To appear in: Proceedings of the 22nd Real-Time Embedded Technology and Applications Symposium (RTAS 2016)
Vienna, Austria, 11–14 April 2016

A Kernel for Energy-Neutral Real-Time Systems with Mixed Criticalities

Peter Wägemann, Tobias Distler, Heiko Janker, Phillip Raffeck, and Volkmar Sieh
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

***Abstract*—Energy-neutral real-time systems harvest the entire energy they use from their environment, making it essential to treat energy as an equally important resource as time. As a result, such systems 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 paper, 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 during phases when one is considered less critical than the other. When switching the energy criticality mode, the system also changes the set of tasks to be executed 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.

I. 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 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 Google's Loon balloon [1] or Facebook's solar drone Ascenta [2], 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 [3], 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 [4], [5], 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 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 [6]. 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 obtaining 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 there is enough energy available to continue system execution for at least a predefined 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 paper, we address the challenges identified above with ENOS, an operating system kernel for energy-neutral real-time systems. In contrast to existing scheduling models [3], [4], 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., soft and hard) 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. Furthermore, our evaluation reveals that the decoupling of time and energy constraints and the use of distinct energy criticality modes besides time criticality levels 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 interrupts*. 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 making decisions on 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 the tasks involved [7], [8]. 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 paper are:

1) We present a scheduling model that considers mixed criticalities as well as time and energy constraints.
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 paper is structured as follows: Section II gives an overview of ENOS and its system model. Section III details mixed-criticality scheduling in ENOS. Section IV presents energy interrupts and Section V the suspend/resume mechanism. Our current ENOS implementation is outlined in Section VI and evaluated in Section VII. Finally, Section VIII discusses related work and Section IX concludes.

II. The ENOS Kernel

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

A. System Model

ENOS addresses energy-neutral systems that are able to draw energy from at least one, possibly multiple sources such as the sun or wind. If external conditions are good, a system harvests enough energy to provide all of its services (e.g., the collection and aggregation of sensor data), but this scenario does not necessarily always apply. Instead, for example, a change in weather conditions or the setting sun may lead to energy becoming scarce. 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., the failure of a solar panel), the amount of newly harvested energy can drop almost instantly from its maximum to zero. An energy-neutral system must be prepared to deal with such situations and suspend system operations (in case of a stationary system) or even abort its mission entirely (e.g., by initiating an emergency landing) before running out of energy.

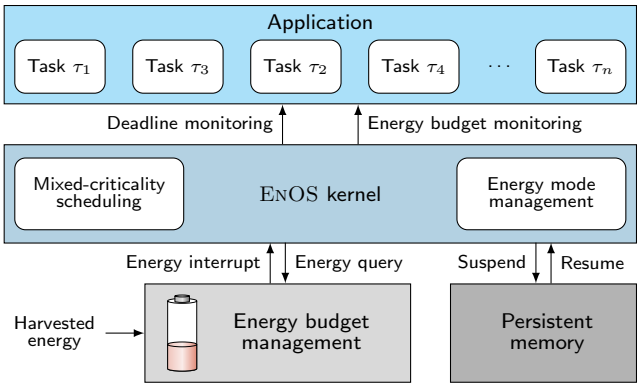

Figure 1: Overview of the ENOS architecture.

In order to be able to react to emergencies, an energy-neutral system requires a battery in which unused harvested energy is stored. The capacity of the battery is usually large enough to keep operations alive for hundreds or a few thousands of system cycles, but far too small to store energy for the entire mission. As a result, the initial state of the battery at the start of the mission is negligible.

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 [9]. In consequence, the system needs to provide a persistent-memory module that enables ENOS to store the state if energy is low.

B. Architecture

Figure 1 shows an overview of the ENOS architecture 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 III, 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 IV). 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 V).

III. Scheduling with Mixed Criticalities and Mixed Constraints

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

A. 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 [10]. 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 [11] 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 [7], [12], are less costly than pessimistic analyses but may under-approximate the WCET and WCEC. Therefore, in this case 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 [4] 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, 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, because 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 [13], this term can be seen as umbrella term for various older scheduling approaches. For instance, the work of Lehoczky et al. [14] considers dual-criticality systems with soft and hard tasks. 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.

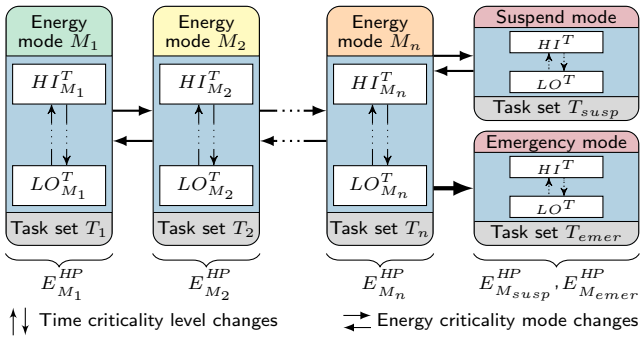

Figure 2: The ENOS scheduling model comprises different energy (criticality) modes, in each of which a dedicated task set is executed. Within an energy mode, the system time criticality level can vary between low (LO^T) and high (HI^T).

Völp et al. [3] 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.

B. 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 2, 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 drone 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 drone 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 drone 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 drone 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 V).

At low and medium energy criticality modes, in which time constraints predominate, ENOS applies the mixed-criticality approach proposed by Vestal (see Section III-A) to guarantee timeliness. Nevertheless, even in these modes energy constraints are of importance: At all times, ENOS monitors the amount of energy currently available in order to ensure that there is sufficient energy left for the system to switch to higher energy criticality modes and eventually reach the suspend or emergency mode in case it becomes necessary. The decisions of 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 III-D for details). While for low energy criticality modes optimistic analysis techniques are applied to determine energy-consumption estimates, more conservative techniques are used for higher energy criticality 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.

C. Mode-Specific Task Sets

A task set T in ENOS consists of implicit deadline-constrained, periodic, 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 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 III-A) 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.

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 criticality modes the algorithm with the lowest energy consumption is selected.

D. 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 criticality 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 in order to allow the system to degrade gracefully in a safe way.

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:

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

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. 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 III-B). 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 [7]. For all other energy modes, ENOS relies 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 and the maximum physical power consumption of the hardware platform (see Section VI-C). 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 exceeded. In order to learn about thresholds being surpassed, ENOS sets up an energy interrupt (see Section IV). If the

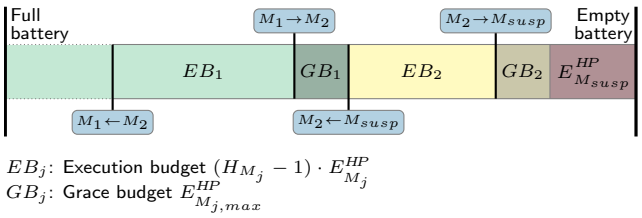

Figure 3: Mode-switch thresholds for an example system with three different energy criticality modes M_1, M_2, and M_{susp}.

energy interrupt fires in the middle of a hyperperiod, ENOS continues to execute the task set of the current energy criticality mode until the end of the hyperperiod is reached 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.

2) Extended Mechanism: 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 3, 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 [7]. 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 [3].

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

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

Figure 3 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.

3) Discussion: Energy-mode switches in ENOS only occur at hyperperiod boundaries, even if an energy interrupt triggers during the execution of a hyperperiod (see Section III-D1). 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 hyperperiods still allows the system to react to changes in environmental conditions quickly, compared to the minutes or hours ENOS usually remains in the same energy mode. Second, reconfiguring the system during the execution of a hyperperiod 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 III-C). 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 hyperperiod 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 III-D1, 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.

IV. 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.

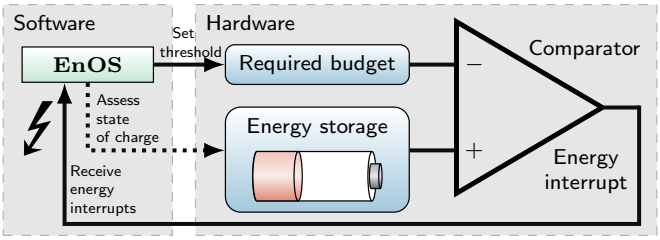

Figure 4: ENOS uses mechanisms for energy interrupts to enforce energy budgets and to assess the current energy budget.

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.

A. Approach

In order to be notified when the state of charge of the battery reaches a certain threshold, ENOS sets up an energy interrupt and then immediately continues to execute the application. As illustrated in Figure 4, 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 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 component.

B. 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 [15]. 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 for 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. [16] 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 VII-D). Although already practical, our goal is to further improve precision as part of future work.

A possible alternative to the comparator-based approach currently employed in ENOS is to benefit from recent improvements in coulomb-counting circuits [17]. Such circuits provide an interrupt once a predefined amount of energy has been consumed, which makes them a perfect match to ENOS's interfaces for enforcing energy budgets. However, coulomb-counting circuits are not able to assess the total state of charge, but the drawn charge. Furthermore, adding such a component to the hardware prototype, which is described in Section VI-A, would require additional circuitry.

V. Surviving Blackouts

For many energy-neutral systems it is not possible to guarantee a continuous flow of energy, for example, because they harvest energy via solar panels. For such systems, it is essential to safely suspend operations once a blackout occurs and to resume them after energy becomes available again. Below, we present the suspend/resume mechanism of ENOS and discuss how our approach facilitates application development.

A. Suspension & Resumption

As discussed in Section III-B, 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 currently relies on ferroelectric RAM (FRAM), which provides acceptable access latencies, low static power consumption, and long durability [9].

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 criticality 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 (see Section III-D2). When the battery's state of charge exceeds the threshold, the hardware triggers an interrupt, as a result of which ENOS resumes execution.

```
1  /* Global variables */
2  int index, value;
4  energy_agnostic_store(int sensor_value) {
5    /* Begin transaction protecting the variables index and value. */
6    begin(index, value);
8    /* Increment index and store sensor value. */
9    index++;
10   value = sensor_value;
12   /* Commit transaction and make updates persistent. */
13   commit(index, value);
14 }
16 enos_store(int sensor_value) {
17   index++;
18   value = sensor_value;
19 }
```

Figure 5: Example functions consistently storing a collected sensor value assigned to a unique index in energy-agnostic systems (top) and ENOS (bottom).

B. 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 [7] 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 [18]. 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 obtain relying on a precise energy-measurement device that is further described in [19]. The benchmark generator considers different sizes of both read and write accesses on 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.

C. Impact on Applications

The fact that ENOS is able to guarantee that at all times there is enough energy available to fully execute the suspend mode not only increases the resilience of energy-neutral systems, it also facilitates their design and implementation. In traditional systems, application tasks running on top of an energy-agnostic operating system themselves have to take care of ensuring that their state remains consistent, for example, by relying on transactions. In contrast, application tasks executed by ENOS are freed of the responsibility to worry about consistency, because the system guarantees that they run to completion once they have been started and that the state of the application is safely stored in case of a blackout.

Figure 5 illustrates the differences in approaches using an example function that stores the latest sensor value collected accompanied by a unique index. In this example, we assume it is crucial for the application that the index is only increased if the value stored actually reflects the latest sensor reading. As shown by the `energy_agnostic_store` function, a typical approach enabling an application to fulfill such a requirement is the use of transactions, with which in this case it is possible to guarantee that the values of the two variables `index` and `value` are kept consistent: If the task runs long enough to successfully commit the transaction (line 13), both updates are stored; otherwise, the changes are discarded. Without the use of a transaction, consistency could not be ensured, since the system might run out of energy after having executed the increment of `index` (line 9), but before having written the new sensor data into `value` (line 10).

As shown by example of the `enos_store` function in Figure 5, ENOS does not require the application to use transactions due to guaranteeing that the system will not run out of energy while executing an application task. Besides providing the benefit of minimizing development and implementation costs, this system property also allows saving energy, because state variables only have to be written to persistent memory once per system suspension, instead of once per task execution.

VI. 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.

A. 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 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) [20]. 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. Consequently, the overall

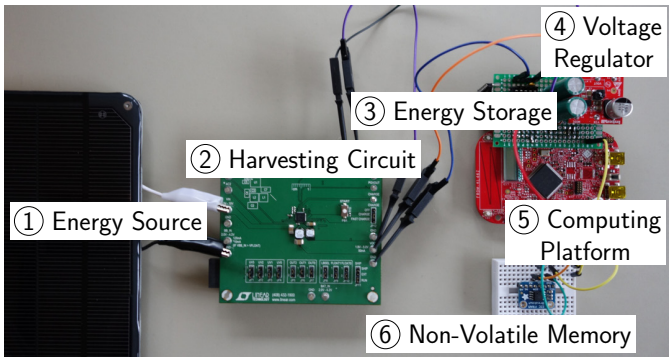

Figure 6: The ENOS operating system requires a tightly coupled interaction between hard- and software components.

energy in the system can be approximated using the equation $E = \frac{1}{2}CU^2$. The voltage is read using an internal 12-bit analog-to-digital converter. Thus, 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 [21]. However, since our prototype platform is equipped with a total capacity of 5 F the effect can be considered as minor (i.e., in the range of 20 µA [22]) during the runtime of around 10 min assuming no energy is harvested from the environment.

B. WCET & WCEC Analyses

ENOS requires knowledge on the energy consumption of the system in different energy criticality modes. For this purpose, we rely on the *0g* WCEC analyzer [7], which we extended for this work. In general, *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 III-D2). On the other hand, *hard energy budgets* are based on analyses with the implicit path enumeration technique (IPET) [11] 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 V-B).

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 (DMA) features, the setup of the comparator and the digital-to-analog converter, accesses to persistent memory (see Section V-B), as well as 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.

Our current prototype distinguishes between two criticality levels for time (low and high) and three criticality levels for energy (i.e., low, medium and high). To introduce additional (medium) criticality 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 in general more accurate results can be obtained by prolonging their execution.

C. Computing Grace Budgets

ENOS guarantees that even in case of an instant blackout, there is enough energy available to execute an entire hyperperiod of each higher energy criticality mode (see Section III-D). This is possible, because when computing the thresholds for energy-mode switches, the system considers the maximum amount of energy that could be consumed within a hyperperiod 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 V-B), 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 [15], [23]. Based on these values and the fixed duration of the hyperperiod 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. For instance, for a hyperperiod of 100 ms the budget is about 82 mJ, which is less than 0.27 % of the capacity of the energy-storage component (i.e., about 30 J, see Section VI-A). To put this number further into perspective: The average current drawn of 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 nearly 15 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 two seconds earlier than actually necessary.

D. Mixed-Criticality Mixed-Constraints Scheduler

The implementation of our mixed-criticality scheduler only comprises about 1,100 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 neces-

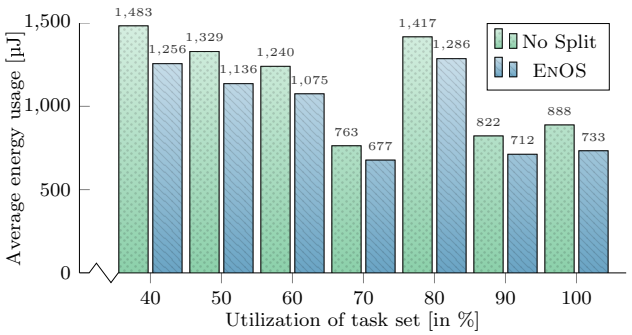

Figure 7: EnOS yields smaller energy budgets for hyperperiods due to treating energy only in distinct situations as critical.

Task	Criticalities	WCETs	WCEC
τ_{1,M_1}	LO^T, LO^E	13 ms	649 µJ
τ_{2,M_1}	HI^T, LO^E	13 ms, 37 ms	649 µJ
τ_{3,M_1}	LO^T, LO^E	15 ms	845 µJ
τ_{1,M_2}	LO^T, MID^E	13 ms	807 µJ
τ_{2,M_2}	HI^T, MID^E	13 ms, 37 ms	807 µJ
τ_{3,M_2}	LO^T, MID^E	15 ms	1183 µJ
$\tau_{1,M_{susp}}$	LO^T, HI^E	13 ms	965 µJ
$\tau_{2,M_{susp}}$	HI^T, HI^E	13 ms, 37 ms	965 µJ
$\tau_{3,M_{susp}}$	LO^T, HI^E	82 ms	3225 µJ

TABLE I: Task parameters considered during the mixed-criticality mixed-constraints scheduling of EnOS.

sary (i.e., for switching tasks in the scheduler). The mode changes between lower and higher energy criticality levels are performed as described in Section III-D. 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 [24], which allows to optimistically return to lower levels as soon as slack time is detected through idling. If necessary, it would also be possible to utilize more sophisticated level-change protocols [25].

VII. 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.

A. Benefits of Criticality-Dependent Energy Budgets

At lower energy criticality modes, EnOS utilizes optimistic energy-consumption estimates to make scheduling decisions (see Section III-D2). 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 proposed by Völp et al. [3]. 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 [26] with utilization values ranging from 40 % to 100 % (in steps of 10 %). Only mixed-criticality schedulable tasks are considered in this evaluation. The WCEC values are approximated using to an average power consumption of 55 mW (i.e., the power consumption in standard run mode without using peripherals, see Section VI-C) 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 [8]. 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 for 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.

B. 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 I). 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. Note that in addition to the calculation of energy budgets described in the EA-OCBP approach [3], we extended the algorithm to also consider the energy consumption of idle times in order to be able to compare calculated execution budgets with actual measurements. 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.

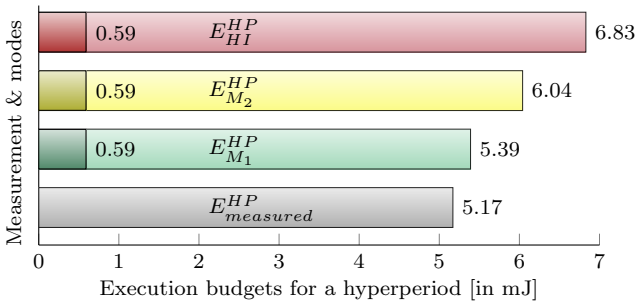

Figure 8: The over-estimation of statically calculated hyperperiod energy budgets depends on the mode's criticality.

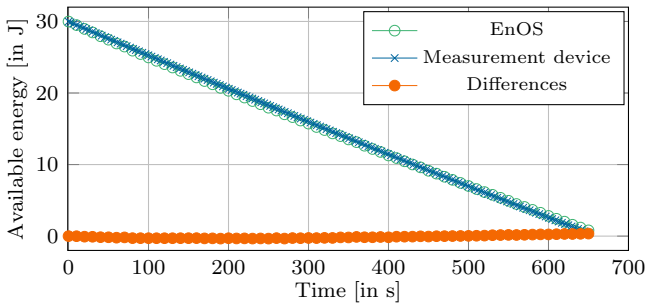

Figure 9: Available energy determined inside the ENOS kernel compared to an external measurement device.

For our measurements on the real target platform, we rely on a dual-channel source meter, 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 8 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 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 $E^{HP}_{HI} = 6.83$ 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.

C. Evaluating State-of-Charge Assessment

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 9, ENOS precisely assesses the amount of energy available over the entire range of charge levels.

D. Evaluating Energy Interrupts

To evaluate the granularity of detecting energy-budget violations using the energy-interrupt mechanism described in Section IV, 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 %.

E. 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 VI 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 VII-B. 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 I).

Simulation-Based Measurements: In order 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 [27] 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 10 shows the result of this experiment. Initially, the battery is fully charged, which means that ENOS is able to run in the lowest energy criticality 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 criticality mode ($t = 86$ s) in which it remains for

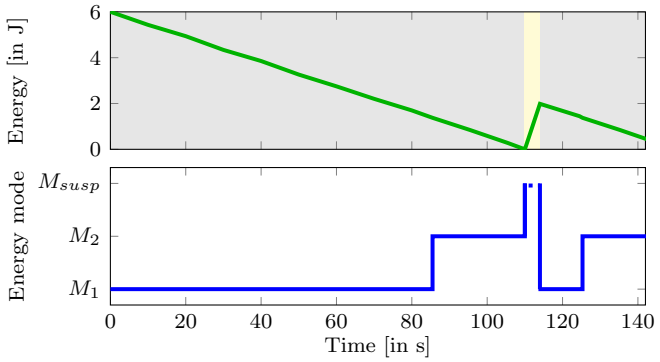

Figure 10: Energy-mode switches in the small system: At $t = 110$ s the system executes the suspend mode and sleeps until enough energy has been harvested to resume ($t = 114$ s).

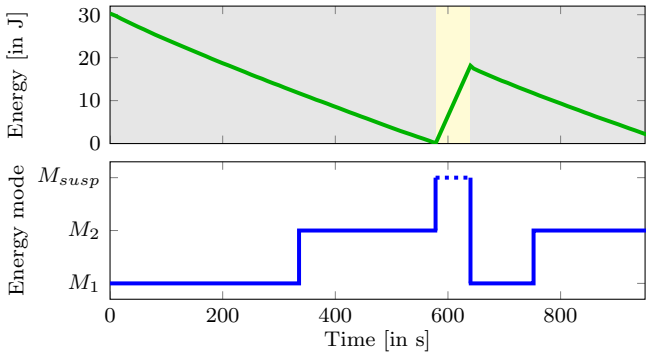

Figure 11: Energy consumption and energy-mode switches over time measured on the real ENOS hardware platform.

Mode Switch	Time	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

TABLE II: Comparison of calculated and actual mode-switch thresholds on the real ENOS hardware platform.

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 11 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, Table II 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 mode switch during the experiment. Based on these numbers, we make two important observations: First, for switches to the lower energy criticality modes M_1 and M_2, the difference between the calculated and observed thresholds are small (i.e., between 0.3 % 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 because 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.

VIII. 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. [3] 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 the system has sufficient energy available.

Power management in embedded energy-harvesting wireless sensor networks is a broad area of ongoing research [28], [29], [30], which comprises static (i.e., capacity planning) or dynamic power management (i.e., reducing the duty cycle of the sensing node). Buchli et al. [29] 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) [31], [32], [33]. 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 [9], [34]. 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.

IX. Conclusion

Energy-neutral real-time systems have unique characteristics that distinguish them from other embedded 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.

Acknowledgment

We thank Martin Hierold and Klaus Hochradel for their help. This work is supported by the German Research Foundation (DFG), in part by Research Grant no. SCHR 603/13-1 (Power-Aware Critical Sections), Research Unit FOR 1508 under grant no. SCHR 603/11-2 (BATS), and the Bavarian Ministry of State for Economics under grant no. 0704/883 25 (EU EFRE funds).

References

[1] Google Inc. Project Loon. [Online]. Available: https://www.google.com/loon/

[2] J. Constine. (2014) Facebook will deliver Internet via drones with "Connectivity Lab" project powered by acqhires from Ascenta. [Online]. Available: http://techcrunch.com/2014/03/27/facebook-drones/

[3] M. Völp, M. Hähnel, and A. Lackorzynski, "Has energy surpassed timeliness? – Scheduling energy-constrained mixed-criticality systems," in *Proc. of the 20th Real-Time and Embedded Technology and Applications Symp.*, 2014, pp. 275–284.

[4] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. of the 28th Intl. Real-Time Systems Symp.*, 2007, pp. 239–243.

[5] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," *IEEE Trans. on Computers*, vol. 61, no. 8, pp. 1140–1152, 2012.

[6] H. Zeng, X. Fan, C. Ellis, A. Lebeck, and A. Vahdat, "ECOSystem: Managing energy as a first class operating system resource," in *Proc. of the 10th Conf. on Architectural Support for Programming Languages and Operating Systems*, 2002, pp. 123–132.

[7] P. Wägemann, T. Distler, T. Hönig, H. Janker, R. Kapitza, and W. Schröder-Preikschat, "Worst-case energy consumption analysis for energy-constrained embedded systems," in *Proc. of the 27th Euromicro Conf. on Real-Time Systems*, 2015, pp. 105–114.

[8] R. Jayaseelan, T. Mitra, and X. Li, "Estimating the worst-case energy consumption of embedded software," in *Proc. of the 12th Real-Time and Embedded Technology and Applications Symp.*, 2006, pp. 81–90.

[9] B. Lucia and B. Ransford, "A simpler, safer programming and execution model for intermittent systems," in *Proc. of the 36th Conf. on Programming Language Design and Implementation*, 2015, pp. 575–585.

[10] Wilhelm et al., "The worst-case execution-time problem – Overview of methods and survey of tools," *ACM Trans. on Embedded Computing Systems*, vol. 7, no. 3, pp. 1–53, 2008.

[11] P. Puschner and A. Schedl, "Computing maximum task execution times: A graph-based approach," *Real-Time Systems*, vol. 13, pp. 67–91, 1997.

[12] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner, "Measurement-based timing analysis," *Leveraging Applications of Formal Methods, Verification and Validation*, vol. 17, pp. 430–444, 2008.

[13] A. Burns and R. Davis, "Mixed criticality systems – A review," Department of Computer Science, University of York, York, UK, 6th Edition, 2015.

[14] J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced aperiodic responsiveness in a hard-real-time environment," in *Proc. of the 8th Real-Time Systems Symp.*, 1987, pp. 261–270.

[15] C. Renner and V. Turau, "State-of-charge assessment for supercap-powered sensor nodes: Keep it simple stupid!" in *Proc. of the Intl. Work. on Algorithms and Concepts for Networked Sensing Systems Powered by Energy Harvesters*, 2012.

[16] B. Buchli, D. Aschwanden, and J. Beutel, "Battery state-of-charge approximation for energy harvesting embedded systems," in *Proc. of the 10th Europ. Conf. on Wireless Sensor Networks*, 2013, pp. 179–196.

[17] Linear Technology, *LTC4150 – Coulomb Counter/Battery Gas Gauge*.

[18] J. Pallister, S. Kerrison, J. Morse, and K. Eder, "Data dependent energy modelling: A worst case perspective," *Computing Research Repository, arXiv*, 2015.

[19] T. Hönig, H. Janker, C. Eibel, O. Mihelic, R. Kapitza, and W. Schröder-Preikschat, "Proactive energy-aware programming with PEEK," in *Proc. of the Conf. on Timely Results in Operating Systems*, 2014, pp. 1–14.

[20] Fujitsu Semiconductor, *FRAM MB85RC256V*.

[21] T. Zhu, Z. Zhong, Y. Gu, T. He, and Z.-L. Zhang, "Leakage-aware energy synchronization for wireless sensor networks," in *Proc. of the 7th Intl. Conf. on Mobile Systems, Applications, and Services*, 2009, pp. 319–332.

[22] Eaton, *HV Supercapacitors – Cylindrical cells*.

[23] Intersil, *ISL85412 – Synchronous Buck Regulator*.

[24] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Systems*, vol. 26, no. 2, pp. 161–197, 2004.

[25] I. Bate, A. Burns, and R. Davis, "A bailout protocol for mixed criticality systems," in *Proc. of the 27th Euromicro Conf. on Real-Time Systems*, 2015, pp. 259–268.

[26] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, pp. 129–154, 2005.

[27] M. Sand, S. Potyra, and V. Sieh, "Deterministic high-speed simulation of complex systems including fault-injection," in *Proc. of the 39th Conf. on Dependable Systems and Networks*, 2009, pp. 211–216.

[28] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Trans. on Embedded Computing Systems*, vol. 6, no. 4, 2007.

[29] B. Buchli, F. Sutton, J. Beutel, and L. Thiele, "Dynamic power management for long-term energy neutral operation of solar energy harvesting systems," in *Proc. of the 12th Conf. on Embedded Network Sensor Systems*, 2014, pp. 31–45.

[30] B. Buchli, P. Kumar, and L. Thiele, "Optimal power management with guaranteed minimum energy utilization for solar energy harvesting systems," in *Proc. of the 11th Intl. Conf. on Distributed Computing in Sensor Systems*, 2015, pp. 147–158.

[31] E. Bini, G. Buttazzo, and G. Lipari, "Speed modulation in energy-aware real-time systems," in *Proc. of the 17th Euromicro Conf. on Real-Time Systems*, 2005, pp. 3–10.

[32] M. Bambagini, M. Bertogna, and G. Buttazzo, "On the effectiveness of energy-aware real-time scheduling algorithms on single-core platforms," in *Proc. of the 20th Intl. Conf. on Emerging Technology and Factory Automation*, 2014, pp. 1–8.

[33] Y. Zhu and F. Mueller, "Feedback EDF scheduling exploiting dynamic voltage scaling," in *Proc. of the 10th Real-Time and Embedded Technology and Applications Symp.*, 2004, pp. 84–93.

[34] M. Zwerg, A. Baumann, R. Kuhn, M. Arnold, R. Nerlich, M. Herzog, R. Ledwa, C. Sichert, V. Rzehak, P. Thanigai, and B. O. Eversmann, "An 82μA/MHz microcontroller with embedded FeRAM for energy-harvesting applications," in *Proc. of the Intl. Solid-State Circuits Conf.*, 2011, pp. 334–336.