is also fundamentally disruptive on that topic (engineering and safety). In addition to detection, containment can be performed in simulation during development on an ideal model of protection that can be more restrictive than the final model on the target hardware, reinforcing the testing cycle.

Ensuring that budgets for each task are correct and provide required coverage

The robustness of the system is also achieved by ensuring that the budgets for each task are correct and cover all possible task's states, which is a very difficult theoretical problem:

Key points :

- Engineering needs to know the residual margin of the system, and needs to best allocate (e.g. equitably) the margin for each budget to be respected

When automatically generating RSFs, ASTERIOS tools automatically calculate a load-smoothing solution to maximize the minimum time remaining between each scheduling point. This makes it possible to know the remaining margin of the system with precision, and to spread the margin evenly if desirable by re-iterating with new budgets taking the margin into account.

The spatial and temporal partitioning guaranteed by ASTERIOS RTK does not depend on the safety level of the tasks themselves. This means that non-critical tasks (e.g. data management) can be easily and safely integrated near critical tasks (e.g. loop control) if necessary, with safety, flexibility and performance coming together.

How does KRONO-SAFE technology fit into a certified process?

A suite of qualified tools

The objectives of a certification are defined in the standards applicable to the different industrial domains, such as DO-178C for aeronautics and space, ISO / IEC 61508 for industrial manufacturing, 61513 and 60880 for nuclear, etc. Our purpose here is not to describe their content, but to elaborate on the benefits of the ASTERIOS methodologies and tools for users who need to certify their systems.

If we analyze DO-178C as an example, it defines objectives, many of which can be satisfied by using ASTERIOS, such as software architecture, HLR and non-ambiguous requirements (being more precise at the vocabulary level) that correspond to the level of design realized in PSY language in the ASTERIOS tools. For a more complete and detailed vision of how ASTERIOS aligns with the DO-178C, please contact KRONO-SAFE.

It is clear that the automation of code generation and dynamic architecture up to the binary facilitates:

1. the traceability required between the different stages from high-level design, as the tools are in charge of the production of the binary; from this point of view, the development cycle with ASTERIOS tools is closer to a Y cycle than a V cycle (ref. [6]);
2. the justification of achieving the certification objectives covered by the tools;
3. the certification of the code created by the automatic code generation.

The final users are therefore released to take in charge these points that are undertaken by the certification kit proposed by KRONO-SAFE.

The major advantage for the users is that part of the certification effort that is normally the user's responsibility is now taken care of by using a suite of tools qualified for this, and that the qualification of this code generation tool suite is the responsibility of the tool provider, KRONO-SAFE.

The effort can be shared because it does not depend on the application but on the tools and normative reference to be applied for the qualification of the tools, e.g. the DO-330 for aeronautics. In this case, the tool provider offers a tool qualification kit jointly with the tool suite, which KRONO-SAFE supports.

Several tool qualification strategies are available and KRONO-SAFE's strategy is to develop a suite of verification tools in parallel with the suite of code generation tools. With respect to the constraints to be taken into consideration (e.g. no common part to avoid a common failure), the verification tools must concretely satisfy the TQL-5 qualification objectives (DO-330) for a certification of the highest level in aeronautics (DO-178C DAL A). In the case of ASTERIOS, the ASTERIOS Checker suite consists of several qualified tools. Since the dynamic architecture is automatically calculated and compiled as an RSF (see RSF concept in previous episodes), the verification of the calculated RSF is performed by one of the verification tools, RSF-Checker. We do not need to prove that the calculation algorithm is correct, we prove that the calculated result satisfies the needs of all tasks in all circumstances in terms of time-allocated budgets

(model-checking). In particular, the RSF-Checker tool thus performs an exhaustive analysis / verification of all the time execution paths of all the tasks.

A certified Real Time Kernel

The RTK must also be certified, and the ASTERIOS RTK is simpler than a conventional RTK because it does not have a user interface with conventional system primitives, as the interface with the application software being only supported with automatically generated runtime tables. In this case, the RTK provider offers a RTK certification package along with the RTK, which KRONO-SAFE supports. On the other hand, the clear separation of the ASTERIOS RTK between system layer and micro-Kernel (see above) makes it possible to clearly separate the generic parts independent of the hardware target and the parts dependent on the hardware targets (e.g. the system layer is 100% generic). This approach facilitates:

1. the certification of the RTK, because its largely generic part does not need to be re-certified (except to re-run some tests if the compiler changes);
2. the pooling of certification efforts between projects.

Conclusion

In conclusion, the perfect storm of proliferation, complexity and multi-core architectures hitting the real-time embedded systems market creates a unique opportunity for the makers of software development tools for these systems. As major industries are hitting the wall of productivity, reliability and performance with existing tools and methodologies, KRONO-SAFE offers an innovative new approach to the embedded software engineering cycle and delivers a suite of tools that elegantly meets all the needs of embedded systems development. Its success and adoption in major industries, such as aerospace, automotive, railways, energy and industrial IOT, already proves that KRONO-SAFE can dramatically reduce the cost of software engineering while delivering systems that are ultra-reliable and high-performance by design, not by iteration. The transition to multi-core architectures is pushing many software engineering teams to re-evaluate their tools and methodology, and KRONO-SAFE's breakthrough technology can easily meet their requirements. Thanks to KRONO-SAFE's ASTERIOS offering, the engineering of critical real-time embedded software has never been so efficient: organizations can automate software integration on target hardware regardless of the complexity of the application, and therefore achieve in a few hours an activity that used to take several months at each iteration!

In the next (and last) episode of the series, we will describe and illustrate a use case based on a Flight Control System application. We will show in practice the full ASTERIOS software development flow applied on a multi-core system architecture: from software design up to integration and validation.

Document references

[1] «Seeking a Deterministic Multitask Framework for Safety-Critical Systems: the OASIS Project», Vincent David, Christophe Aussaguès, Jean Delcoigne and Marc Aji, NPIC&HMIT 2000. December 2000.

[2] «Method for executing tasks in a critical real-time system». Patent WO2014167197A9.

[3] «Towards Parallelizing Legacy Embedded Control Software Using the LET Programming Paradigm», Julien Hennig, Hermann von Hasseln, Hassan Mohammad, Stefan Resmerita, Stefan Lukesch and Andreas Naderlinger, 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (WiP).

[4] Concept 33 "Support for predictable concurrent software execution", Document ID 586, AUTOSAR.

[5] "Security Method Making Deterministic Real Time Execution of Multitask Applications of Control and Command Type with Error Confinement. Brevet": WO2002039277A1

[6] Wikipedia : Cycle de développement (logiciel)

Author biography

Vincent DAVID

Vincent joined KRONO-SAFE in 2013 after a 25-year career in research and development in embedded software at the French aerospace lab (ONERA) and CEA. During his time at CEA, he founded the LaSTRE laboratory, where he managed a team of 30 expert researchers and engineers. Vincent is the main inventor of most of the software technologies incorporated into KRONO-SAFE's products, and is the author of 12 patents and 40 international publications.

About KRONO-SAFE

KRONO-SAFE is a privately owned company founded in 2011 to commercialize software technologies originally developed with the French atomic energy commission (CEA).

KRONO-SAFE develops and markets a software tool-suite named ASTERIOS® based on an innovative real-time kernel (RTK) for safety-critical real-time embedded systems and providing an integrated development environment (IDE) to simulate exhaustively and integrate automatically the application on the single- or multi-core hardware target platform.

KRONO-SAFE serves markets in need of a safer and more efficient solution to develop complex real-time embedded applications. These extend to both well-established markets such as aerospace, defense, automotive, industrial automation, transportation, energy, medical and new markets springing up where safety and security converge like the Industrial Internet of Things.

With a fifty-strong team of experts and a management staff of seasoned professionals, KRONO-SAFE is firmly focused on software development as a way of efficiently serving its customers. The company holds a portfolio of patents securing all its breakthrough solutions developed so far and is actively pursuing its innovation and patenting activities.

KRONO-SAFE is currently based in France, and has plans for expanding to Europe, Asia and the USA. KRONO-SAFE is supported by several investment partners, including CM-CIC Innovation, CEA Investissement, SAFRAN Corporate Ventures and Scientipôle Capital.

Corporate office

Headquarter

86 rue de Paris, 91400 Orsay, France
Tel: +33 (0)1 77 93 21 59
contact@krono-safe.com

COPYRIGHT

KRONO-SAFE
Safe design in real-time