Proceedings of the 22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2016), Vienna, Austria, April 11-14, 2016.

OSEK-Like Kernel Support for Engine Control Applications Under EDF Scheduling

Vincenzo Apuzzo, Alessandro Biondi, Giorgio Buttazzo
Scuola Superiore Sant'Anna, Pisa, Italy
Email: v.apuzzo@hotmail.com, alessandro.biondi@sssup.it, giorgio.buttazzo@sssup.it

***Abstract*—Engine control applications typically include computational activities consisting of periodic tasks, activated by timers, and engine-triggered tasks, activated at specific angular positions of the crankshaft. Such tasks are typically managed by a OSEK-compliant real-time kernel using a fixed-priority scheduler, as specified in the AUTOSAR standard adopted by most automotive industries. Recent theoretical results, however, have highlighted significant limitations of fixed-priority scheduling in managing engine-triggered tasks that could be solved by a dynamic scheduling policy.**

To address this issue, this paper proposes a new kernel implementation within the ERIKA Enterprise operating system, providing EDF scheduling for both periodic and engine-triggered tasks. The proposed kernel has been conceived to have an API similar to the AUTOSAR/OSEK standard one, limiting the effort needed to use the new kernel with an existing legacy application. The proposed kernel implementation is discussed and evaluated in terms of run-time overhead and footprint. In addition, a simulation framework is presented, showing a powerful environment for studying the execution of tasks under the proposed kernel.

I. Introduction

Engine control applications are typically characterized by two types of computational activities: those regularly activated by a timer at constant time intervals (*periodic tasks*) and those activated at specific angular values of the crankshaft (*angular tasks*) [1]. Note that the activation rate of angular tasks is proportional to the engine rotation speed, meaning that the overall processor load (or utilization) also varies with the speed.

In these systems, periodic tasks have periods ranging from a few milliseconds up to 100 ms (see [1], page 152), whereas angular tasks (assuming that engine speed can vary from 500 to 6500 revolutions per minute (RPM) and assuming a single activation per cycle) have interarrival times ranging from about 10 to 120 ms.

To prevent overload conditions at high engine speeds, angular tasks are often implemented to reduce their computational demand as the rotation speed increases [2], exploiting the fact that simpler control algorithms are required at higher speed, where the system is more stable. In particular, angular tasks are implemented as a set of execution modes, each executed within a predefined speed range. For this reason, they are also referred to as adaptive variable-rate (AVR) tasks.

Today, within the European automotive industry, engine control applications are developed on top of fixed priority real-time kernels, under the OSEK [3] standard, within the AUTOSAR [4] standard framework. However, as already observed in [5], a pure fixed-priority scheduler is not the best choice in this type of applications, since there are several engine speeds at which any fixed priority assignment is far from being optimal, thus significantly penalizing the system schedulability. In this context, a dynamic priority scheduler, such as Earliest Deadline First (EDF) [6] would allow achieving a much higher schedulability, independently on the period values. Such a claim is also confirmed by extensive simulation experiments reported in [5], which showed that, while EDF is able to guarantee the schedulability of the system up to high utilization values close to 95%, a fixed priority scheduler exhibits a significant degradation for utilization values much lower than those observed in classical periodic scheduling. Guo and Baruah [7] also confirmed the effectiveness of EDF scheduling for AVR tasks through a speed-up analysis.

This fact was the main motivation that convinced us to develop an efficient support for AVR tasks in a real-time kernel with an EDF scheduler. Today, EDF is available in a few operating systems, as example ERIKA Enterprise [8], Linux, using the SCHED_DEADLINE scheduling class [9], or through the EDF plugin for OSEK/VDX proposed in [10].

The kernel selected for our implementation is ERIKA Enterprise [8] (ERIKA for short). Besides providing an OSEK-certified kernel (today used by automotive industries), ERIKA is (to the best of our knowledge) the only RTOS offering EDF scheduling with an OSEK-like API [11] and a static configuration of the kernel, as mandated by OSEK. However, the native EDF support in ERIKA does not include some features needed to manage engine-triggered tasks. Therefore, in this work we developed an EDF support for engine-triggered tasks that has minimal differences in terms of API and RTOS configuration with respect to standard OSEK specifications, thus minimizing the effort for integrating the proposed kernel with existing engine-control applications.

Although the presented issues have been discussed for the ERIKA RTOS, the proposed implementation can also be extended to other RTOSes that will provide EDF scheduling with an OSEK-like API.

Contributions. This paper presents the following contributions:

- A new kernel implementation within the ERIKA Enterprise operating system is proposed, providing EDF scheduling for both periodic and engine-triggered tasks. Different approaches have been considered, discussed and evaluated for the implementation.
- Experimental results are presented to evaluate the implementation in terms of footprint and run-time overhead.
- A simulation framework based on the Lauterbach Trace32

tool is presented, providing a powerful environment for studying the execution of tasks under the proposed kernel.

Paper structure. The rest of the paper is organized as follows. Section II presents the system model and an overview on the ERIKA RTOS. Section III discusses the scheduling of AVR tasks under both fixed-priority and EDF scheduling. Section IV presents the implementation of the kernel support for scheduling AVR tasks under EDF scheduling. Section V presents the proposed simulation framework. Section VI reports a set of experimental results for evaluating the kernel implementation. Finally, Section VII summarizes the results and states our conclusions.

II. Model and Background

This section introduces the model typically adopted for the engine and the related computational activities.

A. Rotation Source Model

In this paper, the engine is considered as a rotation source that triggers the execution of the AVR tasks at predefined angles. It is characterized by the following state variables:

- θ the current rotation angle of the crankshaft;
- ω the current angular speed of the crankshaft;
- α the current angular acceleration of the crankshaft.

The rotation speed ω is assumed to be limited within a range [ω^{min}, ω^{max}] and the acceleration α is assumed to be limited within a range [α^-, α^+]. Typical realistic values of such parameters are reported in Table I.

Parameter	min	max
ω (RPM)	500	6500
α (RPM/s)	-97.2 $\cdot 10^2$	97.2 $\cdot 10^2$

Table I: Typical ranges for the engine speed and acceleration.

B. Task Model

The software composing an engine control application is modeled as a set n real-time preemptive tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task can be a regular *periodic* task, or an AVR task, activated at specific crankshaft rotation angles. For the sake of clarity, an AVR task is denoted as τ_i^*.

Both periodic and AVR tasks are characterized by a worst-case execution time (WCET) C_i, an interarrival time (or period) T_i, and a relative deadline D_i. However, while for regular periodic tasks such parameters are fixed, for angular tasks they depend on the engine rotation speed ω. An AVR task τ_i^* is characterized by an *angular period* Θ_i and an *angular phase* Φ_i, so that it is activated at the following angles:

$$\theta_i = \Phi_i + k\Theta_i, \quad \text{for } k = 0, 1, 2, \ldots$$

This means that the inter-arrival time of an AVR task (in steady-state conditions) is inversely proportional to the engine speed ω and can be expressed as

$$T_i(\omega) = \frac{\Theta_i}{\omega}. \tag{1}$$

An angular task τ_i^* is also characterized by a relative *angular deadline* Δ_i expressed as a fraction δ_i of the angular period ($\delta_i \in [0, 1]$). In the following, $\Delta_i = \delta_i \Theta_i$ represents the relative angular deadline.

All angular phases Φ_i are relative to a reference position called *Top Dead Center* (TDC) corresponding to the crankshaft angle for which at least one piston is at the highest position in its cylinder. Without loss of generality, the TDC position is assumed to be at $\theta = 0$.

When considering dynamic conditions (i.e., with acceleration) the inter-arrival time of an AVR task cannot be expressed by Equation (1). Following the approach proposed by Buttazzo et al. [12], if ω is the instantaneous engine speed at the release of a job for AVR task τ_i^*, the inter-arrival time to the next job can be computed (assuming a constant acceleration α) as:

$$T_i(\omega, \alpha) = \frac{\sqrt{\omega^2 + 2\Theta_i\alpha} - \omega}{\alpha}. \tag{2}$$

As explained in the introduction, an AVR task τ_i^* is typically implemented as a set $\mathcal{M}_i$ of M_i execution modes. Each mode m has a different WCET C_i^m and operates in a predetermined range of engine speeds $(\omega_i^{m+1}, \omega_i^m]$, where $\omega_i^{M_i+1} = \omega^{min}$ and $\omega_i^1 = \omega^{max}$. Hence, the set of modes of task τ_i^* can be expressed as

$$\mathcal{M}_i = \{(C_i^m, \omega_i^m), m = 1, 2, \ldots, M_i\}.$$

We assume that the computation time of a generic AVR task can be expressed as a non-increasing step function C_i of the instantaneous speed ω at its release, that is,

$$C_i(\omega) \in \{C_i^1, \ldots, C_i^{M_i}\}. \tag{3}$$

An example of $C(\omega)$ function is illustrated in Figure 1.

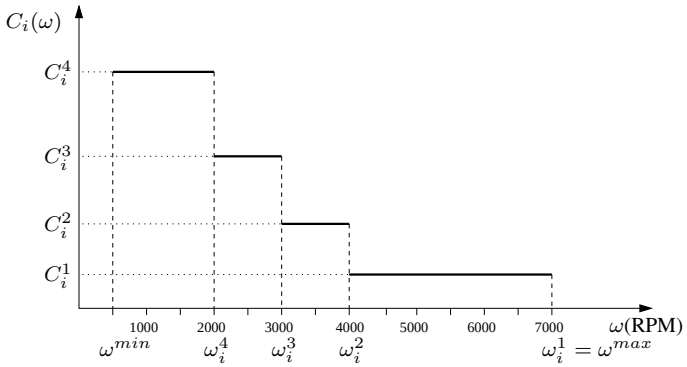

Figure 1: Worst-case execution time of an AVR task as a function of the speed at its activation.

A possible implementation for such tasks can be the one reported in Figure 2 that implements the modes reported in Figure 1. In this example a sequence of `if` statements is used for enabling the execution of a set of functions [2], [12].

The function `read_rotation_speed()` returns the speed estimate of the rotation source available at the time at which the task is activated (not at the time at which the function is executed).

In the following, when a single AVR task is addressed, the

```
#define omega1 7000
#define omega2 4000
#define omega3 3000
#define omega4 2000

TASK (sample_task) {
      omega = read_rotation_speed();
      f0();
      if (omega ≤ omega4) f1();
      if (omega ≤ omega3) f2();
      if (omega ≤ omega2) f3();
      if (omega ≤ omega1) f4();
}
```

Figure 2: Example of implementation of an AVR task.

task index is removed by the AVR task parameters for the sake of readability.

C. ERIKA Enterprise

ERIKA Enterprise [8] (ERIKA for short) is an OSEK/VDX certified RTOS that uses innovative programming features to support time sensitive applications on a wide range of microcontrollers and multi-core platforms. It is provided as an open-source kernel that allows achieving high predictable timing behavior with a very small run-time overhead and memory footprint (in the order of a few kilobytes). ERIKA comes in two kernel versions: (i) the OSEK standard, with its four conformance classes BCC1,BCC2,ECC1 and ECC2, that is also certified as OSEK-compliant, and (ii) other custom (non-standard) conformance classes that are:

- FP, offering a minimal implementation of fixed-priority scheduling with preemption thresholds [13];
- EDF, providing dynamic priority scheduling through the Earliest Deadline First (EDF) algorithm [6] and the Stack Resource Policy (SRP) [14];
- FRSH, implementing the IRIS [15] scheduling algorithm for resource reservation;
- HR, offering a two-level hierarchical scheduling framework through the M-BROE [16] algorithm.

In general, ERIKA supports both periodic and aperiodic tasks under fixed and dynamic priorities and includes mutex primitives for guaranteeing bounded blocking on critical sections.

Two types of interrupt handling mechanisms are provided: a *fast* one (also referred to as Type 1) for short and urgent I/O operations, returning to the application without calling the scheduler, and a *safe* one (also referred to as Type 2) that calls the scheduler at the end of the service routine, meant to be used for the interaction with kernel objects (e.g., for activating a task).

In ERIKA, all the RTOS objects like tasks, alarms and semaphores are *static*, as specified by the OSEK/VDX standard. This means that all the RTOS configurations are predefined at compile time and cannot be changed at run-time. The choice of using a static approach is crucial for containing both footprint and run-time overhead, obtaining a tailored RTOS image that is optimized for a specific application-dependent kernel configuration. In ERIKA, the objects composing a particular application are specified in the OSEK Implementation Language (OIL) and stored in proper configuration files. The ERIKA development environment also includes RT-Druid, which is a tool in charge of processing the OIL configuration to generate the specific ERIKA code for the requested kernel configuration.

The EDF conformance class. In this paper we focus on the EDF conformance class of ERIKA, which has been used a baseline for the development of the support for engine-triggered tasks. A key feature of this implementation is that it has the standard OSEK API, offering the possibility of making a transparent integration with existing OSEK applications. The only difference consists in an additional OIL parameter (with respect to the standard OSEK ones) for configuring the relative deadlines of tasks. The implementation relies on a circular timer for managing the internal time representation. This approach allows treating the absolute time representation (and hence absolute deadlines) with 32-bit variables, containing the runtime overhead and footprint. The access to mutual exclusive resources is regulated through the SRPT algorithm [17], which combines the SRP protocol with preemption thresholds [13] to reduce the number of preemptions and save stack space. Please refer to [11] for additional details, where the implementation of the ERIKA EDF conformance class is discussed in depth.

III. Scheduling of AVR Tasks

When scheduling classical periodic tasks (with implicit deadlines), the Rate-Monotonic priority assignment, which assigns higher priority to tasks having higher rate, is known to be optimal, thus maximizing the task set schedulability with respect to any other fixed-priority assignment. In engine control applications, however, the Rate-Monotonic criterium makes little sense, because engine-triggered tasks are, by definition, variable-rate tasks. For instance, consider a task set composed of two periodic tasks, having respectively periods $T_1 = 20\ ms$ and $T_2 = 50\ ms$, and an AVR task having angular period $\Theta = 2\pi$. A production car engine typically ranges from $\omega^{min} = 500$ RPM to $\omega^{min} = 6500$ RPM, leading to an interarrival time of the AVR task ranging from $T^{min} \approx 10\ ms$ to $T^{max} = 120\ ms$. Hence, for every fixed priority assignment, there are a set of engine speeds for which the priority assignment is different from the optimal one determined by Rate-Monotonic. As an example, consider the case in which the AVR task has the highest priority, followed by the two periodic tasks in Rate-Monotonic order. In this case, when the engine speed is high (specifically, greater than 3000 RPM), the AVR task has an interarrival time lower than $T_1 = 20\ ms$ and the priority order follows the Rate-Monotonic criterium. Conversely, at low speeds, the AVR task has an interarrival time greater than T_1, and possibly also greater than T_2, making the priority assignment far from the one established by Rate-Monotonic. The situation can be worse when considering more than one AVR task and other periodic tasks.

For this reason, it is interesting to consider alternative priority assignment rules as a function of the engine speed to support engine control tasks. EDF is a dynamic priority scheduling algorithm, known to be optimal on uniprocessor systems [18]. Under EDF scheduling, an absolute deadline d must be assigned to each job at its activation time t in order to be scheduled.

For classical periodic tasks, the absolute deadline can easily be computed by using their relative deadline D as $d = t + D$; however, this is not the case for the jobs belonging to an AVR task. As explained in Section II, an AVR task has fixed angular parameters, which becomes *variable* temporal parameters depending on the rotation source. Hence, the relative deadline of an AVR task is a variable parameter, which is a function of engine state at the release of a job and the future evolution of the rotation source in terms of acceleration.

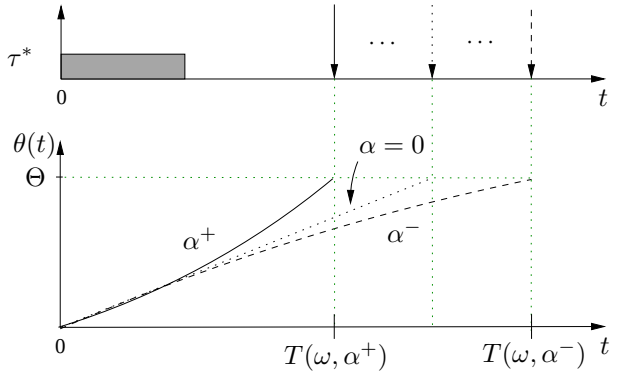

Figure 3: Possible deadlines of an AVR job activated at speed ω.

To better clarify this point, consider an AVR task τ^* having implicit angular deadline ($\Theta = \Delta$) released at time t when the rotation source has instantaneous speed $\omega(t) = \omega_0$ and angular position $\theta(t) = 0$, as reported in Figure 3. The relative deadline of τ^* is therefore equal to the upcoming interarrival time of the task, that is the time t' at which the rotation source reaches the position $\theta(t') = \Theta$. However, the upcoming arrival-time is not known a priori, since it depends on the evolution that the rotation source will have in the angular interval $[0, \Theta]$. In particular, all possible values $T(\omega) \in [T(\omega, \alpha^+), T(\omega, \alpha^-)]$ can be interarrival times to the next job, that are the ones in the range determined by the maximum acceleration α^+ and deceleration α^- allowed for the rotation source. This reasoning brings to the conclusion that computing the *exact* relative deadline of an upcoming job of an AVR task requires clairvoyance, thus preventing the achievement of an optimal schedule with EDF.

Besides the identified issue, it is anyhow possible to achieve a safe schedule assigning each job the earliest possible deadline among those compatible with the speed at its activation, that is, the one derived assuming the maximum acceleration α^+ from the task release on. Using such a rule, the relative deadline of an AVR task τ_i^* released at the instantaneous engine speed ω results

$$D_i(\omega) = \frac{\sqrt{\omega^2 + 2\Delta_i\alpha^+} - \omega}{\alpha^+}. \quad (4)$$

The above equation is obtained as a special case of Equation (2). Please note that since EDF is a job-level fixed-priority scheduling policy, such a deadline will be *fixed* for the whole execution of the upcoming job. Experimental results presented in [5] showed that scheduling engine control applications by EDF can guarantee task sets having utilization up to almost 100% of the processor utilization, while fixed-priority scheduling exhibits a significant degradation for processor utilization values that are much lower than those observed in classical periodic scheduling. The same trend has been observed in the average case by means of a scheduling simulator. As an example, Figure 4 (Figure 10 in [5]) reports the result of an experiment carried out to compare the schedulability performance of EDF and fixed-priority scheduling on a synthetic workload consisting of 5 periodic tasks and a single AVR task. Task priorities were assigned according to Rate-Monotonic, considering the shortest interarrival time of the AVR task. As it can be observed from the graph, the fixed-priority test starts rejecting task sets at very low utilization values ($U = 0.3$), while EDF is able to guarantee schedulability for utilizations very close to 1.0. Please refer to [5] for additional details on such experiments.

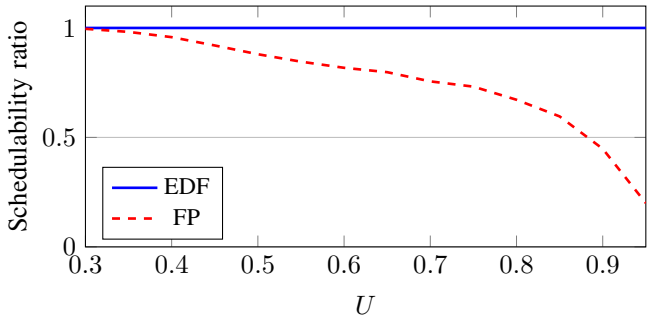

Figure 4: Schedulability performance of EDF and fixed priority scheduling for different task set utilization U. Results were obtained for task sets including 5 periodic tasks and a single AVR task.

These results convinced us to develop a kernel support for AVR tasks under EDF scheduling.

IV. EDF Kernel Support for AVR Tasks

This section describes the EDF kernel support developed for scheduling AVR tasks. Section IV-A first discusses the scheduling flow for managing AVR tasks. Then Section IV-B describes the API extension needed for supporting the activation of AVR tasks and Section IV-C discusses two approaches for handling the deadline computation of AVR tasks. Finally, Section IV-D presents the extension of the OIL specification language needed to configure AVR tasks.

A. Scheduling flow

In this section we address the relationship between the engine physical parameters (namely the engine speed and the crankshaft position) and the scheduling parameters managed by the RTOS.

Engine control applications typically maintain the notion of current engine speed (generally through an estimation technique [19], [20]), which is also used by the code of the AVR tasks to self-adapt their behavior (i.e., performing a mode change) as illustrated in Figure 2.

In this work, two measurement units are considered for the engine speed:

- *RPM*, expressed by an integer value;
- *revolutions/ticks*, expressed by a floating point value.

The first case covers the situation in which the engine speed is provided by an external device, named Time Processing Unit (TPU), commonly present in microcontrollers designed for automotive applications. An example is described in [21], where the microcontroller manufacturer provides a firmware library for the TPU returning the engine speed in RPM as a 32-bit integer value. At the same time, this case can be considered as a representative scenario in which the speed is represented as an integer value. On the other hand, the second case considers a representative scenario in which the engine speed is measured by using a temporal reference of the microcontroller (e.g., an internal timer) producing a floating point number.

In both the cases we assume that the engine speed is available at any time in a memory location.

AVR tasks are triggered by an event which is dependent on the engine rotation. Once the crankshaft reaches some particular angular positions θ a set of AVR tasks have to be activated. To this purpose, an interrupt signal is provided to the microcontroller notifying that the crankshaft has reached the position θ. This interrupt is typically originated by an external device connected to a crankshaft position sensor [21]. The firmware running on the microcontroller can then react to this interrupt with an interrupt service routine (ISR) invoking and RTOS call to actually activate the task.

As discussed in Section III, a speed-dependent relative deadline has to be computed *at the release-time of each job* to support EDF scheduling of AVR tasks. This is not required for scheduling classical periodic/sporadic tasks where the relative deadline is constant. This fact affects the mechanism of the kernel that activates a new job in the system because (i) the activation of an AVR task must take into account the engine speed for computing the absolute deadlines of its jobs and (ii) the RTOS API must offer a system call for activating AVR tasks.

In the following sections we propose an API extension for AUTOSAR/OSEK RTOS to overcome this limitation and we present a study on how to efficiently compute job relative deadlines for AVR tasks.

It is worth observing that, although AVR tasks make use of the engine speed to adapt their behavior, no specific RTOS support is required for their mode change. In fact, since an EDF scheduler is agnostic of the WCETs, AVR tasks can self-adapt their functional behavior without any intervention of the scheduler.

B. Handling activations of AVR tasks

In the standard AUTOSAR/OSEK API the task activation is performed with the following system call

```
ActivateTask (TaskType TaskID),
```

where `TaskID` represents the identifier of the task for which a job has to be activated. In the former ERIKA EDF support, the same system call has been held to keep the API OSEK-compliant. Such a call computes the absolute deadline for each upcoming job by using the implicit (constant) relative deadline.

As reported in Equation (4), however, the relative deadline of an AVR task depends on the current engine speed ω, hence determining the need for extending the standard OSEK API with a new `ActivateTask` system call, as follows:

```
ActivateTask (TaskType TaskID, SpeedType w),
```

where `w` represents the current engine speed at the time at which the task is activated.

An example of usage of the new `ActivateTask` system call is reported in Figure 5. In this example we suppose to have an interrupt signal that is generated every half rotation of the crankshaft which is handled by a type 2 interrupt service routine (please refer to Section II-C). The code in the body of the ISR simply retrieves the current engine speed and then makes use of the new system call for activating an AVR task having task identifier `AVRTask_180_degrees`.

```
ISR2 (crankAngle_0_180) {
     SpeedType w = getCurrentSpeed();
     ActivateTask (AVRTask_180_degrees, w);
}
```

Figure 5: Example of usage of the new `ActivateTask` system call.

Please note that Equation (4) holds when ω represents the actual instantaneous speed at a task activation. However, in practice, it is not possible to have an exact characterization of the instantaneous engine speed which has to be estimated in some way (e.g., as an average speed in some intervals as reported in [20]): this error can be anyway accounted by overestimating the maximum acceleration α^+.

The new `ActivateTask` system call is in charge of computing the current relative deadline $D_i(\omega)$ expressed in Equation (4), so obtaining the absolute deadline $d_i = t + D_i(\omega)$, where t is the current time at which `ActivateTask` is invoked. Once the absolute deadline is computed, all the existing code in ERIKA for managing task activations under the EDF conformance class can be reused. At this point, it results clear that the key challenge for supporting AVR tasks is to provide an efficient implementation of Equation (4) and manage the additional data structures needed to store the AVR task parameters.

C. Deadline computation for AVR tasks

Equation (4) can be computed by means of the native square root function available in the standard C mathematical library, however this approach results in an inefficient implementation, leading to unnecessary runtime overhead at each task activation. For this reason, two alternative approaches are investigated and compared in this paper for implementing the deadline computation:

1) The first approach uses an iterative computation method to approximate the value of the square root function. It has

a negligible impact in terms of footprint, but increases the run-time overhead of the `ActivateTask` system call.
2) The second approach uses a lookup table to store a set of deadline values with a given speed granularity. It introduces a small run-time overhead, but it significantly increases the footprint, which becomes dependent on the required resolution and the number of AVR tasks (as explained below).

Both approaches have been implemented in the kernel to precisely evaluate them. The user can select the preferred method in the RTOS configuration, as it will be described in Section IV-D. In the following, the two approaches are discussed and evaluated in terms of numerical error, footprint, and run-time overhead.

Please note that the main purpose of the following sections is not to describe how to efficiently implement specific numerical approximation methods, but to show how such details may significantly affect the performance of a specific RTOS implementation.

1) Fast Square Root Approach: Considering that OSEK RTOSes are static (i.e., no tasks can be created at run-time), a part of Equation (4) can be pre-computed at compile time, thus reducing the amount of computation needed at run-time. To this purpose, for each AVR task τ_i^* we define the parameters $K_i^{(a)} = \frac{1}{\alpha^+}$ and $K_i^{(b)} = 2\Delta_i\alpha^+$.

Using such parameters, Equation (4) can be rewritten as

$$D_i(\omega) = K_i^{(a)} \left(\sqrt{\omega^2 + K_i^{(b)}} - \omega \right), \tag{5}$$

thus avoiding one division and one multiplication at run-time. Hence, the additional data structures needed to support the scheduling of AVR tasks are:

```
TypeAVRParams K_A[MAX_AVR_TASKS];
TypeAVRParams K_B[MAX_AVR_TASKS];
```

where `MAX_AVR_TASKS` represents the number of AVR tasks in the task set and `TypeAVRParams` is a floating point data type.

Observing Equation (5), it is easy to note that the bottleneck for having an efficient implementation for the deadline computation is the square root function.

After comparing different methods for implementing the square root function, we identified an algorithm, denoted as *FastSQRT* [22], providing a fast computation of an approximation of the square root. Such an algorithm relies on the Newton's method for iteratively computing the square root of a number, but improves on its original formulation by selecting an initial value for the iteration that is able to considerably reduce the error with only two iterations. The initial value for the iteration is computed by means of a particular constant which has been formally studied in [22], [23].

As reported in Appendix A, we identified that the FastSQRT algorithm presents an error always lower than 0.04% for this specific application domain. This algorithm has also been evaluated in terms of run-time overhead: the results are reported in Section IV-C3.

$\Delta\omega$ (RPM)	AVG err (%)	MAX err (%)	Footprint (bytes)
32	0.002	0.013	750
64	0.009	0.05	376
128	0.036	0.2	188
256	0.145	0.79	96
512	0.58	2.99	48
1024	2.36	10.493	24

Table II: Percentage of error and footprint for the lookup table approach under different values of $\Delta\omega$.

2) Lookup Table Approach: The second approach considered in this work for computing the deadline of AVR tasks relies on an off-line pre-computation of Equation (4) for the whole range of engine speeds. The values of Equation (4) are computed off-line by using a quantization step $\Delta\omega$ and stored as constants in an array. Details on the construction of the lookup table are reported in Appendix B.

Please note that, the value of $\Delta\omega$ results now crucial for determining the precision and the additional footprint imposed by this approach. Table II reports a numerical evaluation of both error and footprint for different values of $\Delta\omega$. Since deadlines in ERIKA are represented in ticks, each element of the lookup table can be represented as a 32-bit integer value leading to a cost of 4 bytes each.

Overall, although a considerably small footprint is required for achieving a good precision (e.g., with $\Delta\omega = 256$ RPM), the main drawback of this approach is that (in the general case of AVR tasks having all different angular deadlines) a lookup table has to be generated for each task, so limiting the scaling for high number of tasks on memory-constrained platforms.

3) Run-Time Overhead Comparison: Table III reports the run-time overhead for both the FastSQRT algorithm and the lookup table approach, together with the one of using the SQRT function of the standard C library. The results have been obtained on the STM32F4 platform running at 168Mhz with FPU enabled and using the GNU ARM compiler. As it can be noted from the table the FastSQRT algorithm shows a significant improvement on the standard SQRT function, halving the run-time in case of representation in RPM and being one-third in case of using revolutions/ticks. The lookup table approach resulted very efficient for the RPM case, where no floating point operations are involved, while resulted comparable to the FastSQRT in case of speed representation in revolutions/ticks, due to the floating point operations involved in Equation (6). Finally, note also that the application of the FastSQRT algorithm in the RPM case resulted more costly with the respect to the revolution/ticks case: this is because of the final conversion of the deadline from minutes (as the temporal unit on RPM) to ticks. Overall, this study highlights how a proper implementation for the deadline computation can significantly reduce the overhead containing both footprint and precision errors.

D. Extended OIL Specification

As stated in Section II-C, AUTOSAR/OSEK RTOSes are configured at compile time through a specific standard language

RPM

	FastSQRT	*Lookup Table*	*SQRT*
MAX	2.23 μs 188 cycles	0.3 μs 26 cycles	5.42 μs 456 cycles
AVG	2.08 μs 176 cycles	0.25 μs 22 cycles	5.09 μs 428 cycles

revolutions/ticks

	FastSQRT	*Lookup Table*	*SQRT*
MAX	1.69 μs 143 cycles	1.9 μs 164 cycles	5.21 μs 438 cycles
AVG	1.51 μs 127 cycles	1.55 μs 131 cycles	4.89 μs 411 cycles

Table III: Run-time comparison of the considered approaches for computing the deadline of AVR tasks. The results were obtained on a STM32F4 microcontroller running at 168Mhz with FPU enabled.

named OIL.

Standards play a crucial role in industrial design flows, therefore keeping the implementation adherent as much as possible to a standard increases its practical applicability. For this reason, in the following we show that the proposed kernel implementation can be managed by making minimal changes on the standard AUTOSAR/OSEK RTOS configuration.

In order to support the definition of AVR tasks, the standard OIL specification used for a task has been extended by adding the following fields:

- `ALPHA_MAX`, containing the maximum acceleration for the rotation source triggering the AVR task;
- `ANG_DEADLINE`, containing the angular deadline Δ of the task.

Both the fields must be inserted inside an `AVR_TASK` OIL construct, as illustrated in the example reported in Figure 6. Different measurements units are available for representing the values of such fields.

Since our implementation is able to handle the speed representation in RPM or revolutions/ticks, an OIL field named `SPEED_TYPE = {RPM, REVS_TICKS}` is provided for selecting among these two options. If `RPM` is selected, the kernel is configured, as a default option, for using lookup tables to compute AVR task deadlines; otherwise, if `REVS_TICKS` is selected, the FastSQRT algorithm is used as a default option. Additional OIL fields are also provided to have a custom configuration in both the cases, forcing the use of the FastSQRT algorithm or lookup tables with different quantization steps $\Delta\omega$.

The RT-DRUID tool, provided together with ERIKA, has been modified to support the extended specification. Such a tool is in charge of identifying the AVR tasks and extract the values specified in the `ANG_DEADLINE` and `ALPHA_MAX` fields. When selecting the FastSQRT algorithm, such values are used to precompute the parameters $K_i^{(a)}$ and $K_i^{(b)}$ to fill the data structures `K_A` and `K_B` with their corresponding values. Otherwise, the values are used by RT-DRUID to automatically generate the lookup tables.

```
KERNEL_TYPE = EDF {
      TICK_TIME = "11.9ns";
      SPEED_TYPE = "RPM";
};
TASK sampleTask {
      AVR_TASK = TRUE {
         ALPHA_MAX = "0.000162 RPms2";
         ANG_DEADLINE = "180 degrees";
      };
      SCHEDULE = FULL;
      STACK = SHARED;
};
```

Figure 6: Example of OIL configuration for an AVR Task.

V. A Simulation Framework

In this section we present a simulation framework for studying the execution of tasks under the kernel support proposed in this paper. The framework is based on the Lauterbach TRACE32® PowerView IDE: an overview on such an environment is reported in Section V-A. The Lauterbach IDE has been extended with two custom plugins (presented in Section V-B) aiming at supporting the execution of both periodic and engine-triggered tasks with the proposed kernel for ERIKA Enterprise.

A. Lauterbach TRACE32

Lauterbach TRACE32 is a set of modular microprocessor development tools produced by Lauterbach GmbH, the world's larger producer of hardware assisted debug tools for microprocessors. The solution consists in both a software system and a set of hardware instruments where all is managed through the TRACE32 PowerView Integrated Development Environment (IDE) which offers a clean and powerful user-interface. The modular hardware and software solutions supports up to 350 different CPUs.

The TRACE32 PowerView IDE offers intuitive, consistent, and fast access to debug and trace information. Advanced debug features, profiling, support of multicore and multiprocessor systems, and support of almost every RTOS facilitate the analysis of system performance ensure the quality the quality of embedded software designs. In fact, without proper tools, the development and debug process of software for embedded microcontrollers is generally challenging, especially in case of time-dependent software where the debug can perturbate its timings. For this reason, the use of an hardware tracer is commonly adopted.

Together with all the software infrastructure offered for the Lauterbach hardware instruments, the TRACE32 suite provides an instruction set simulator, that supports a large number of microcontrollers. The TRACE32 suite based on the instruction set simulator is today distributed for free by Lauterbach. Thanks to this simulator, it is possible to execute real code collecting a large set of debug and trace informations without having any hardware device (neither the Lauterbach hardware instruments nor the actual microcontroller) hence enabling the possibility

to build very powerful testing and development environment. For this reason, we decided to extend the TRACE32 suite for supporting the execution of the EDF kernel proposed in this work. The next section describes the details of the realized simulation framework.

B. Description of the framework

The main limitation of the TRACE32 suite is that it only offers the instruction set simulator for the CPU of a microcontroller and not the simulation of its peripherals devices (e.g, the timers). However, TRACE32 offers a standard interface, referred to as Peripheral Simulation Model (PSM) [24], that allows developing custom simulated peripherals. In particular, the PSM allows a developer to write custom software that reacts and accesses to specific registers located into the physical memory of the simulated microcontroller, thus allowing the simulation of memory-mapped peripheral registers. In addition, the PSM provides functions for interacting with the simulated CPU and other modules of the TRACE32 simulator. The custom extensions are developed in C language and compiled as dynamic linked libraries that can be loaded from the TRACE32 PowerView IDE.

In order to simulate the execution of our EDF kernel executing engine-triggered tasks we needed to implement two external libraries for the simulator, that are

- *Crankshaft Simulator*, in charge of generating interrupts related to the rotation of a simulated crankshaft, hence generating the activation of AVR tasks;
- *Free Running Timer*, used by the EDF kernel to handle the time representation in the system.

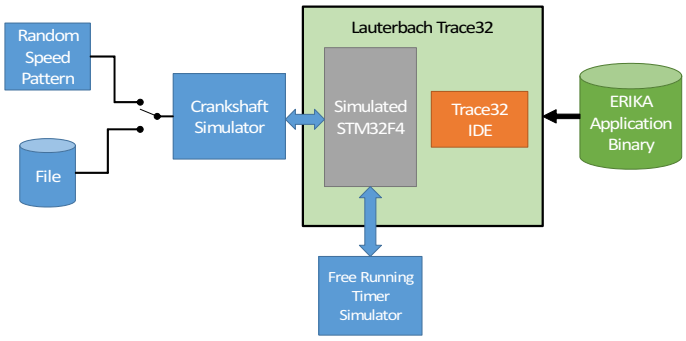

Figure 7: Schematic representation of the implemented simulation framework.

Figure 7 reports a schematic representation of the implemented simulation framework. As a reference platform we selected the STM32F4 produced by STMicroelectronics, whose CPU simulation is offered by Lauterbach TRACE32. The simulator module of the STM32F4 is connected to an interrupt controller simulator module (provided by Lauterbach) which is in turn connected to our Crankshaft Simulator module. The Crankshaft Simulator module generates angular events as interrupts, at which it is possible to react through interrupt service routines containing `ActivateTask` primitives that trigger AVR tasks (see Figure 5). As explained in Section II, the time instants at which such angular events occur are directly dependent on the speed evolution of the engine. Two different sources are available for generating angular events: (i) Random Speed Pattern, which generates a random speed evolution according to the rotation source model presented in Section II given a set of configurations parameters (like maximum/minimum acceleration), and (ii) File, which loads a predetermined speed pattern from a file. In addition to the generation of angular events, the Crankshaft Simulator uses the PSM interface for communicating the current engine speed to the application into a memory area, thus simulating (as example) the presence of a TPU [21].

The Free Running Timer module simulates a 32 bit timer available in the STM32F4. Its implementation uses the PSM interface for reacting to specific registers of the microcontroller that are used for configuring the timer and reading its current value. The value of such a timer is the temporal reference needed by the kernel for computing absolute deadlines of tasks and is configured with the resolution specified in the `TICK_TIME` field in the OIL configuration (please refer to Figure 6).

The realized simulation framework is able to collect a full trace related to the execution of an application, which can then processed, explored and partially re-executed. As an example, we reported in Figure 8 a screenshot taken from the Lauterbach TRACE32 PowerView IDE, where a functionality of the tool is adopted for collecting the execution trace of the tasks.

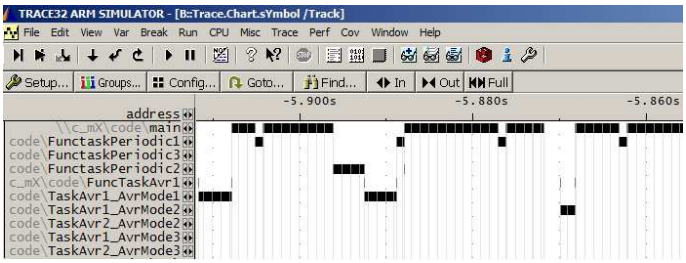

Figure 8: Screenshot from the Lauterbach TRACE32® PowerView IDE.

For the sake of completeness, Table IV reports the time needed to collect the trace for a given simulation time (indicated as Trace time), together with the processing time needed to extract the execution trace (indicated as Processing time) and the corresponding memory usage for storing the trace in RAM. Such results are obtained executing the simulation framework on a machine equipped with an Intel Core i7 4790k processor running at 4 GHz with 32GBs of RAM. As it can be noted from the table, the simulator is able to collect and process 6 seconds of simulation in about 3 minutes; however it requires a significant amount of memory (up to 25 GBs) for the storage of all such data. This huge amount of memory is required by the tool because it stores all the state of the CPU and the simulated peripheral devices for every CPU clock tick. At today the possibility of limiting the information that are stored during a simulation (hence limiting the memory occupancy) is not available in the TRACE32 simulator. This limitation can be overcome in part by storing execution traces into files that can be later processed in batch.

Simulation time (secs)	Trace time (secs)	Processing time (secs)	RAM occupancy
1	19	16	4.1 GBs
3	48	40	12.3 GBs
6	96	92	25.2 GBs

Table IV: Times and memory consumption for the proposed simulation framework.

VI. Experimental Evaluation

In this section we report experimental results aiming at evaluating the implementation of the proposed EDF kernel (referred to as EDF-AVR) in terms of footprint and run-time overhead. For comparison purposes we report also experimental results obtained with the OSEK kernel of ERIKA, the conformance class FP (providing a minimal implementation of fixed-priority scheduling) and the original EDF conformance class. Please refer to Section II-C for further details. Regarding the OSEK kernel we selected its variant named BCC2, which provides fixed-priority scheduling with stack sharing, more than one task per priority level and multiple pending task activations.

Figure 9 reports the footprint in bytes for OSEK BCC2, FP, EDF and EDF-AVR kernels as a function of the number of AVR tasks keeping the number of periodic tasks to 2. We decided to study the footprint varying the number of AVR tasks because the OSEK BCC2, AVR and FP kernels do not distinguish such tasks from the periodic ones, while the kernel proposed in this paper provides additional code and data structures for AVR tasks. The results are obtained compiling the kernels for the STM32F4 platform using the GNU ARM compiler with the size optimization (-Os) flag enabled. In these experiments we considered the EDF-AVR kernel configured for using the FastSQRT algorithm for computing the deadlines of AVR tasks. Please refer to Table II for evaluating the footprint for the case in which the kernel is configured for using lookup tables.

As it can be observed from the graph, the OSEK BCC2 kernel shows the greater footprint: this is because it contains a consistent number of data checks and mechanism required for being fully OSEK compliant. The EDF-AVR kernel has the same footprint when no AVR tasks are present (as clearly expected) and requires around 200 additional bytes for handling one AVR task (including the implementation of the `ActivateTask` primitive) with respect to EDF kernel.

Another experiment has been conducted for measuring the resulting run-time overhead of the `ActivateTask` system call. Such a system call computes the task deadlines and manages the ready queue, resulting in the most time-consuming kernel mechanism. Table V reports the maximum and the average run-time overhead (expressed in microseconds and cycles) for the `ActivateTask` of EDF-AVR under both the cases of application of the FastSQRT algorithm and the lookup table. For comparison purposes, the table also reports the overhead for the EDF kernel used as a baseline for this work. The run-time overhead for the `ActivateTask` for the FP kernel resulted about 2.2 μs (420 cycles) with marginal variations with the number of tasks (due to the constant time

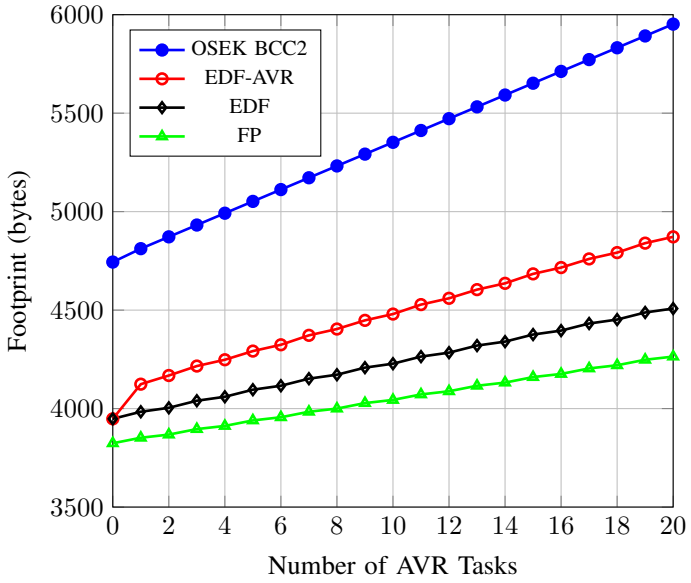

Figure 9: Footprint in bytes for different conformance classes of ERIKA Enterprise as a function of the number of AVR tasks on STM32F4 with GNU ARM compiler.

ready queue management). As it can be observed from the table, the run-time overhead of the proposed kernel is comparable with the one of the FP kernel when a lookup table is used (with integer speed representation), while is twice larger when the FastSQRT algorithm is used (with floating point speed representation). In both the cases a marginal increment has been identified as the number of tasks increases.

VII. Summary and Conclusions

This paper presented a new kernel for the ERIKA Enterprise real-time operating system for supporting EDF scheduling of engine control applications. Being engine control applications typically managed by an AUTOSAR/OSEK standard operating system (RTOS), the new kernel has been conceived to have an API with minimal differences with respect to the OSEK standard, thus limiting the effort for adopting the proposed solution in existing engine control applications. The same design constraints have been considered for the RTOS configuration, providing minimal extensions to the OSEK Implementation Language (OIL), which is the standard language for configuring an OSEK RTOS.

It has been identified that deadlines of engine-triggered tasks depend on the engine speed and two different approaches have been considered for the kernel implementation depending on the representation of the engine speed available in the system. An approach is based on a fast algorithm for computing the square root function (FastSQRT), while the other one relies on lookup tables. Both approaches have been discussed and compared in terms of precision, footprint, and run-time overhead. A comparison made with the existing EDF kernel in ERIKA Enterprise, showed that the additional overhead introduced by the proposed implementation ranges from a few hundreds of nanoseconds to a maximum of 1.5 microseconds (in a reference platform running at 168Mhz), depending on

Num. of Tasks		3	5	7	10
EDF MAX	μs	2.62	2.77	2.82	2.90
	cycles	440	465	484	487
EDF AVG	μs	2.59	2.70	2.78	2.86
	cycles	435	453	467	480
EDF-AVR (FastSQRT) MAX	μs	4.10	4.21	4.36	4.46
	cycles	689	707	732	749
EDF-AVR (FastSQRT) AVG	μs	4.0	4.18	4.25	4.39
	cycles	762	702	714	737
EDF-AVR (Lookup Table) MAX	μs	2.95	3.04	3.15	3.22
	cycles	495	510	529	541
EDF-AVR (Lookup Table) AVG	μs	2.91	3.01	3.14	3.20
	cycles	489	505	527	537

Table V: Maximum and average run-time overhead in microseconds and cycles for the proposed EDF kernel under different configurations. The overhead has been measured on an STM32F4 platform running at 168Mhz with FPU enabled.

its configuration. In the presence of 10 AVR tasks, the new implementation requires about 250 bytes of additional footprint with respect to the existing EDF kernel and less than 500 bytes increment with respect to a minimal implementation of fixed-priority scheduling.

In this paper, we also proposed a simulation framework for studying the execution of tasks under the proposed kernel. Such a framework has been realized extending the Lauterbach TRACE32® suite with an engine crankshaft simulator and other modules to support the execution of the implemented EDF kernel. Thanks to the instruction set simulator offered by Lauterbach TACE32, it is now possible to collect execution traces of real code without adopting hardware debuggers/tracers, nor the actual microcontroller.

As a final summary, to use the proposed kernel with an existing engine control application running upon an OSEK RTOS, the user has to

- identify the mechanism for reading the engine speed (which is generally already present in such applications [2]);
- replace all the occurrences of the `ActivateTask` call used for triggering the AVR tasks with the new version proposed in Section IV-B;
- introduce the additional parameters (angular deadline and maximum engine acceleration) in the OIL configuration file, as specified in Section IV-D.

Please, also note that the proposed EDF kernel requires a temporal reference for managing absolute task deadlines. Such a reference can be implemented by reserving a timer in the microcontroller (which can automatically be handled by the device drivers in ERIKA) or connecting a temporal reference already present in an existing application.

ACKNOWLEDGEMENTS

The authors like to thank Paolo Gai from Evidence S.R.L. and Maurizio Menegotto from Lauterbach Italia for their support which helped to improve this work.

REFERENCES

[1] L. Guzzella and C. H. Onder, *Introduction to Modeling and Control of Internal Combustion Engine Systems*. Springer-Verlag, 2010.

[2] D. Buttle, "Real-time in the prime-time," Keynote speeach given at the 24th Euromicro Conference on Real-Time Systems (ECRTS 2012), Pisa, Italy, July 12th, 2012.

[3] OSEK, *OSEK/VDX Operating System Specification 2.2.1*. http://www.osek-vdx.org: OSEK Group, 2003.

[4] AUTOSAR, *AUTOSAR Release 4.1, Specification of Operating System*, http://www.autosar.org, 2013.

[5] A. Biondi, G. Buttazzo, and S. Simoncelli, "Feasibility analysis of engine control tasks under EDF scheduling," in *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS 2015)*, Lund, Sweden, July 8-10, 2015.

[6] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, January 1973.

[7] Z. Guo and S. Baruah, "Uniprocessor EDF scheduling of AVR task systems," in *Proc. of the ACM/IEEE 6th International Conference on Cyber-Physical Systems (ICCPS 2015)*, Seattle, USA, April 2015.

[8] P. Gai, G. Lipari, L. Abeni, M. di Natale, and E. Bini, "Architecture for a portable open source real-time kernel environment," in *Proceedings of the Second Real-Time Linux Workshop and Hand's on Real-Time Linux Tutorial*, November 2000.

[9] D. Faggioli, F. Checconi, M. Trimarchi, and C. Scordino, "An EDF scheduling class for the linux kernel," in *Proc. of the 11th Real-Time Linux Workshop (RTLWS)*, Dresden, Germany, September 28-30, 2009.

[10] C. Diederichs, U. Margull, F. Slomka, and G. Wirrer, "An application-based EDF scheduler for OSEK/VDX," in *Proc. of the Design, Automation and Test Conference in Europe (DATE 2008)*, Munich, Germany, March 10-14 2008.

[11] G. Buttazzo and P. Gai, "Efficient implementation of an EDF scheduler for small embedded systems," in *Proceedings of the 2nd Workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT 2006)*, Dresden, Germany, July 2006.

[12] G. Buttazzo, E. Bini, and D. Buttle, "Rate-adaptive tasks: Model, analysis, and design issues," in *Proc. of the Int. Conference on Design, Automation and Test in Europe (DATE 2014)*, Dresden, Germany, March 24-28, 2014.

[13] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in *Proc. of the 6th IEEE Int. Conference on Real-Time Computing Systems and Applications (RTCSA'99)*, Hong Kong, China, December 13-15, 1999.

[14] T. P. Baker, "Stack-based scheduling for realtime processes," *Real-Time Systems*, vol. 3, no. 1, pp. 67–99, April 1991.

[15] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo, "IRIS: A new reclaiming algorithm for server-based real-time systems," in *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium*, Toronto, Canada, May 25-28 2004.

[16] A. Biondi, G. Buttazzo, and M. Bertogna, "Supporting component-based development in partitioned multiprocessor real-time systems," in *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS 2015)*, Lund, Sweden, July 8-10, 2015.

[17] C. Diederichs, U. Margull, F. Slomka, and G. Wirrer, "Design methodologies and tools for real-time embedded systems," in *Special Issue of Design Automation for Embedded Systems*, 2002.

[18] M. Dertouzos, "Control robotics: the procedural control of physical processes," *Information Processing*, vol. 74, 1974.

[19] A. Biondi and G. Buttazzo, "Real-time analysis of engine control applications with speed estimation," in *Proc. of the Int. Conference on Design, Automation and Test in Europe (DATE 2016)*, Dresden, Germany, March, 2016.

[20] R. I. Davis, T. Feld, V. Pollex, and F. Slomka, "Schedulability tests for tasks with variable rate-dependent behaviour under fixed priority scheduling," in *Proc. 20th IEEE Real-Time and Embedded Technology and Applications Symposium*, Berlin, Germany, April 15-17, 2014.
[21] Freescale semiconductor Application Note AN3769. Using the engine position (CRANK and CAM) eTPU functions. [Online]. Available: http://cache.freescale.com/files/32bit/doc/app_note/AN3769.pdf
[22] C. Lomont. Fast inverse square root. [Online]. Available: http://www.lomont.org/Math/Papers/2003/InvSqrt.pdf
[23] B. Self. Efficiently computing the inverse square root using integer operations. [Online]. Available: https://www.math.washington.edu/~morrow/336_12/papers/ben.pdf
[24] Trace32 instruction set simulator. [Online]. Available: http://www2.lauterbach.com/pdf/simulator_api.pdf

APPENDIX A
ERROR OF THE FASTSQRT ALGORITHM

As part of this work we performed a numerical study on the deadline computation by using the FastSQRT algorithm. Such an algorithm has been applied considering all the speeds in a typical range for a production car engine, that is from 500 to 6500 RPM. Then, the error with respect to the exact square root function has been computed. The results are reported in Figure 10 for both the measurement units of the engine speed considered in this work.

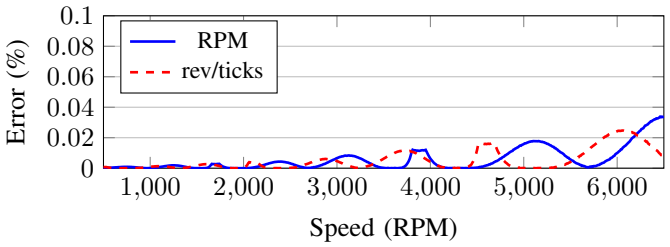

Figure 10: Error of the FastSQRT algorithm.

APPENDIX B
LOOKUP TABLE CONSTRUCTION

As stated in Section IV-C2, the values of Equation (4) are computed off-line by using a quantization step $\Delta\omega$ and stored as constants in an array whose elements are denoted as $W_j, j = 0, 1, \ldots, \left\lceil \frac{\omega^{max} - w^{min}}{\Delta\omega} \right\rceil$, so obtaining a lookup table. At run-time, a weighted average is then used to estimate the value of the deadline. More specifically, suppose to have a current engine speed ω and let $Y = \frac{\omega - \omega^{min}}{\Delta\omega}$. An index j for the look table is computed as $j = \lfloor Y \rfloor$ and the value $D(\omega)$ is obtained as

$$D(\omega) = \frac{qW_j + lW_{j+1}}{q + l}, \tag{6}$$

where q and l represent the weights of the weighted average and their definition depends of the measurement unit used to represent the engine speed ω. When the speed is represented in RPM (as an integer value) we defined $q = \Delta\omega - l$ and $l = (w - w^{min}) - j\Delta\omega$, leading to $q + l = \Delta\omega$, where l can be computed by means of a modulo operation. In this way, if $\Delta\omega$ is defined as a power of two, the value of $D(\omega)$ can be efficiently computed without involving any floating point operation implementing the ratio as a bit shift operation.

On the other hand, when the speed ω is represented in revolutions/ticks (as a floating value), different formulations for q and l can be defined to optimize the computation. In fact, defining $q = 1 - l$ and $l = Y - j$, we obtain $q + l = 1$, so avoiding the division in Equation (6). In addition, the value of l can be computed by a truncation operation by a casting to an integer value.