Operating Energy-Neutral Real-Time Systems

PETER WÄGEMANN, TOBIAS DISTLER, HEIKO JANKER, PHILLIP RAFFECK, VOLKMAR SIEH, and WOLFGANG SCHRÖDER-PREIKSCHAT, Friedrich-Alexander-Universität Erlangen-Nürnberg

Energy-neutral real-time systems harvest the entire energy they use from their environment. In such systems, energy must be treated as an equally important resource as time, which creates the need to solve a number of problems that so far have not been addressed by traditional real-time systems. In particular, this includes the scheduling of tasks with both time and energy constraints, the monitoring of energy budgets, as well as the survival of blackout periods during which not enough energy is available to keep the system fully operational.

In this article, we address these issues presenting EnOS, an operating-system kernel for energy-neutral real-time systems. EnOS considers mixed time criticality levels for different energy criticality modes, which enables a decoupling of time and energy constraints when one is considered less critical than the other. When switching the energy criticality mode, the system also changes the set of executed tasks and is therefore able to dynamically adapt its energy consumption depending on external conditions. By keeping track of the energy budget available, EnOS ensures that in case of a blackout the system state is safely stored to persistent memory, allowing operations to resume at a later point when enough energy is harvested again.

CCS Concepts: • **Computer systems organization → Real-time operating systems**;

Additional Key Words and Phrases: Energy harvesting, energy budget monitoring

ACM Reference format:
Peter Wägemann, Tobias Distler, Heiko Janker, Phillip Raffeck, Volkmar Sieh, and Wolfgang Schröder-Preikschat. 2017. Operating Energy-Neutral Real-Time Systems. *ACM Trans. Embedd. Comput. Syst.* 17, 1, Article 11 (August 2017), 25 pages.
https://doi.org/http://dx.doi.org/10.1145/3078631

1 INTRODUCTION

With the efficiency of energy-harvesting equipment (e.g., solar panels, buck-boost converters) steadily increasing, more and more embedded systems are deployed whose lifetime is not limited by their initial battery charge. Instead, such energy-neutral systems remain operational as long as they are able to draw energy from their environment. Such an approach is especially beneficial in

This work is supported by the German Research Foundation (DFG), in part by Research Grant no. SCHR 603/13-1, Research Unit FOR 1508 under grant no. SCHR 603/14-2, the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89, Project C1), and the Bavarian Ministry of State for Economics under grant no. 0704/883 25 (EU EFRE funds).
Author's addresses: P. Wägemann, T. Distler, H. Janker, P. Raffeck, V. Sieh, and W. Schröder-Preikschat. Department of Computer Science 4, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Martensstraße 1, 91058 Erlangen, Germany; emails: {waegemann, distler, janker, raffeck, sieh, wosch}@cs.fau.de.

1539-9087/2017/8-ART11 $15.00
https://doi.org/http://dx.doi.org/10.1145/3078631

locations and scenarios where manually replenishing the energy storage is not possible, for example, due to a system being airborne, as it is the case for an autonomous solar-powered quadcopter whose mission is to collect environmental sensor data. Other examples include Google's Loon balloon [14] or Facebook's solar drone Ascenta [11], which both provide access to the Internet in areas where no fibers are established.

While the problem of energy being a scarce resource is not new to the domain of embedded real-time systems, energy-neutral systems introduce new challenges that have to be addressed. For example, energy neutrality requires a real-time system to treat energy as a first-class constraint, equal to timeliness; in fact, there can even be situations in which energy becomes more important than timeliness [32], for example, when there is not enough energy available to keep all system services continuously running. In order to be able to deal with such situations, it is crucial to have a scheduling model that takes into account that some tasks are more critical than others [2, 31], and at the same time also considers the different time and energy constraints of different tasks. *Challenge 1: Energy-neutral systems create the need for a scheduling model that offers the flexibility to handle both mixed criticalities as well as mixed constraints at task level.*

Scheduling tasks (e.g., computations, communications) with mixed constraints makes it necessary for an energy-neutral system to not only enforce timing deadlines but also energy budgets. Unfortunately, while determining task execution times via timers is usually straightforward, state-of-the-art processors lack mechanisms to monitor the energy consumption of tasks [37]. Even worse, many existing battery-operated platforms are not able to provide accurate information about the amount of energy currently available and consequently impede scheduling decisions. *Challenge 2: Energy-neutral systems require mechanisms to obtain fine-grained information about consumed and available energy budgets.*

Probably the greatest difference between energy-neutral and other battery-operated systems is their behavior in situations where a system has drained its energy storage. While other systems at this point reach the end of their lifetime, an energy-neutral system switches into a sleep mode and wakes up again as soon as enough energy has been harvested to resume system operations. For this purpose, an energy-neutral system needs to provide means to safely store its state to persistent memory after having detected that energy is low. Furthermore, the system must have detailed knowledge about the energy consumption of both tasks as well as suspend/resume procedures in order to guarantee that it only wakes up if enough energy is available to operate for at least a certain amount of time (e.g., a minimum number of hyperperiods). *Challenge 3: Energy-neutral systems must be able to ensure consistency across blackout periods.*

In this article, we address the challenges identified above with EnOS, an operating-system kernel for energy-neutral real-time systems. In contrast to existing scheduling models [31, 32], EnOS considers independent criticality levels for time and energy by introducing different energy criticality modes. Relying on this scheduling approach that takes both mixed resources (i.e., time and energy) as well as mixed criticalities (i.e., low and high) of tasks into account, the resources of tasks with lower criticality are potentially reallocated to tasks with higher criticality in order to guarantee their execution in time or without exceeding given energy budgets. To solve the associated task scheduling problem, each energy criticality mode executes a unique set of tasks, which differs from the task sets of other modes. This flexibility allows EnOS to favor timeliness over energy during normal-case operation when the energy available is sufficient to execute all tasks, and to favor energy over timeliness when energy is low. Our evaluation reveals that the decoupling of time and energy constraints and the use of distinct energy criticality modes besides time criticality levels also enables more optimistic schedules in phases where energy is less critical.

To obtain accurate and timely information about energy-related events, EnOS relies on a signaling mechanism partly implemented in hardware: *energy-triggered interrupts*, or *energy interrupts* for short. The most important use case of the energy-interrupt mechanism in EnOS is monitoring the battery's state of charge and notifying the kernel when a specified threshold is reached. Such functionality is essential as it enables EnOS to adjust the functionality provided by the system (and consequently its energy consumption) to the amount of energy actually available. In particular, energy interrupts assist EnOS in deciding when to switch energy criticality modes.

Energy interrupts also play a crucial role in ensuring that when the system enters a blackout period there is sufficient energy left to not only initiate but also complete the suspend procedure that saves the system state to persistent memory. The suspend procedure is started as soon as the amount of energy available falls below a certain threshold, which is determined by exploiting a priori knowledge on the worst-case energy consumptions of tasks [18, 33]. After the end of a blackout, when energy can be harvested again, EnOS does not resume system operations right away. Instead, it programs the hardware to trigger an energy interrupt if enough energy becomes available for the system to at least provide a specific range of functions for a configurable amount of time. This way, EnOS prevents the system from trying to resume every time a tiny amount of energy is harvested.

In summary, our main contributions in this article are:

(1) We present a scheduling model that considers mixed criticalities as well as time and energy constraints in a leakage-aware environment.
(2) We introduce the concept of energy interrupts and provide details on hardware techniques that enable EnOS to enforce energy budgets.
(3) We discuss the mechanism that allows EnOS to keep its system state consistent across blackout periods.
(4) We provide a comprehensive evaluation of the EnOS implementation using both measurements and simulations of the hardware and the environment as well as analyses of our scheduling approach.

The remainder of this article is structured as follows: Section 2 describes two application scenarios for energy-neutral real-time systems in general and for EnOS in particular. Section 3 gives an overview of EnOS and its system model. Section 4 details mixed-criticality scheduling in EnOS. Section 5 presents energy interrupts and Section 6 the suspend/resume mechanism. Our implementation is outlined in Section 7 and evaluated in Section 8. Finally, Section 9 discusses related work and Section 10 concludes. A preliminary version of this work was published at RTAS '16 [34].

2 APPLICATION SCENARIOS OF ENERGY-NEUTRAL REAL-TIME SYSTEMS

In this section, we first present two examples of use cases with both timing and energy constraints for which we developed EnOS: Section 2.1 discusses an energy-harvesting solar quadcopter. Section 2.2 describes the application scenario of a smart blindman's stick powered by energy-harvesting shoes.

2.1 Energy-Harvesting Quadcopter

For demonstration purposes, we are currently developing an energy-neutral quadcopter that harvests all the energy it consumes through solar panels; the quadcopter is based on the I4Copter platform [30]. A typical mission of such a quadcopter is to autonomously fly a predefined route, which is specified by a sequence of waypoints (e.g., GPS locations), in order to collect sensing data, for example, for soil-profile modeling. For some missions, the quadcopter may also be programmed to land at certain waypoints to fulfill special tasks, such as taking photographs of the environment from ground level. However, this does not necessarily mean that these are the only waypoints

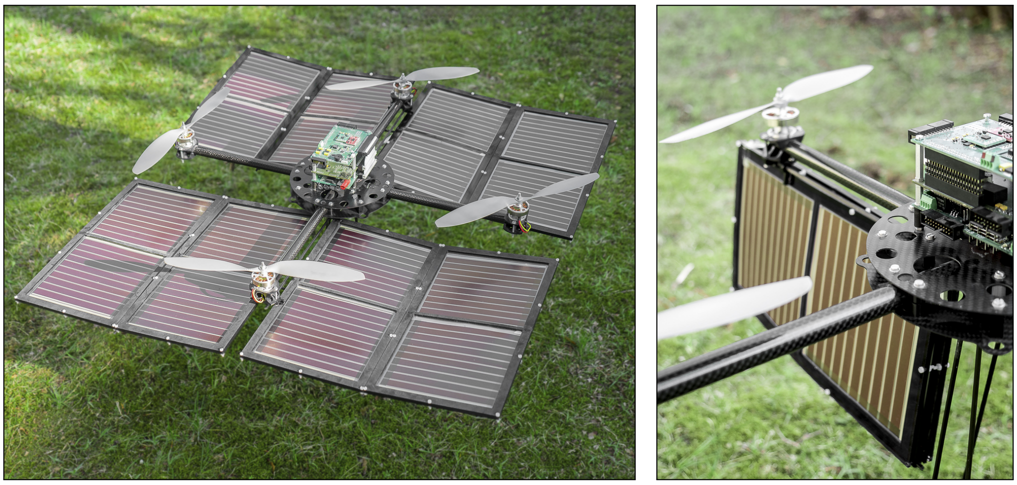


Fig. 1. Energy-neutral quadcopters must respect both timing and energy constraints. However, both constraints are not necessarily critical at the same time and within distinct modes (i.e., stationary, flying) to a different degree of time or energy criticality.

at which the quadcopter actually lands: Due to the capacity of its energy storage not being large enough to power the vehicle for the entire mission, the quadcopter sometimes is forced to make auxiliary landings to recharge its battery. As shown in the left picture of Figure 1, in such case the quadcopter expands its solar panels to maximize energy-harvesting efficiency. In contrast, while the vehicle is airborne, the solar panels are collapsed and positioned under the side arms to avoid disturbing the aerodynamic behavior of the system, as illustrated in the right picture of Figure 1.

During the execution of its mission, an energy-neutral quadcopter goes through different phases with very different characteristics and requirements: While the vehicle is airborne and energy is plentiful, for example, timeliness of position control is highly critical, since missing deadlines of sampling and driving actuators (i.e., rotation speed of motors) is likely to cause crashes. At this point, there is no need to treat energy as the most critical resource. However, this changes significantly once the battery's state of charge requires an auxiliary landing. Then, the system must guarantee that there is enough energy remaining to safely land the vehicle, expand the solar panels, and consistently suspend the running applications; otherwise, the entire mission might be in danger. In such cases, the timeliness of tasks becomes uncritical, as there can be no guarantees of timely execution while the system experiences a blackout.

2.2 Energy-Harvesting Shoes Connected to Blindman's Stick

Analogous considerations about phases where timeliness and energy constraints are both critical, but not at the same time, apply to wearable applications that harvest energy through piezoelectric layers in soles of shoes [27]. This way of harvesting by walking is a possible energy source for smart blindman's sticks [22], which are used to assist visually disabled persons through sonar sensors in traffic. The power transfer between shoes and stick could be achieved via cables. Tasks running in the sticks are able to feature detecting obstacles with sonar sensors, navigation with GPS, or sending GPS position data to remote nursing staff. In this exemplary use-case scenario, reacting in time on possible obstacles approaching the walker (i.e., the detection of collisions) is a task with the highest time criticality. In contrast to collision detection, the transfer of GPS positions

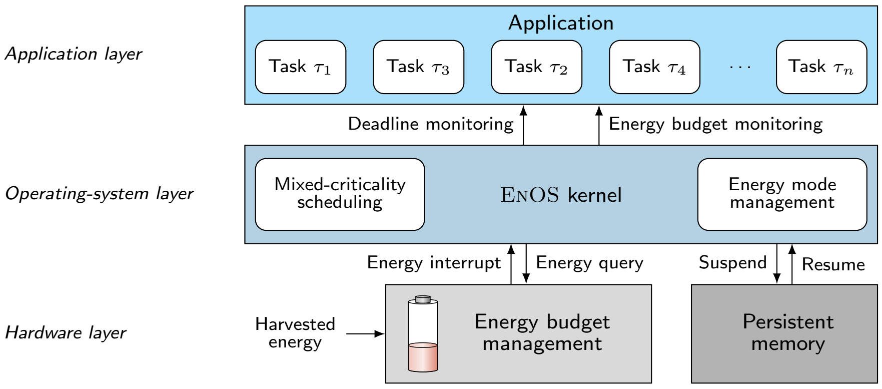


Fig. 2. Interplay of components in EnOS.

should be executed with lower time priority. During the runtime of the system, services (i.e., the feature set of the application) are activated or deactivated depending on the charge level of the battery. If energy drops below a critical level, when the harvested power breaks off to keep the system operational, a task with low timing criticality but high energy criticality must be issued to inform the remote staff about this incident. Consequently, such types of systems have both timing and energy constraints that must be respected.

2.3 Summary

In both described application scenarios, the quadcopter as well as the blindman's stick, situations exist where energy surpasses timeliness and vice versa and the high criticality of one resource at any point in time does not necessarily lead to a high criticality of the other resource. This observation supports the argument for decoupling energy and time constraints within distinct modes.

3 ENOS

In this section, we present the system model of EnOS and give an overview of its basic approach. Details of key components and mechanisms are deferred to later sections.

3.1 System Model

EnOS addresses energy-neutral systems whose main characteristic is the possibility to gather energy from sources such as the sun or movement. Unfortunately, being dependent on the environment means that there are times at which no or only small amounts of energy can be harvested, for example, due to a change in weather conditions or the setting sun. In the general case, such changes occur gradually over time, that is, at the scale of minutes or hours. However, under exceptional circumstances (e.g., a damage of cables), the amount of newly harvested energy can drop instantly from its maximum to zero. An energy-neutral system must be prepared to deal with such situations and suspend system operations, which possibly includes sending an emergency signal, or even abort its mission entirely before running out of energy.

Energy-neutral systems commonly rely on a battery to buffer the harvested energy, which mainly solves two problems: First, it enables an operating system like EnOS to ensure reliable operations for a certain period of time, despite the fact that in general it is impossible to precisely predict when and how much energy will be harvested. Second, the battery allows the energy-neutral system to power applications whose peak power demand exceeds the amount of power the harvesting source is able to provide. Although the capacity of the energy storage is usually large enough to keep operations alive for hundreds or a few thousands of system cycles, this is not sufficient for the entire, potentially endless, mission. As a result, the initially available energy is negligible compared to the overall amount of energy consumed.

Applications executed on top of EnOS have real-time constraints. The nature of energy-neutral systems dictates that hard real-time constraints can only be guaranteed while the system is running; as system operations might be suspended due to a lack of energy, there cannot be a general guarantee that the system will make progress. EnOS supports applications comprising a state that may be modified by tasks at runtime. If the state of an application were to be lost or became inconsistent as the result of a blackout, the entire mission of the system could be affected [21]. In consequence, the system needs to provide a persistent-memory module that enables EnOS to store the state if energy is low.

3.2 The EnOS Approach

Figure 2 shows an overview of the interplay between components in EnOS as well as the services provided by the EnOS kernel. Similar to traditional real-time systems supporting mixed time criticalities, these include the monitoring of timing deadlines. However, in addition, EnOS supports different energy criticalities and consequently also enforces energy budgets. As further discussed in Section 4, EnOS distinguishes between different energy modes, which offer the possibility to adjust the energy consumption of the system to the amount of energy currently available. For decisions on when to trigger a mode switch, EnOS relies on the energy-budget management component with which it interacts in two possible ways: On the one hand, EnOS is able to directly issue a query to retrieve information on the current state of the battery. On the other hand, it may instruct the energy-budget management component to trigger an energy interrupt when the battery has reached a certain state (see Section 5). The latter approach is also used in the context of the suspend/resume mechanism that enables the system to remain consistent across blackouts (see Section 6).

4 SCHEDULING WITH MIXED CRITICALITIES AND MIXED CONSTRAINTS

Below, we first provide background on mixed-criticality scheduling and then detail the approach used in EnOS to handle time and energy criticalities independently.

4.1 Background

Scheduling decisions in energy-neutral real-time systems require a priori knowledge on both the worst-case execution time (WCET) and the worst-case energy consumption (WCEC) of tasks. Unfortunately, precisely determining/capturing the actual WCET and actual WCEC of a task is usually not possible for real-world applications [36]. In general, there are two ways to address this problem: On the one hand, there are static analysis approaches such as the implicit path enumeration technique (IPET) [24] whose results tend to be very pessimistic over-approximations of the actual WCET and WCEC. As a result, scheduling decisions based on such values lead to significant unexploited slack time and unutilized energy resources. On the other hand, optimistic analysis approaches, which are often measurement-based [33, 35], are less costly than pessimistic analyses but may under-approximate the WCET and WCEC. Therefore, it cannot be ruled out that a task misses its deadline or exceeds its energy budget.

The concept of mixed-criticality scheduling proposed by Vestal [31] for real-time systems is able to mitigate this dilemma. It builds on the basic idea of reacting to emergency situations by redistributing the resources of tasks with lower criticalities to tasks with higher criticalities. In particular, each task is assigned a criticality level and only executed if this level is greater or equal than a global system criticality level. Initially, the system starts with the lowest criticality level, meaning that all tasks are scheduled. If at any time a task misses its deadline, which may happen through a lower criticality task with an under-approximated WCET estimate, the system increases its criticality level, causing tasks with lesser criticality levels to be discarded. In this approach, the time budget granted to a task depends on the current system criticality level: The higher the system criticality level, the more pessimistic WCET estimates are used to select time budgets. As a result, costly WCET analyses can be limited to highly critical tasks, as only they are executed at higher system criticality levels. For all other tasks, it is sufficient to determine optimistic WCET estimates.

The term *mixed criticality* in this paper refers to time or energy criticality and is consequently not directly related to safety criticality. As Burns and Davis point out in their review on mixed-criticality systems [10], this term can be seen as umbrella term for various older scheduling approaches. For instance, the work of Lehoczky et al. considers dual-criticality systems with soft and hard tasks [20]. In these systems, in contrast to soft tasks, the timely execution of hard tasks is guaranteed. Soft real-time tasks are scheduled in a best-effort strategy, for example, to reduce their response time.

Völp et al. [32] showed that for energy-constrained systems it is crucial for the scheduling model to take energy into account, otherwise mixed-criticality guarantees could be violated. In their approach, the system criticality level not only determines the degree of confidence that is necessary for a task's execution-time estimate but also for its energy-consumption approximation.

4.2 Approach

The scheduling model of EnOS builds on the works presented in the previous section, but in contrast to the approach by Völp et al. considers dedicated *energy criticality modes* (or "energy modes" for short) that are independent of time criticality levels. The rationale behind this design decision is the observation that in energy-neutral systems time and energy constraints are both equally important, but not necessarily at the same time. For example, during normal-case operation when sufficient energy is available, guaranteeing timeliness is essential. On the other hand, when an energy-neutral system is about to run out of energy, the most important task is to safely store its state to persistent memory, independent of how long this procedure takes.

In EnOS, different tradeoffs between time and energy constraints are represented by different energy criticality modes. As shown in Figure 3, in each energy mode the system executes a dedicated set of tasks. This offers the possibility to adjust the functionality provided by the system to the amount of energy currently available. For example, whenever energy is plentiful, a solar-powered quadcopter may run in the lowest energy criticality mode, thereby collecting sensor data, preprocessing it, and transmitting the results to a base station. If due to a change in weather conditions, the amount of energy harvested is no longer sufficient for the quadcopter to operate at full capacity, the system switches to a higher energy criticality mode, in which it might still collect and preprocess data but defer transmission. If conditions improve after some time, the quadcopter can resume full operation, otherwise it continues to gradually deactivate services. In the worst case, the amount of energy available becomes so low that the quadcopter needs to abort its mission and initiate an emergency landing. For such purposes, EnOS provides an *emergency mode* from which, once entered, the system will never return. However, not all energy-neutral systems require such a mode, for example, due to being stationary and therefore able to survive with all services being switched off for a long period of time. For these use-case scenarios, EnOS offers a *suspend mode* that on entry stores the system's state to persistent memory and also handles the wakeup when external conditions improve (see Section 6).

At low and medium energy criticality modes, in which time constraints predominate, EnOS applies the mixed-criticality approach proposed by Vestal (see Section 4.1) to guarantee timeliness.

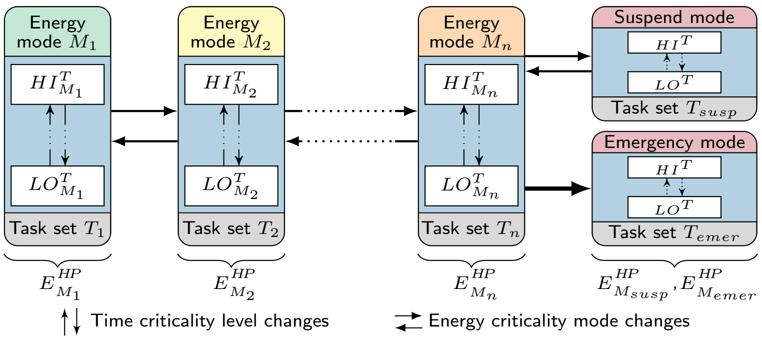


Fig. 3. The scheduling model applied by EnOS comprises different energy (criticality) modes, in each of which a dedicated set of tasks is executed. Within an energy criticality mode, the system time criticality level can vary between low (LO^T) and high (HI^T).

Nevertheless, even in these modes energy constraints are of importance: At all times, EnOS monitors the amount of energy currently available to ensure that there is sufficient energy left to switch to higher energy criticality modes and eventually reach the suspend or emergency mode in case it becomes necessary. The decisions on when to switch are based on energy-consumption estimates for hyperperiods, which are calculated for each energy criticality mode independently considering the WCEC estimates of the respective tasks (see Section 4.4 for details). While for low energy modes optimistic analysis techniques are applied to determine energy-consumption estimates, more conservative techniques are used for higher energy modes. This approach offers the key benefit of costly energy-consumption analyses only being required for the few tasks that are executed in the highest energy criticality modes, in which the system provides a limited functionality.

4.3 Mode-Specific Task Sets

A task set T in EnOS consists of implicit deadline-constrained, sporadic, mixed-criticality tasks; each task is independent of every other task in the set. A task $\tau_i \in T$ is defined by the quadruple $\tau_i = (D_i, C_i, L_i^T, E_i)$, where D_i denotes the deadline as well as the minimum inter-release time between subsequent jobs of the task. The vector C_i denotes the execution-time estimates for different system time criticality levels (see Section 4.1) and L^T the time criticality level of the task. For better readability, we use the constants LO^T and HI^T to denote the lowest and highest time criticality level, respectively. Due to not only taking time constraints but also energy constraints into account, for each task EnOS furthermore relies on an energy-consumption estimate E_i, which is determined using either optimistic or pessimistic analysis approaches, dependent on the energy criticality mode. All system overheads, such as task switches or timer interrupts, in contrast are modeled in a pessimistic way, independent of the energy criticality mode. The hyperperiod denotes the amount of time equal to the least common multiple of the minimum inter-release time D_i of all tasks τ_i of a task set. The busy period, however, marks the maximum possible continuous time interval where the processor is not idled.

The task sets of different energy modes in EnOS do not necessarily have to be disjoint. Instead, a task may be executed in more than one energy mode. With the most energy being available at the lowest energy criticality mode, in general this mode provides the most services. For higher energy criticality modes, more and more services are deactivated, which typically results in their task sets becoming smaller. However, this does not mean that the task set of an energy mode is always a subset of the task sets of lower energy modes. In fact, it is usually a good idea to implement the handling of low-energy situations in separate tasks and to only include them in the task sets of higher energy modes. Overall, the use of mode-specific task sets has two major benefits: First, it allows tasks to be executed only when they are actually needed. As a consequence, the size of task sets is minimal, which greatly improves schedulability. Second, it offers the possibility to use different implementations of the same service at different energy criticality modes. For example, when energy is plentiful, sensor data can be preprocessed with the algorithm that provides the highest precision, while at higher energy modes the algorithm with the lowest energy consumption is selected.

4.4 Switching Between Energy Criticality Modes

Managing time and energy criticalities independently offers EnOS the flexibility to treat both time and energy as first-class constraints and still favor one over the other when it becomes necessary. Of key importance in this context is the mechanism that allows EnOS to switch between energy criticality modes in order to adapt the energy consumption of the system to the amount of energy currently available. If external conditions change slowly (i.e., at the scale of minutes or hours), which we consider the normal case, such mode switches occur rather infrequently, resulting in the system to remain in the same energy mode for many consecutive hyperperiods, which usually take in the order of tens or hundreds of milliseconds. In contrast, under exceptional circumstances it may be required to switch energy modes more quickly. However, even in such cases EnOS guarantees that each energy mode entered is executed for at least an entire hyperperiod and that a switch only occurs when the processor is idle in order to allow the system to degrade gracefully in a safe way.

4.4.1 Basic Mechanism. EnOS initiates an energy-mode switch when the amount of energy stored in the battery falls below (for increasing the mode level) or exceeds (for decreasing the mode level) predefined thresholds:

$$E_{switch}(i) = E^{HP}_{susp/emer} + \sum_{i \leq j \leq |M|} E^{HP}_{M_j,max} \tag{1}$$

In the basic case, the thresholds E_{switch} are calculated using Equation 1, in which $|M|$ describes the total number of modes (not including the suspend and emergency modes) and E^{HP} an energy-consumption estimate of the hyperperiod in a particular energy mode. Due to uncertain release times, tasks may still be executing at the end of a hyperperiod in sporadic systems. This estimate therefore needs to include the maximum possible overlap at the end of a hyperperiod, which is equal to the duration of one busy period. In this context, $E^{HP}_{susp/emer}$ either represents the value for the suspend mode or the emergency mode, depending on which of these alternatives is configured in the system (see Section 4.2). To ensure that there is sufficient energy available to complete the suspension/emergency procedure, $E^{HP}_{susp/emer}$ is an over-approximation determined by energy analyses performed with the *0g* WCEC analyzer [33]. For all other energy modes, EnOS relies on a mode-specific *grace budget* $E^{HP}_{M_j,max}$ to guarantee the execution of at least one hyperperiod of each mode. $E^{HP}_{M_j,max}$ is an upper bound based on the mode-specific duration of a hyperperiod, including the maximum possible overlap of one busy period, and the maximum physical power consumption of the hardware (see Section 7.3). In summary, by considering only pessimistic energy-consumption estimates for the computation of thresholds, EnOS can guarantee that even under exceptional circumstances, the system is able to reach and complete the suspend or emergency mode.

A mode switch from an energy criticality mode M_m to the next higher mode M_{m+1} occurs if the amount of energy available falls below the threshold $E_{switch}(m)$. In contrast, a switch from mode M_m to the next lower energy criticality mode M_{m-1} is triggered if the threshold $E_{switch}(m-1)$ is

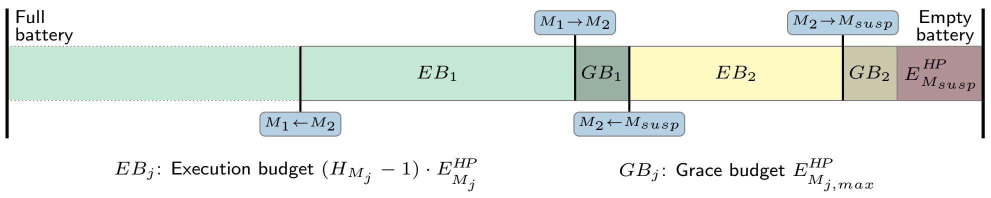

Full battery
Empty battery
EB_1
GB_1
EB_2
GB_2
$E^{HP}_{M_{susp}}$
$M_1 \rightarrow M_2$
$M_2 \rightarrow M_{susp}$
$M_1 \leftarrow M_2$
$M_2 \leftarrow M_{susp}$
EB_j: Execution budget $(H_{M_j} - 1) \cdot E^{HP}_{M_j}$
GB_j: Grace budget $E^{HP}_{M_{j,max}}$

Fig. 4. Mode-switch thresholds for an example system with three different energy criticality modes M_1, M_2, and M_{susp}.

exceeded. In order to learn about thresholds being surpassed, EnOS sets up an energy interrupt (see Section 5). If the energy interrupt fires during the execution of a task, EnOS continues to execute the task set of the current energy criticality mode until the end of the current busy period is reached, that is, until the processor is idle, and then performs the mode switch. As a result, highly time-critical tasks are always able to run to completion. After a mode switch, the new energy criticality mode starts with the lowest system time criticality level.

4.4.2 Extending the Basic Mechanism with Execution Budgets. As an extension to the basic mechanism presented in the previous section, EnOS allows system developers to configure mode-switch decisions by indicating how many hyperperiods H_{M_j} should be executed in each energy criticality mode M_j. By this means, it is possible to control the thresholds for mode switches, and consequently to customize the degradation process.

As illustrated in Figure 4, the extended mechanism not only relies on grace budgets but also considers mode-specific *execution budgets*. In contrast to the pessimistic grace budgets, the execution budgets for lower energy criticality modes are determined by optimistic, measurement-based energy analyses. Such analyses provide close under-approximations of the actual WCEC of a hyperperiod and offer the key benefit of being less costly than static code analysis approaches [33]. While a grace budget targets the completion of a single hyperperiod, an execution budget aims at remaining in an energy mode for at least $H_{M_j} - 1$ hyperperiods. Computing mode-switch thresholds based on a combination of both grace budgets and execution budgets consequently allows EnOS to take into account the number of hyperperiods H_{M_j} specified by the system developer for an energy mode M_j. In the extended mechanism, the mode-switch thresholds E^*_{switch} are computed using Equations 2 and 3 by adding the grace-budget-based thresholds E_{switch} of the basic mechanism to the sum of execution budgets E_{exec}; $E^{HP}_{M_j}$ is the measurement-based estimate of the energy consumption of a hyperperiod in energy criticality mode M_j [32].

$$E^*_{switch}(i) = E_{switch}(i) + \begin{cases} E_{exec}(i) & \text{for } M_m \rightarrow M_{m+1} \\ E_{exec}(i-1) & \text{for } M_m \rightarrow M_{m-1} \end{cases} \tag{2}$$

$$E_{exec}(i) = \sum_{i<j\leq|M|} (H_{M_j} - 1) \cdot E^{HP}_{M_j} \tag{3}$$

Figure 4 shows that the thresholds for switching to higher energy criticality modes in the extended mechanism differ from the thresholds at which EnOS changes to lower energy criticality modes. The rationale behind this approach is to prevent the system from frequently changing energy modes due to the amount of energy available fluctuating around a mode-switch threshold. EnOS achieves this by applying the following strategy relying on different thresholds: When energy becomes scarce, the system remains in its current mode as long as possible before switching to a higher mode. On the other hand, when the battery charges, EnOS delays the switch to a lower energy mode until there is enough energy to execute the new mode M_j for a certain number of hyperperiods H_{M_j}.

Note that the number of hyperperiod execution cycles H_{M_j} configured by the developer is only an advice to EnOS on when to trigger an energy-mode switch. If the actual energy consumptions during runtime never exceed the optimistically determined WCECs, H_{M_j} represents the minimum number of hyperperiod executions in mode M_j. In particular, the system may remain longer in an energy mode in case external conditions improve after having entered the mode. On the other hand, if the execution budgets determined for lower energy modes turn out to be too optimistic for the given circumstances, the system may switch to a higher energy criticality mode early. Nevertheless, even then the pessimistic grace budgets ensure that there is sufficient energy available to execute at least one full hyperperiod in each energy criticality mode. Additionally, as the actual arrival time of sporadic jobs is not known a priori, the number of execution cycles H_M is not able to indicate the actual amount of work performed. Instead, it denotes the maximum amount of work that can possibly be performed, if the jobs actually arrive at their respective minimum inter-release time.

4.4.3 Introducing Leakage Awareness. Leakage currents are a significant issue for battery-operated systems [38]. This is especially true for supercapacitor-powered platforms, such as the current EnOS hardware prototype (see Section 7.1), which must respect leakage in order to be able to guarantee precise hyperperiod energy budgets $E^{HP}_{M_j}$. We therefore extend the computation of $E^{HP}_{M_j}$ with an additional term modeling the energy loss through leakage. The leakage energy during the hyperperiod is modeled with the worst-case leakage-power consumption $P_{leak,max}$ multiplied by the duration of the hyperperiod HP_{M_j} (including one busy period interval) on mode M_j, as shown in Equation 4.

$$E^{*,HP}_{M_j} = E^{HP}_{M_j} + P_{leak,max} \cdot HP_{M_j} \quad (4)$$

The value for the duration of the hyperperiod is computed from the task set's parameters, whereas the value of $P_{leak,max}$ can be deduced from the data sheet of the utilized component and the power characteristics of the employed platform [12].

4.4.4 Discussion. Energy-mode switches in EnOS only occur during idle intervals, even if an energy interrupt triggers during a busy period (see Section 4.4.1). This design decision has been made for mainly two reasons: First, with hyperperiods in EnOS often taking only tens or hundreds of milliseconds, switching between busy periods still allows the system to react to changes in environmental conditions quickly, compared to the minutes or hours applications on EnOS usually remain in the same energy mode. Second, reconfiguring the system during the execution of a job would require additional measures to ensure safety. This is especially true if the reconfiguration involves a switch to another task set, as it might be the case in EnOS (see Section 4.3). In contrast, performing an energy-mode switch safely is straightforward at the end of a hyperperiod when no tasks are running and the system is in a consistent state.

In order to be able to guarantee that in any case the system has enough energy to complete the current busy period before switching the energy mode, the thresholds used to set up energy interrupts in EnOS include a grace budget for the current mode (and all higher modes, respectively), that is, a pessimistic estimate of the energy consumption of a single hyperperiod. As discussed in Section 4.4.1, the size of the grace budget depends on the characteristics of the hardware platform and is determined so that it is physically impossible for the system to consume more energy while completing the rest of a hyperperiod, independent of when exactly the energy interrupt triggers.

This approach allows EnOS to ensure that enough energy is available to switch modes without requiring costly energy analyses for lower energy criticality modes.

4.4.5 Application-Triggered Energy-Mode Switching. As the operating system, EnOS has no deeper knowledge of the nature of the application and consequently may not always be able to decide which energy criticality mode is currently the most suitable mode to execute. Without additional information, EnOS would therefore be limited to triggering energy-mode switches based on its assessment of the amount of energy currently available. To circumvent this limitation, EnOS offers the possibility to rely on the application for mode-switch decisions. This way, the application, for example, can instruct EnOS to skip a certain number of energy modes instead of waiting for the operating system to perform multiple single-step mode switches.

For this purpose, EnOS provides an interface allowing the application to request switches to arbitrary energy criticality modes. As EnOS knows the worst-case energy consumption of every mode, it can verify if a requested switch to a lower energy criticality mode is possible and consequently deny that switch if the available energy is insufficient. In contrast, switching to a higher energy mode can always be performed safely. However, in such case additional measures need to be taken in order to prevent EnOS from immediately switching again due to automatically detecting that there is sufficient energy available to continue execution in a lower mode, thereby negating the switch requested by the application. Therefore, after a switch to a higher energy criticality mode by the application, EnOS ceases switching to lower energy modes until the application explicitly allows it.

One example of a system benefiting from application-triggered energy-mode switches is the energy-harvesting quadcopter presented in Section 2.1. Here, the switching mechanism described above enables the application to request a mode switch if the mission enters a new phase. For example, when the quadcopter arrives at a waypoint on its route at which it is supposed to land, the application can instruct EnOS to switch to the energy mode implementing the tasks for the landing procedure. Not surprisingly, this approach is significantly more efficient than the alternative: remaining airborne until the amount of energy available in the battery drops to the threshold at which EnOS autonomously initiates an emergency landing.

5 ENERGY INTERRUPTS

Considering time constraints as well as energy constraints requires means for monitoring both timing deadlines and energy budgets. Most embedded hardware platforms are equipped with numerous general-purpose timers with high accuracies, which can be exploited to monitor timing deadlines. However, commercial off-the-shelf platforms do not provide equivalent hardware features for reacting on energy-related events. Although some embedded platforms exist that offer the possibility to poll the energy budget available using suitable sensors, this technique does not support the monitoring of thresholds, which is essential for building a lightweight energy-mode switching mechanism in EnOS. To circumvent this problem, we utilize an interrupt-oriented approach that is partly implemented in hardware: energy interrupts. In the following, we first present a basic overview of how energy interrupts are realized in EnOS and then discuss hardware-related details.

5.1 Approach

In order to be notified when the state of charge of the battery reaches a certain threshold, EnOS instructs the hardware component to set up an energy interrupt and then immediately continues to execute the application. As shown in Figure 5, to set up an energy interrupt, the kernel forwards the required energy budget as well as the threshold type (i.e., whether it is an upper or a lower threshold; omitted in the figure) to a special hardware component capable of accessing the system's

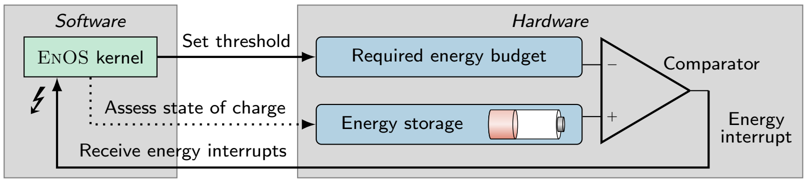


Fig. 5. EnOS uses mechanisms for energy interrupts to enforce energy budgets and to assess the current energy budget.

energy storage. At this point, the hardware starts to continuously compare the configured value with the amount of energy actually available. Depending on the threshold type, if the battery's state of charge exceeds or falls below the threshold, the hardware triggers an interrupt, which is then handled by EnOS. In addition to being notified about energy-related events via interrupt, EnOS is also able to synchronously assess the current state of charge by polling the hardware.

5.2 Practical Considerations

To implement precise energy interrupts in practice, it is crucial to be able to accurately assess the amount of energy available in the battery. For this purpose, EnOS currently exploits electric double-layer capacitors (i.e., supercapacitors) as energy storage. Supercapacitors provide several advantages compared to traditional lithium-ion batteries: For example, supercapacitors have a monotonic relationship between the voltage over the capacitor and the energy stored and can therefore be modeled as ideal capacitors [26]. As a consequence, precise digital-to-analog converters in combination with comparators can be used to set the required budget. The analog value of this budget and the voltage over the supercapacitor form the inputs to a comparator whose output then indicates the energy interrupt. Furthermore, charging a supercapacitor can be achieved by simply limiting the input voltage, making additional charging circuitry unnecessary.

Buchli et al. [6] have shown that the state-of-charge assessment under varying operation conditions (i.e., aging-effects, temperature influence) is feasible within acceptable tolerances (i.e., $\leq 5\,\%$). To compensate the remaining imprecision and to ensure that enough energy is available in the system, EnOS makes pessimistic assumptions when assessing the current state of charge. Furthermore, the system monitors energy budgets at the granularity of energy modes; this is practical considering the precision of our current energy-interrupt implementation (see Section 8.3).

6 SURVIVING BLACKOUTS

For many energy-neutral systems it is impossible to guarantee a continuous flow of energy, for example, because they harvest energy via solar panels. In such cases, it is essential to safely suspend operations once a blackout occurs and to resume when energy becomes available again. Below, we detail how this is supported by EnOS.

6.1 Suspension & Resumption

As discussed in Section 4.2, EnOS provides a dedicated energy criticality mode, the suspend mode, that encapsulates the functionality that is necessary for the energy-neutral system to survive sustained blackouts; the suspend mode is assigned the highest energy criticality. In contrast to other energy modes, while in suspend mode, the system no longer considers time constraints but only focuses on safely making its state persistent, independent of how much time this procedure takes. For this purpose, EnOS relies on ferroelectric RAM (FRAM), which provides acceptable access latencies, low static power consumption, and long durability [21].

During the suspension procedure, the system needs to save all volatile state that is required either by the application or by EnOS itself in order to resume execution from the point of suspension as soon as enough energy can be harvested again. In general, the state to store includes all data segments of the application and the kernel as well as the system's hardware state (e.g., calibration values of peripherals).

After a blackout period is over, the system begins to recharge its battery by harvesting energy. EnOS offers developers the possibility to configure the wakeup point by selecting the energy mode at which system operations are to be resumed. Before going to sleep, the system sets up an energy interrupt for the threshold of the specified energy mode. The predefined threshold for this energy mode is computed by the number of hyperperiods for each mode and the respective hyperperiod energy budget (see Section 4.4.2). Depending on the application scenario, developers state a higher energy criticality mode when the system is supposed to receive further energy during operation, which leads to switches to lower criticality modes. In contrast to this, systems that receive only comparatively small energy budgets during operation might resume their operation only in modes where energy is plentiful. When the battery's state of charge exceeds the threshold, the hardware triggers an interrupt, as a result of which EnOS resumes execution.

6.2 Determining the Suspend-Mode Energy Budget

To ensure consistency, it is crucial that the system does not run out of energy while writing its state to persistent memory. As a consequence, it is essential for EnOS to reserve an energy budget that is equal to or larger than the worst-case energy consumption of the suspend mode. To determine the size of this energy budget, we perform static code analyses [33] on the task set executed in the suspend mode. Such analyses require fine-grained energy-cost models comprising information on the energy consumption of hardware components, including the persistent memory.

Unfortunately, determining safe over-approximations for FRAM accesses is not straightforward: First, in contrast to timing behavior, the energy characteristics of a platform are usually not accurately documented; this is especially true for fine-grained information such as the energy consumption of single read/write accesses, since the behavior highly depends on the actual wiring of the platform. Second, the energy consumption of a memory operation in some cases depends on the actual values of the data to be stored, as the number of low-level hardware components involved varies for different inputs [23]. We solve both issues by conducting extensive measurement-based analyses. The foundation for these analyses is a tool automatically generating micro-benchmarks, which are executed on the target platform. The measurement results are then obtained relying on a precise energy-measurement device that is further described in [16]. The benchmark generator considers different sizes of both read and write accesses to the FRAM memory as well as different values for these accesses. The gathered energy-cost models are then utilized in the static analysis of the suspend mode's energy consumption.

7 IMPLEMENTATION

In this section, we present details of our current EnOS implementation (depicted in Figure 6) and discuss how we determine grace budgets for energy consumption.

7.1 Hardware Components

Solar cells (①), which are able to provide up to 5.6 W, serve as energy source for the energy-neutral system. The cells are hooked up to an energy-harvesting circuit (②) that either steps up or down the input voltage to charge supercapacitors (③) storing the harvested energy. A switching voltage regulator (④) provides a constant input voltage for the embedded computing platform, an NXP

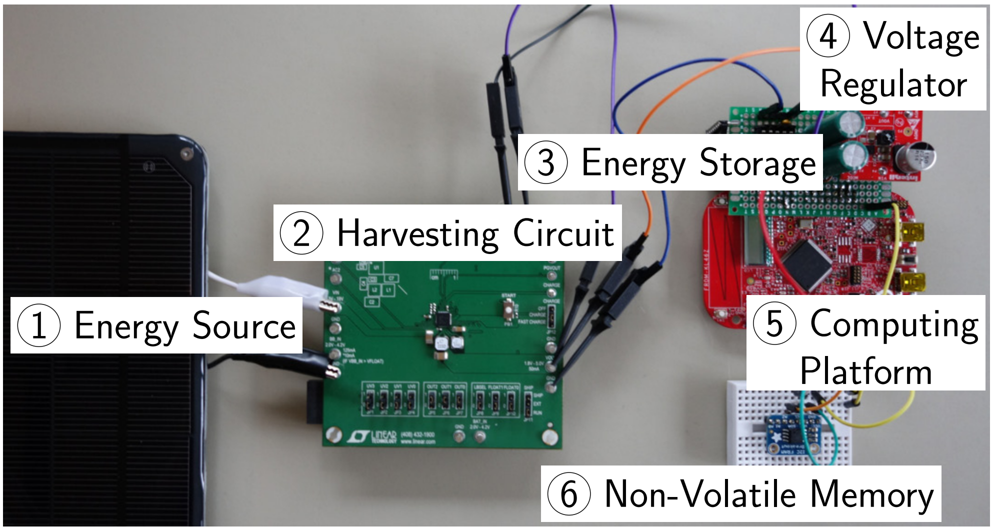


Fig. 6. The EnOS operating system requires a tightly coupled interaction between hard- and software.

Freedom KL46Z board (⑤), which features a Cortex-M0+ processor. The platform uses an external non-volatile FRAM chip (⑥) to make the system state persistent when not enough energy is available to continue operation. FRAM is a suitable technology for this application scenario due to the fast read and write access latencies, the data retention (up to 200 years), and the sufficient read/write endurance (up to 10^{12} times per byte) [13]. Besides the low dynamic power consumption, the FRAM component has another important advantage: It can be connected to a general input/output pin that allows to power the component only when it actually needs to be utilized, that is, for the duration of writing the system state to memory.

The supercapacitors are charged using a balancing circuit, which is integrated into the energy-harvesting step-up/-down converter (LTC3331), up to 5 V. Thus, the overall energy in the system can be approximated using the equation $E = \frac{1}{2}CU^2$. As the voltage is read using an internal 12-bit analog-to-digital converter, up to about 30 J can be stored in the system. For comparison, this is enough to suspend and resume the system more than 3,900 x with up to 10 KiB of memory. Energy interrupts are implemented using a digital-to-analog converter and a comparator both featured by the board.

When using supercapacitors as energy storage, the influence of leakage currents cannot be disregarded in general (see Section 4.4.3). In the following, a hyperperiod of 100 ms is considered (as also used in the evaluation in Section 8.2). Our prototype platform is equipped with a total capacity of 5 F and both supercapacitors have a total leakage of 40 μA [12]. Taking this into consideration and using Equation 4, the additional energy for one hyperperiod is 20 μJ, about 0.4 % of the total hyperperiod energy budget. Consequently, the effect of leakage can be considered as minor in the current prototype during the runtime of around 10 min assuming no energy is harvested from the environment. Nevertheless, when utilizing larger capacitors the leakage parameters are more significant and the extension of Section 4.4.3 to the energy-budget calculation solves the problem of underestimating budgets due to leakage.

7.2 WCET & WCEC Analyses

EnOS requires knowledge on the energy consumption of the system in different energy modes. For this purpose, we rely on the *0g* WCEC analyzer [33], which we extended for EnOS. *0g* provides two different kinds of energy-consumption estimates: On the one hand, *soft energy budgets* are determined by means of genetic algorithms and represent close under-approximations of the actual WCEC; consequently, they can be used in EnOS to compute execution budgets for the mode-switch mechanism (see Section 4.4.2). On the other hand, *hard energy budgets* are based on analyses with the implicit path enumeration technique (IPET) [24] and represent safe over-approximations of the actual WCEC; as such, they play an important role in determining the energy budget of the suspend mode (see Section 6.2).

In order to utilize *0g* for EnOS, we had to extend the energy-cost model of the hardware platform, which provides information on the energy consumption of different operations and components. In particular, we determined the energy consumption of non-CPU-internal components such as I^2C communication, usages of direct-memory-access features, the setup of the comparator and the digital-to-analog converter, accesses to persistent memory (see Section 6.2), and static power consumption. To determine these values, we implemented a benchmark generator that allowed us to conduct extensive measurements on the hardware platform using a wide spectrum of micro-benchmarks.

0g has originally been designed to only provide energy-consumption estimates, which is why for EnOS we also had to extend the tool with an instruction-level execution-time model of the embedded computing platform in order to yield WCET results for tasks. Similar to energy estimates, *0g* now uses genetic algorithms to determine soft budgets for execution times, while hard budgets are based on IPET analyses. We used the approach described in [29] to generate both instruction-level cost models (for execution time and energy consumption), which we integrated into *0g*.

Our prototype distinguishes between two criticality levels for time (low and high) and three criticality levels for energy (low, medium, and high). To introduce additional (medium) levels based on budgets provided by *0g*, it is possible to exploit a characteristic of genetic algorithms: such algorithms trade off analysis time for precision, which means that more accurate results can be obtained by prolonging execution.

7.3 Computing Grace Budgets

EnOS guarantees that even in case of an instant blackout, there is enough energy available to execute the taskset of each higher energy criticality mode once (see Section 4.4). This is possible, because when computing the thresholds for energy-mode switches, the system considers the maximum amount of energy that could be consumed during the execution of the respective taskset at each higher energy mode (i.e., the grace budgets $E^{HP}_{M_j,max}$ in Equation 1). Note that one possibility to determine these mode-specific grace budgets is to perform a full worst-case energy-consumption analysis of the corresponding task sets. However, while such an approach is feasible for a single energy mode (i.e., the suspend/emergency mode, see Section 6.2), applying it to all energy modes in the system is too expensive. With the exception of the highest energy mode, we therefore use a different technique to assess grace budgets: We exploit the physical restrictions that are inherent to the hardware of the energy-neutral system.

The current drawn by the overall system is bounded due to physical limitations of the voltage regulator. For example, the switching regulator of the EnOS platform is able to provide currents of up to 150 mA. Furthermore, the regulator's maximum input voltage is 5.0 V, which is dictated by the harvesting circuit. The switching regulator allows for constant efficiency modeling in the range of the typical output load of the complete system [17, 26]. Based on these values and the fixed hyperperiod duration of an energy mode (for which inaccuracies of the system clock need to be considered), it is possible to calculate a grace budget for the mode.

Due to the uncertainty of the arrival times of sporadic jobs, some jobs can still be executing at the end of a hyperperiod. The grace budget therefore has to include not only the hyperperiod, but also the maximum possible overlap at the end of a hyperperiod, that is, the length of one busy period. For instance, for a hyperperiod of 100 ms and a busy period of 50 ms the budget is about 122 mJ, which is less than 0.41 % of the capacity of the energy-storage component (i.e., about 30 J, see Section 7.1). To put this number further into perspective: The average current drawn from the overall system in the standard run mode of the processor is about 11 mA, resulting in an energy consumption of about 5.5 mJ per hyperperiod for a utilization of 100 %, which is a factor of 22 difference. This means that, in the worst case, the over-approximation of the grace budget causes EnOS to trigger the switch to the next higher energy mode only about three seconds earlier than actually necessary. Note that in strictly periodic systems no overlap at the end of a hyperperiod occurs, which allows EnOS to operate with more optimistic grace budgets.

7.4 Mixed-Criticality Mixed-Constraints Scheduler

The implementation of our mixed-criticality scheduler only comprises about 1,500 lines of C code, without utilities such as lists and drivers for accessing peripherals (e.g., digital-to-analog converter, comparator). In addition, only a limited amount of about 100 lines of assembly code is necessary (i.e., for switching tasks in the scheduler). The scheduling algorithm is priority-based and consists of a combination of EA-OCBP [32] and LPA [15]. Initial priorities are computed offline based on criticality and WCEC and then lazily adjusted at runtime to react to the actual arrival times of the individual jobs[1].

The mode changes between lower and higher energy criticality levels, regardless of whether initiated by the operating system (see Section 4.4) or requested by the application (see Section 4.4.5), are performed using an idle protocol, by switching as soon as no task is ready to run. Both mode changes are therefore bounded, according to the classification by Burns [9], and no tasks of two modes ever overlap in execution. For our current use cases this approach is practical; technically, it would also be possible to rely on more sophisticated mechanisms to perform energy-mode switches [25].

The monitoring of deadlines and thus also the switches to higher time criticality levels are performed on each timer tick (i.e., each millisecond). For switches to lower time criticality levels, we implemented an idle-time protocol [25], which allows EnOS to optimistically return to lower levels as soon as slack time is detected through idling. If necessary, it would also be possible to utilize other level-change protocols [3].

8 EVALUATION

In this section, we present evaluations of our scheduling approach and the benefits of criticality-dependent energy budgets for hyperperiods. Furthermore, we compare the off-line calculation of hyperperiod energy bounds with measurements conducted on our prototype hardware platform and evaluate the state-of-charge assessment. Finally, we demonstrate measurements of the energy-interrupt mechanism and show execution traces of energy-mode switches. In all the following evaluations, a periodic task model is used, that is, all jobs are always released at exactly the minimum inter-release time. This leads to more optimistic budgets, as described in Section 7.3, but does otherwise not infer with the evaluation, as periodic tasks are a special case of sporadic

[1]If the EA-OCBP algorithm is unable to find a feasible priority assignment, the developer is in charge of modifying the task set (e.g., by splitting a long-running task into multiple shorter tasks) in order to facilitate finding a viable schedule.

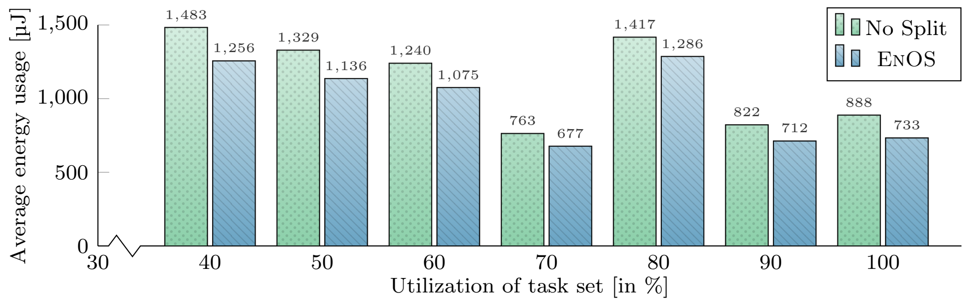


Fig. 7. EnOS yields smaller energy budgets for hyperperiods due to treating energy only in distinct situations as critical.

tasks. A periodic system is used for evaluation purposes as it facilitates reproducing the results in comparison to using highly unpredictable sporadic tasks, which are released at arbitrary points in time between the minimum inter-release time.

8.1 Benefits of Criticality-Dependent Energy Budgets

At lower energy criticality modes, EnOS utilizes optimistic energy-consumption estimates to make scheduling decisions (see Section 4.4.2). In the following, we analyze the impact of this approach on the energy budgets of hyperperiods (E^{HP}) by comparing them to energy budgets determined using the EA-OCBP approach [32]. This algorithm simulates all possible mode transitions during the hyperperiod and computes the maximum energy value from these traces. Unlike EnOS, EA-OCBP does not differentiate between energy criticality modes and time criticality levels; instead, the criticality level of a task influences both the WCET and WCEC estimate.

We synthetically generate 10,000 task sets using the UUniFast algorithm by Bini and Buttazzo [4] with utilization values ranging from 40 % to 100 % (in steps of 10 %). Only mixed-criticality schedulable task sets are considered in this evaluation. The WCEC values are approximated to an average power consumption of 55 mW (i.e., the power consumption in standard run mode without using peripherals, see Section 7.3) multiplied with the chosen WCET values. Note that in general deriving WCEC estimates by multiplying the average power consumption with a WCET value cannot be performed in a safe way [18]. However, for this evaluation, this assumption gives valuable insight about approximative energy savings. Additionally, for the synthetic task set generation, we choose 0.7 as the criticality factor between time criticality levels and consider the two levels *LO* (low) and *HI* (high). The hyperperiods range from 100 ms to 10,000 ms. We use periods that are factors of these hyperperiods (excluding one), as randomly chosen periods might lead to unrealistically large hyperperiods due to prime numbers.

Figure 7 shows the results of the computation of hyperperiod budgets without splitting the time and energy dimension ("No Split") and the budgets used in EnOS for the lowest energy criticality mode. In this figure, the average energy usage describes the arithmetic mean of energy bounds at each utilization level of the feasible task sets; the utilization refers to the utilization of *HI*-tasks. For the energy value, only the dynamic energy consumption is considered (i.e., approximately 2 mA from the 11 mA of total current drawn). The results show that by relying on optimistic energy-consumption estimates for all tasks, EnOS is able to operate with up to 17.5 % smaller energy budgets. Consequently, the thresholds that determine when to perform a mode switch are also smaller, which allows the system to stay longer at lower energy criticality modes.

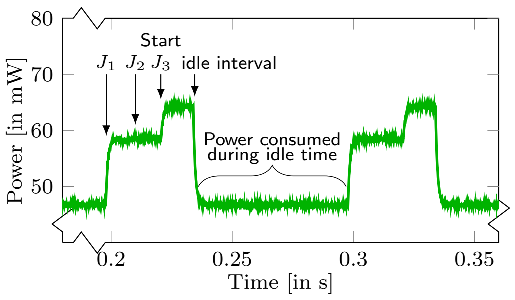


Fig. 8. The power consumption of the considered task set highly depends on the executed jobs within the hyperperiod.

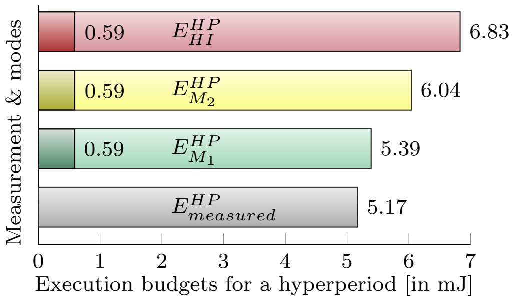


Fig. 9. The over-estimation of statically calculated hyperperiod energy budgets depends on the mode's criticality.

8.2 Precision of Execution Budgets

In the following, we evaluate the precision of execution budgets by comparing the estimates used by EnOS to compute mode-switch thresholds with energy-consumption measurements gained from executions on the real hardware platform. For this purpose, we analyze a set of three tasks τ_i for two different energy criticality modes M_1 and M_2 (see the first two task sets τ_{i,M_1} and τ_{i,M_2} in Table 1). The tasks τ_1 and τ_2 are sorting algorithms, while task τ_3 computes the next prime number of 1201. In all cases the job's release time is at the start of the hyperperiod. One hyperperiod on M_1 and M_2 takes 100 ms. As illustrated in Figure 8, the execution of the different jobs causes the power consumption to vary over time. In addition to the energy consumed during the execution of the respective jobs, the power drawn during idle-time periods (approximately 45 mW) must be respected in the computation of hyperperiod energy budgets. Consequently, we extended the original algorithm described in the EA-OCBP approach [32] to also consider the energy consumed during idle periods in order to be able to compare the budgets with actual measurements performed on the target platform. In addition to idle times, our algorithm also considers both execution-time as well as energy-consumption overheads including, for example, context switches, budget violations, and timer ticks.

For our energy-consumption measurements on the real target platform, we rely on a dual-channel measuring unit, namely the Keithley 2612. The instrument is able to measure minimal currents of 100 fA and minimal voltages down to 100 nV at a temporal resolution of up to 20 µs. The device was instrumented through Keithley's Test Script Processor, to trigger measurements on request of the EnOS kernel using general-purpose input-output pins.

Figure 9 shows the results of this experiment for one hyperperiod. With the worst-case paths being triggered, the mean value of the measured energy consumption is 5.17 mJ and the standard

Task	τ_1, M_1	τ_2, M_1	τ_3, M_1	τ_1, M_2	τ_2, M_2	τ_3, M_2	τ_1, M_{susp}	τ_2, M_{susp}	τ_3, M_{susp}
Time criticality	*LO*	*HI*	*LO*	*LO*	*HI*	*LO*	*LO*	*HI*	*LO*
Energy criticality	*LO*	*LO*	*LO*	*MID*	*MID*	*MID*	*HI*	*HI*	*HI*
WCETs [ms]	13	13; 37	15	13	13; 37	15	13	13; 37	82
WCEC [μJ]	649	649	845	807	807	1183	965	965	3225

Table 1. Task parameters for mixed-criticality mixed-constraints scheduling in EnOS.

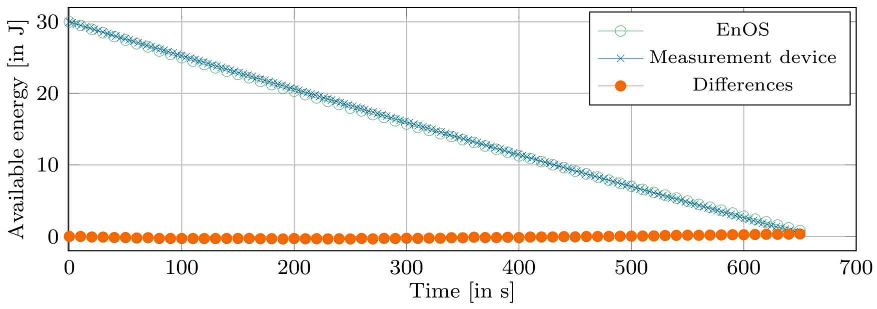


Fig. 10. Available energy determined inside the EnOS kernel compared to an external measurement device.

deviation is 3.6 µJ (250 executions). At the lowest energy criticality mode M_1, EnOS uses an execution budget of 5.39 mJ (including a mode-independent estimate of 0.59 mJ for the system overhead), which is an over-approximation of around 4.3 % of the value actually measured. Due to more pessimistic estimates being used, the execution budget of the energy mode M_2, which for comparison in this experiment executes the same tasks as the energy mode M_1, increases to 6.04 mJ. Note that this execution budget is still significantly smaller than the energy budget $E_{HI}^{HP} = 6.83$ mJ that would be required if the evaluated task set were to be executed in the highest energy criticality mode, which is also the budget existing approaches would use if all tasks had the highest time criticality. In contrast, EnOS is able to rely on smaller execution budgets for the lower energy modes M_1 and M_2, which allows the system to switch back about 21 % and 11 % earlier, respectively, at times when energy is less critical.

8.3 Evaluating State-of-Charge Assessment and Energy Interrupts

We evaluate how precisely EnOS assesses the battery's current state of charge by comparing the values reported by EnOS with the results provided by an external measurement device (Keithley 2612). The experiment starts with a fully charged energy storage and ends when the system has completed the suspension procedure. As shown in Figure 10, EnOS precisely assesses the amount of energy available over the entire range of charge levels.

To evaluate the granularity of detecting energy-budget violations using the energy-interrupt mechanism described in Section 5, we set up the budget interrupt to a predefined lower threshold and detect the error of this setup. The interrupt is detected at a maximum offset of 198 mJ (mean: 94 mJ, 250 executions). Putting this in relation to the overall available energy, it corresponds to a worst-case resolution of 0.6 %.

8.4 Evaluating Energy-Mode Switches

In this section, we evaluate the mode-switching mechanism of EnOS under varying environmental conditions for two target platforms with different characteristics: 1) a simulated small system in which energy-mode switches occur comparably often due to the energy storage being limited to 6 J and 2) the EnOS prototype presented in Section 7 that is able to use up to 30 J to execute tasks. Both systems comprise three energy criticality modes M_1, M_2, and M_{susp}. In mode M_1 and mode M_2, the systems execute the set of tasks that has been used for the experiment in Section 8.2. In contrast, the suspend mode runs two sorting tasks τ_1 and τ_2 handling final computations and a task τ_3 responsible for executing the suspension of the system (see Table 1).

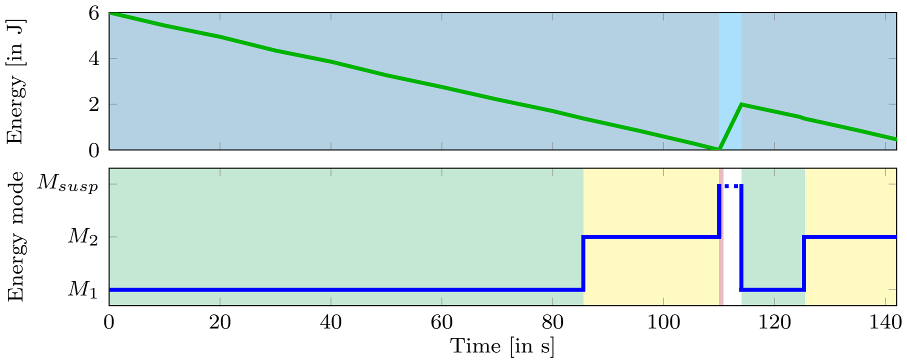


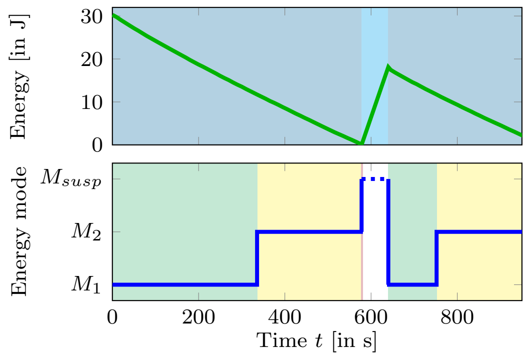
Fig. 11. Energy-mode switches in the small system: At $t = 110$ s the system enters the suspend mode and sleeps until it has harvested enough energy to resume ($t = 114$ s).

Simulation-Based Measurements. To evaluate EnOS with an additional platform besides our actual hardware, we simulate a system that is backed by a small energy storage. For this purpose, we have extended the virtual machine FAUMachine [28] and run a comprehensive simulation that comprises the energy source with configurable amounts of harvested energy, the energy storage, the non-volatile memory, and the entire NXP Freedom KL46Z board. Figure 11 shows the result of this experiment. Initially, the battery is fully charged, which means that EnOS runs in the lowest energy mode M_1. At $t = 0$ s, we switch off the energy source in order to investigate the system's reaction to a blackout. Due to no longer harvesting energy, after some time EnOS is forced to switch to the next higher energy mode ($t = 86$ s) in which it remains for 24 seconds. Then, the system detects that the battery is almost completely drained and consequently decides to initiate the suspend mode ($t = 110$ s).

However, by determining the threshold for this switch based on pessimistic energy-consumption estimates, EnOS can ensure that at this point, there is still enough energy left to execute one hyperperiod in the suspend mode before eventually putting the system to sleep. For this experiment, we have configured EnOS to resume system operations at energy mode M_1. After we reactivate the energy source, at $t = 114$ s EnOS has harvested enough energy to continue in this mode. Finally, another blackout causes the system to once again change its energy criticality mode to M_2 ($t = 125$ s).

Measurements on Hardware. The evaluation on the target platform comprises the whole system consisting of harvesting circuitry, supercapacitors, switching regulator, and the embedded computing platform. An external current source serves as input to the harvesting circuitry. Figure 12 presents the traces gained from this experiment, which show that the EnOS prototype correctly executes the energy-mode switches required to adjust to changing conditions. In particular, this includes the execution of the suspend mode after 579 seconds when energy becomes scarce as well as the resumption of system operations 57 seconds later. Furthermore, the numbers of hyperperiods are fulfilled throughout the trace ($H_{M_1} = 1000$, $H_{M_2} = 2000$, hyperperiods of 100 ms).

To illustrate the accuracy with which the EnOS hardware executes mode switches, Figure 12 shows a comparison of the thresholds at which an energy-mode switch is due and the thresholds at which the prototype has actually initiated a switch during the experiment. Based on these numbers, we make two important observations: First, for switches to the lower energy modes M_1 and M_2, the difference between the calculated and observed thresholds are small (i.e., between 0.3 %



Mode Switch	Time t	Static $E_{switch}(i)$	Observed $E_{switch}(i)$
$M_1 \rightarrow M_2$	336 s	12,252 mJ	12,218 mJ
$M_2 \rightarrow M_{susp}$	579 s	87 mJ	65 mJ
$M_{susp} \rightarrow M_1$	636 s	17,640 mJ	18,122 mJ
$M_1 \rightarrow M_2$	742 s	12,252 mJ	12,218 mJ

Fig. 12. Energy consumption and energy-mode switches measured on the real EnOS hardware platform; comparison of calculated and actual switching thresholds.

and 3 %). Second, for all switches from a lower to a higher mode, the absolute difference between the calculated threshold and the state of charge at which EnOS actually performed the switch was less than the grace budget for the lower mode (i.e., 81.52 mJ for M_1 and M_2). This is important as it confirms that the EnOS hardware is able to implement the energy budgets defined by the kernel, which include enough energy to complete a started hyperperiod.

8.5 Evaluating Application-Triggered Mode Switches

Previous evaluations demonstrate the accuracy and reliability of the energy-mode switches issued by the operating system. In the following, we present an example trace showing the application overwriting the mode-switch mechanism of EnOS, as described in Section 4.4.5. As shown in Figure 13, EnOS starts execution normally on the lowest energy criticality mode M_1. After 500 hyperperiods, however, the application requests a switch to the next higher energy mode. Thus, execution continues on mode M_2 although the available energy is sufficient to operate on mode M_1. As soon as the application gives control back to the default switching mechanism of EnOS at 150 s, EnOS switches back to mode M_1. After another 500 hyperperiods, the application requests another switch to mode M_1. This time no energy is harvested causing EnOS to switch to the next higher energy mode M_2 at 396 s, confirming that necessary changes to higher energy criticality modes are still working, although the application has not yet given control back to EnOS.

9 RELATED WORK

To our knowledge, EnOS is the first operating-system kernel for energy-neutral real-time systems with both mixed criticalities and mixed (time and energy) constraints. However, we are not the first to investigate mixed-criticality scheduling in combination with energy constraints. Völp et al. [32] demonstrated that energy can surpass timeliness in mixed-criticality systems. To improve schedulability in such systems, they presented an energy-aware OCBP algorithm that for each criticality mode considers distinct WCET values and WCEC estimates. Extending on their work, we introduce energy criticality modes that are independent of time criticality levels. As a result, EnOS is able to not only utilize optimistic WCEC estimates for tasks with low criticality levels, but for entire task sets executed while sufficient energy is available.

Power management in embedded energy-harvesting wireless sensor networks is a broad area of ongoing research [7, 8, 19], which comprises static (i.e., capacity planning) or dynamic power

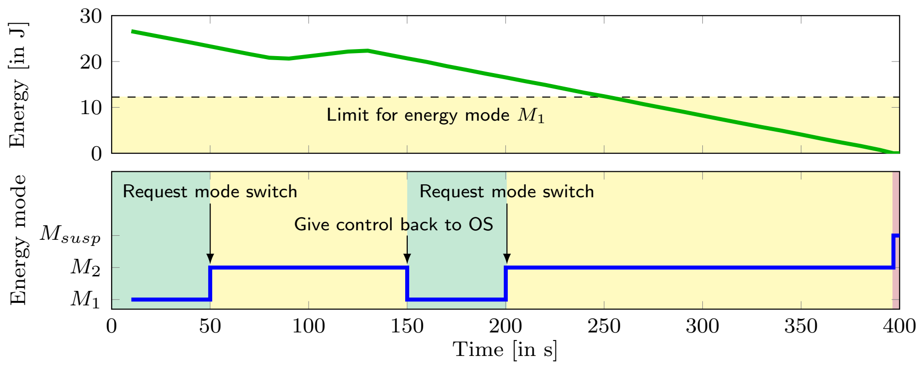


Fig. 13. Energy-mode switches by the application, showing the possibility to operate on a mode, although sufficient energy for a lower mode is available.

management (i.e., reducing the duty cycle of the sensing node). Buchli et al. [8] presented a power-management approach to guarantee uninterrupted system operation on a scale of multiple years. To achieve the goal of energy-neutral operation, their approach exploits varying circumstances (i.e., intensity of sunlight) and dynamic load adaption (i.e., reduction of duty cycles). Within a duty cycle, the system stays active for computations or transmitting data. In comparison to their work, the EnOS operating-system kernel also considers real-time requirements and intermittent operation, and applies suspension and resumption techniques to enable an energy-neutral system to survive blackout periods.

Reducing energy consumption under the objective of guaranteeing timeliness is a well-explored topic through the utilization of dynamic voltage and frequency scaling (DVFS) [1, 5, 39]. However, such energy-aware approaches are not directly applicable for energy-neutral real-time systems as energy consumption does not necessarily correlate with timing behavior. For example, switching a general-purpose input-output pin might be performed within only a few processor cycles; however, if such an operation activates an external sensor, it leads to a significantly higher power consumption. Consequently, WCET and WCEC estimates must be independently determined and considered for scheduling, which is possible in EnOS's scheduling approach.

Ongoing research on replacing volatile with non-volatile memory offers new possibilities for energy-neutral platforms facing intermittent operation [21, 40]. If the application has the possibility to only access non-volatile memory, after a blackout it can resume execution at the point at which it ran out of energy. However, in contrast to the EnOS kernel, such approaches have the shortcoming that no guarantees on timeliness are provided.

10 CONCLUSION

The rise of energy-neutral systems due to steadily improving energy-harvesting hardware comes with a variety of possibilities to make systems self-sufficient. A major challenge is to make these systems as reliable as possible under the hard to predict environmental circumstances. Energy-neutral real-time systems have unique challenging characteristics that distinguish them from other energy-neutral systems: During periods when sufficient energy is available, energy-neutral systems behave like traditional real-time systems in which timeliness is the most important requirement. However, if the amount of energy harvested drops below the minimum level required to keep system operations alive, energy consumption suddenly becomes the highest priority. That is, time and energy constraints are equally important, but not at the same time.

In this paper, we presented EnOS, an operating-system kernel that has been specifically designed to address the unique characteristics of energy-neutral systems. EnOS offers the possibility to specify different energy criticality modes, each of which executes a dedicated set of tasks with mixed time criticalities. By switching between energy modes, the system is able to dynamically adjust the rules by which tasks are scheduled, depending on external conditions. In the most extreme case, EnOS stops enforcing timeliness altogether in favor of ensuring that its state is safely stored to persistent memory before the system runs out of energy.

REFERENCES

[1] M. Bambagini, M. Bertogna, and G. Buttazzo. 2014. On the Effectiveness of Energy-Aware Real-Time Scheduling Algorithms on Single-Core Platforms. In *Proceedings of the 20th International Conference on Emerging Technology and Factory Automation.* 1–8.

[2] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. 2012. Scheduling Real-Time Mixed-Criticality Jobs. *IEEE Trans. Comput.* 61, 8 (2012), 1140–1152.

[3] I. Bate, A. Burns, and R. I. Davis. 2015. A Bailout Protocol for Mixed Criticality Systems. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems.* 259–268.

[4] E. Bini and G. C. Buttazzo. 2005. Measuring the Performance of Schedulability Tests. *Real-Time Systems* 30 (2005), 129–154.

[5] E. Bini, G. Buttazzo, and G. Lipari. 2005. Speed Modulation in Energy-Aware Real-Time Systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems.* 3–10.

[6] B. Buchli, D. Aschwanden, and J. Beutel. 2013. Battery State-of-Charge Approximation for Energy Harvesting Embedded Systems. In *Proceedings of the 10th Europ. Conference on Wireless Sensor Networks.* 179–196.

[7] B. Buchli, P. Kumar, and L. Thiele. 2015. Optimal Power Management With Guaranteed Minimum Energy Utilization For Solar Energy Harvesting Systems. In *Proceedings of the 11th International Conference on Distributed Computing in Sensor Systems.* 147–158.

[8] B. Buchli, F. Sutton, J. Beutel, and L. Thiele. 2014. Dynamic Power Management for Long-term Energy Neutral Operation of Solar Energy Harvesting Systems. In *Proceedings of the 12th Conference on Embedded Network Sensor Systems.* 31–45.

[9] A. Burns. 2014. System Mode Changes – General and Criticality-Based. In *Proceedings of the 2nd Workshop on Mixed Criticality Systems.* 3–8.

[10] A. Burns and R. I. Davis. 2015. *Mixed Criticality Systems – A Review.* 6th Edition. Department of Computer Science, University of York, York, UK.

[11] J. Constine. 2014. Facebook Will Deliver Internet Via Drones With "Connectivity Lab" Project Powered By Acqhires From Ascenta. (2014). http://techcrunch.com/2014/03/27/facebook-drones/

[12] Eaton. 2016. *HV Supercapacitors – Cylindrical cells.*

[13] Fujitsu Semiconductor. 2013. *FRAM MB85RC256V.*

[14] Google Inc. 2016. Project Loon. (2016). https://www.google.com/loon/

[15] C. Gu, N. Guan, Q. Deng, and W. Yi. 2013. Improving OCBP-Based Scheduling for Mixed-Criticality Sporadic Task Systems. In *Proceedings of the 19th International Conference on Embedded and Real-Time Computing Systems and Applications.* 247–256.

[16] T. Hönig, H. Janker, C. Eibel, O. Mihelic, R. Kapitza, and W. Schröder-Preikschat. 2014. Proactive Energy-Aware Programming with PEEK. In *Proceedings of the Conference on Timely Results in Operating Systems.* 1–14.

[17] Intersil. 2016. *ISL85412 – Synchronous Buck Regulator.*

[18] R. Jayaseelan, T. Mitra, and X. Li. 2006. Estimating the Worst-Case Energy Consumption of Embedded Software. In *Proceedings of the 12th Real-Time and Embedded Technology and Applications Symposium.* 81–90.

[19] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava. 2007. Power Management in Energy Harvesting Sensor Networks. *ACM Transactions on Embedded Computing Systems* 6, 4 (2007).

[20] J. P. Lehoczky, L. Sha, and J. K. Strosnider. 1987. Enhanced Aperiodic Responsiveness in a Hard-Real-Time Environment. In *Proceedings of the 8th Real-Time Systems Symposium.* 261–270.

[21] B. Lucia and B. Ransford. 2015. A Simpler, Safer Programming and Execution Model for Intermittent Systems. In *Proceedings of the 36th Conference on Programming Language Design and Implementation.* 575–585.

[22] M. H. Mahmud, R. Saha, and S. Islam. 2013. Smart Walking Stick – An Electronic Approach to Assist Visually Disabled Persons. *International Journal of Scientific & Engineering Research* 4, 10 (2013).

[23] J. Pallister, S. Kerrison, J. Morse, and K. Eder. 2015. Data Dependent Energy Modelling: A Worst Case Perspective. *Computing Research Repository, arXiv* (2015).

[24] P. Puschner and A. Schedl. 1997. Computing Maximum Task Execution Times: A Graph-Based Approach. *Real-Time Systems* 13 (1997), 67–91.

[25] J. Real and A. Crespo. 2004. Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal. *Real-Time Systems* 26, 2 (2004), 161–197.

[26] C. Renner and V. Turau. 2012. State-of-Charge Assessment for Supercap-Powered Sensor Nodes: Keep it Simple Stupid!. In *Proceedings of the 9th International Conference on Networked Sensing.*

[27] K. Ryokai, P. Su, E. Kim, and B. Rollins. 2014. EnergyBugs: Energy Harvesting Wearables for Children. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* 1039–1048.

[28] M. Sand, S. Potyra, and V. Sieh. 2009. Deterministic High-Speed Simulation of Complex Systems Including Fault-Injection. In *Proceedings of the 39th Conference on Dependable Systems and Networks.* 211–216.

[29] V. Sieh, R. Burlacu, T. Hönig, H. Janker, P. Raffeck, P. Wägemann, and W. Schröder-Preikschat. 2017. An End-To-End Toolchain: From Automated Cost Modeling to Static WCET and WCEC Analysis. In *Proceedings of the 20th International Symposium on Real-Time Computing.* 1–10.

[30] P. Ulbrich, R. Kapitza, C. Harkort, R. Schmid, and W. Schröder-Preikschat. 2011. I4Copter: An Adaptable and Modular Quadrotor Platform. In *Proceedings of the 26th ACM Symposium on Applied Computing (SAC '11).* 380–396.

[31] S. Vestal. 2007. Preemptive Scheduling of Multi-Criticality Systems with Varying Degrees of Execution Time Assurance. In *Proceedings of the 28th Real-Time Systems Symposium.* 239–243.

[32] M. Völp, M. Hähnel, and A. Lackorzynski. 2014. Has Energy Surpassed Timeliness? – Scheduling Energy-Constrained Mixed-Criticality Systems. In *Proceedings of the 20th Real-Time and Embedded Technology and Applications Symposium.* 275–284.

[33] P. Wägemann, T. Distler, T. Hönig, H. Janker, R. Kapitza, and W. Schröder-Preikschat. 2015. Worst-Case Energy Consumption Analysis for Energy-Constrained Embedded Systems. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems.* 105–114.

[34] P. Wägemann, T. Distler, H. Janker, P. Raffeck, and V. Sieh. 2016. A Kernel for Energy-Neutral Real-Time Systems with Mixed Criticalities. In *Proceedings of the 22nd Real-Time and Embedded Technology and Applications Symposium.* 25–36.

[35] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner. 2008. Measurement-Based Timing Analysis. *Leveraging Applications of Formal Methods, Verification and Validation* 17 (2008), 430–444.

[36] R. Wilhelm et al. 2008. The Worst-Case Execution-Time Problem – Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems* 7, 3 (2008), 1–53.

[37] H. Zeng, X. Fan, C. Ellis, A. Lebeck, and A. Vahdat. 2002. ECOSystem: Managing Energy as a First Class Operating System Resource. In *Proceedings of the 10th Conference on Architectural Support for Programming Languages and Operating Systems.* 123–132.

[38] T. Zhu, Z. Zhong, Y. Gu, T. He, and Z.-L. Zhang. 2009. Leakage-Aware Energy Synchronization for Wireless Sensor Networks. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services.* 319–332.

[39] Y. Zhu and F. Mueller. 2004. Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling. In *Proceedings of the 10th Real-Time and Embedded Technology and Applications Symposium.* 84–93.

[40] M. Zwerg, A. Baumann, R. Kuhn, M. Arnold, R. Nerlich, M. Herzog, R. Ledwa, C. Sichert, V. Rzehak, P. Thanigai, and B. O. Eversmann. 2011. An 82μA/MHz Microcontroller with Embedded FeRAM for Energy-Harvesting Applications. In *Proceedings of the International Solid-State Circuits Conference.* 334–336.

Received June 2016; revised January 2017; accepted April 2017