Real-Time Task Scheduling on Intermittently Powered Batteryless Devices

Mohsen Karimi, *Student Member, IEEE,* Hyunjong Choi, *Student Member, IEEE,* Yidi Wang, *Student Member, IEEE,* Yecheng Xiang, *Student Member, IEEE,* Hyoseung Kim, *Member, IEEE*

***Abstract*—Intermittently-powered devices have gained much interest in recent years. However, scheduling real-time tasks while supporting data consistency, timekeeping, and schedulability guarantees on these devices still remains a challenge. Many sensing tasks need long indivisible sensor reading operations, but most prior work has limited their focus to the forward progress of computation-only tasks. In this paper, we propose a scheduling framework to execute real-time periodic tasks with atomic sensing operations. Our proposed method keeps track of time progress and ensures the periodic execution of sensing tasks while efficiently utilizing intermittent power sources. We provide schedulability analysis to determine if a taskset is schedulable under a given charging condition. As a proof-of-concept, we design a custom programmable RFID tag device, called R'tag, and demonstrate the effectiveness of our framework in a realistic sensing application. Evaluation results show that the proposed method satisfies the real-time task execution requirements on IPDs in terms of task scheduling, timekeeping, and periodic sensing while significantly outperforming prior work.**

***Index Terms*—real-time systems, task scheduling, intermittently powered devices**

I. Introduction

INTERMITTENTLY powered devices (IPDs) recently have gained much interest in a wide variety of fields, from wireless sensor networks to the Internet of Things (IoT), due to their small size, low-cost, and low-maintenance requirements. These devices, which are powered by intermittent power resources such as sunlight, heat, vibration, or Radio Frequency (RF) signals, have diverse applications including smart home, smart agriculture, and health monitoring, to name but a few. IPDs run without batteries, therefore, they do not need battery replacement and regular maintenance. They can last for years, or even decades, without regular care. Furthermore, they can be used in harsh environments where batteries cannot last long, e.g., high temperature environments and inside the body.

Data freshness and timely execution are the key requirements for many sensing tasks, especially those running on IPDs. Data freshness, also known as the age of information, is the time elapsed since the latest data was generated. If a certain event in the environment is sensed long after the actual occurrence, the reaction may become either ineffective or in some cases even dangerous. In most IPDs, long operations get divided into multiple atomic, non-preemptive sub-tasks and execute over multiple power cycles [2]–[5]. Depending on the energy availability, the collected data may become stale, and consequently unusable, in the middle of the process. For example, in blood sugar monitoring for a diabetic to release the proper amount of insulin, any late or improper action would cause a catastrophic damage.

In many sensing applications, data collection from sensor peripherals needs a relatively long execution time, hundreds of milliseconds, which cannot be divided into multiple parts. For example, in gas and environmental sensors such as Bosch BME680 [6], the heater inside the sensor needs to run continuously for a long time before performing a valid measurement. Any device power-off during the execution will require the process to start from the beginning since the hardware state of sensor peripherals cannot be saved and resumed. In IPDs, the energy provided by a small energy storage, typically several μF capacitors, is not sufficient to complete these types of operations. Increasing the energy storage size is not a practical option since it causes long charging times due to the slow rate of voltage increase of large capacitors and may diminish the whole purpose of using IPDs.

In this paper, we propose a real-time scheduling framework for IPDs to address the aforementioned challenges. Our framework provides an energy model that is specifically designed to capture the charging and discharging characteristics of IPDs as well as the periodic execution requirements of sensing tasks. Unlike prior work, our scheduler judiciously controls the charging level of capacitors, which allows storing more energy than what is required to just turn on the MCU. The scheduler computes the required level of voltage to complete the given amount of computation and sensor operations, and makes the device wait in low-power mode until the required level is reached. This enables the successful execution of a long indivisible sensor operation, which was hardly doable by prior work. Furthermore, our work naturally supports timekeeping, which is important to ensure timely and periodic operation and to check the freshness of data obtained from sensors. It is worth noting that our work specifically focuses on IPDs harvesting energy from RFID readers [1]–[3], [7]–[12], but we believe our ideas could be applied to those using different energy sources.

The proposed framework schedules tasks in a non-preemptive manner, which is the fundamental requirement of the state-of-the-art IPD kernels and programming models [2]–[5], [13] to ensure forward progress and memory consistency

This work is supported in part by grants from NSF (1943265), USDA/NIFA (2020-51181-32198), and UC-KIMS CIME (POC2930). An earlier version of this paper was presented at the Embedded Operating Systems Workshop (EWiLi), 2019 [1]. DOI 10.1145/3412821.3412827.

The authors are with the Department of Electrical and Computer Engineering, the University of California Riverside, Riverside, CA 92521, USA (email: mkari007@ucr.edu; hchoi036@ucr.edu; ywang665@ucr.edu; yxian013@ucr.edu ; hyoseung@ucr.edu).

in irregular power losses.[1] This is also important for practical sensing applications as sensor operations are non-suspendable as discussed above. Hence, our work is applicable to existing IPD tools and runtime systems without imposing unrealistic assumptions on task execution.

Based on our scheduler design, we derive the schedulability analysis of periodic tasks and the minimum charging rate required to satisfy all deadlines of a given taskset. To the best of our knowledge, this is the first work that provides sufficient conditions to guarantee real-time task schedulability while considering the energy requirement and non-preemptive nature of tasks on IPDs in practice. We characterize the intermittent availability of a charging source as a periodic energy supply model, and analyze schedulability under fixed-priority scheduling, e.g., Rate Monotonic (RM), and the Earliest-Deadline First (EDF) while considering different energy demands of individual tasks. In addition, we present analyses to answer the following three interesting questions: (i) how much harvested energy is required to schedule a given taskset, (ii) how long can an IPD tolerate and continue to meet task deadlines in the event of occasional energy supply misses, and (iii) how long will it take for the IPD to fully recover from a complete power loss.

To verify the effectiveness of the proposed framework in practical sensing applications, we developed a prototype IPD equipped with an environmental sensor and a chemiresistive sensing capability. We implemented the proposed scheduler on our IPD to guarantee real-time task schedulability. Our implementation also includes the data privatization method [3] to ensure data consistency and forward-progress even in the event of a sudden power loss. Experimental results from both simulation and real hardware demonstrate that our method outperforms the state-of-the-art in both static and dynamic priority scheduling policies, satisfies the timing constraints of real-time tasks on an IPD, and effectively utilizes intermittent energy availability.

II. Related Work

Most prior work on IPDs has focused on the forward progress guarantees of a program between several power failures. In [2], [14], checkpointing methods are proposed to store intermediate results in non-volatile memory and retrieving the results in the next power cycle so that the program can continue to run where it was left at power failure. Mayfly [13] is a graph-based programming language for IPDs, which divides a long-running task into atomic (non-preemptable) subtasks with timing constraints. The program written in Mayfly keeps data consistency by saving the results of each task in non-volatile memory so that they can be used as input for other tasks. Capybara [2] uses multiple capacitor banks to mitigate the atomic execution time problem. [15] presents an energy management unit that allows an IPD to accumulate energy coming from intermittent sources, and proposes a dynamic energy burst scaling technique that keeps track of the load's optimal power point and provides the required burst execution time while minimizing the total energy consumption. However, it requires extra hardware circuitry, which is not appealing to IPDs in a small form factor. Furthermore, similar to the other approaches like [4], [9], the device goes off whenever it exhausts the energy and loses the notion of time so it is unable to schedule sensing tasks with periodic execution requirements. To address this timekeeping issue, [13], [16], [17] presented solutions that enable the device to keep track of time up to several minutes of power failure. However, these approaches either use a real-time-clock (RTC) that is operated by a separate battery [13] or an extra circuit that needs to be attached to the device and lasts up to only several seconds in the event of power failure [16], [17].

InK [3] is an event-based kernel developed for timely execution on IPDs. It relies on an external timer that keeps track of time while the microcontroller (MCU) is in low power mode, and uses interrupts to wake up the device. It uses a similar method to [13] to store the results of each non-preemptive task in non-volatile memory for forward progress and data consistency. Data communication between execution segments is done through data channels, which use a double-buffered method to ensure data consistency between power failures. Efficient task execution on IPDs is also a challenge. [18] proposes a method to maximize the task completion rate, i.e., the number of tasks executable within a fixed interval of time, on an RFID-powered device. It determines when to start tasks in order to minimize the occurrence of power failure in the middle of task execution. However, it does not consider the periodicity and deadline requirements of individual tasks. Zygarde [19] is an imprecise computing-based task execution scheme that enables deep neural networks execution on IPDs with an acceptable inference accuracy. Tygro [20] is a 3D orientation tracking by integrating data from multiple IPDs which are powered by an RFID reader.

Research on the real-time scheduling problem of IPDs with deadline requirements is still in its early stage. Celebi [10] is recent work focusing on this problem. It presents two versions of schedulers, offline and online, among which only the offline scheduler is designed for schedulability in mind. However, Celebi has several limitations for practical use. First, it assumes all tasks are fully preemptive at any point of time. This is incompatible with the state-of-the-art programming models and kernels discussed above and inapplicable to atomic sensing tasks interfacing with sensor peripherals. An unexpected power loss during preemptive execution could lead to data inconsistency or no forward progress, which in turn adversely affects the proposer operation of tasks when the power recovers. Secondly, it assumes that energy harvesting and task execution are mutually exclusive and no discharging occurs when the MCU is in sleep mode, which are not true in many real IPDs [2], [8], [9]. Thirdly, it does not provide an analytical method to test schedulability under its proposed approach, and requires generating a schedule for one hyperperiod (the least common multiple of task periods) to do so. Note that this method is vulnerable to task release jitters and consumes a significant amount of time when task periods are not harmonic. Lee et al. [12] proposed a schedulability analysis method for real-time tasks under fixed-priority and

[1]For example, InK [3] allows preemption only at the boundaries of tasks. Hence, once a task starts execution, it runs non-preemptively until completion.

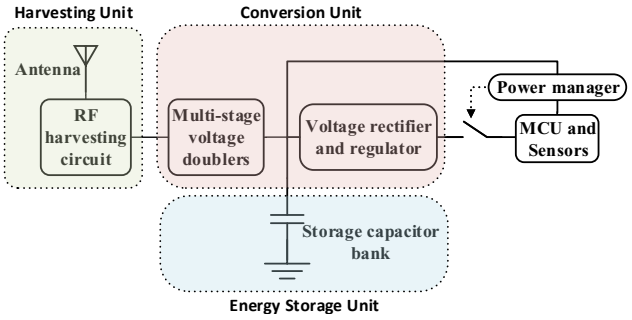


Fig. 1: Block diagram of an RFID energy harvesting device

EDF. However, it has similar limitations to Celebi, i.e., tasks are preemptive at any time with no cost and energy harvesting and task execution are mutually exclusive. In addition, [12] implicitly assumes that the energy source is always available and the charging time and period are determined by the voltage threshold. We address these limitations in this work.

III. Background and System Model

In this section, we describe the hardware and software properties of the system which we use in the rest of the paper.

A. Hardware Characteristics

Most of the energy harvesting devices have the following main units: energy harvester unit, energy storage unit, power management unit, energy conversion unit, and processing unit. The energy harvester unit converts the energy coming from the energy source (e.g. light, wind, vibration, and RF signal) to a type of energy (e.g. voltage and electrical current) that can be accumulated in the energy storage unit. The storage unit, usually consisting of capacitors, can store the energy to be used to power the system. In the energy conversion unit, voltage converters, rectifier, and regulator are used to convert an energy source to the desired voltage for electrical circuits. The power management unit controls when to store energy and when to use the energy to power up the system. Finally, in the processing unit, MCUs and sensors are used to perform the desired operations. In this paper, we focus on IPDs that contain a capacitor as an energy storage unit, following the common model used in most of related work [1]–[3], [7]–[10], [13], [18]–[22] where the size of a capacitor is typically $\leq 200\mu F$ and much smaller than batteries or supercapacitors. It is worth noting that we do not consider devices that do not have a capacitor at all or use different types of energy buffers such as batteries.

Fig. 1 shows the general block diagram of a radio-frequency identification (RFID) energy harvesting device. In this figure, each unit is specifically designed to harvest the RF energy to be used for the MCU and sensors. When the stored energy, i.e., the voltage of the capacitor, reaches a specific level (called *power-on threshold*), the power management unit switches to turn on the MCU and sensors. When the voltage goes down to the minimum voltage level for the MCU and the rest of the circuit (called *power-off threshold*), the system is turned off until the capacitor voltage is recharged to the power-on threshold.

Depending on how energy is consumed, there are multiple types of energy discharging in IPDs. We will analyze the detailed characteristics of each discharging type under our proposed scheduling framework in Section V. Below we give the energy harvesting model used in this paper.

B. Energy Harvesting Model

To find the voltage equations for the circuit, we assume that the energy source has a fixed energy transmission rate while it is charging the device, e.g., an RFID reader is located at a fixed distance from an RFID energy-harvesting device. We consider a parallel resistor R_p to the capacitor, which consists of the equivalent parallel resistor of the storage capacitor and also the rest of the circuit's resistance in parallel with the capacitor. Therefore, the harvesting circuit would become an RC model with a fixed rate energy source. Hence, the voltage of the capacitor with a capacitance of $\mathbb{C}$ can be calculated by:

$$\frac{P}{V} = \mathbb{C}\frac{dV}{dt} + \frac{V}{R_p} \tag{1}$$

By solving this equation, the voltage of the capacitor at time t can be calculated as:

$$V = \sqrt{PR_p - e^{\frac{-2t}{\mathbb{C}R_p}} * (PR_p - V_0^2)} \tag{2}$$

where V_0 is the voltage of capacitor at $t = 0$, and P is the power received from the energy source after going through all the voltage doubler stages. Based on (2), the time to reach from voltage V_0 to V, where $V > V_0$, can be calculated as:

$$t_{charging} = \frac{\mathbb{C}R_p}{2} Ln\left(\frac{PR_p - V_0^2}{PR_p - V^2}\right) \tag{3}$$

The energy source may not be always available to an IPD, e.g., a mobile wireless charger traveling around several IPDs [11]. We characterize the availability of the energy source as a *periodic energy supply model* with two parameters, (C_c, T_c), which means that for a period of T_c, the reader charges the device for at least C_c time units. This can represent both stationary and mobile wireless chargers. In case of a stable, energy source with a constant energy provision rate, it can be modeled as $C_c = T_c$. Note that this periodic energy supply model is much more flexible and easier to use than the one in [10] which requires exact energy-harvesting patterns for every time unit. Also, it is important to mention that the charging time and period, (C_c, T_c), do not impose limitations on the periods of tasks. In other words, T_c can be much larger than task periods or the hyperperiod of the entire taskset. Thus, even with a long interval of the energy source unavailability, the device can maintain the schedulability of the taskset as long as our analysis, which will be given in Section V-C, holds.

C. Task Model

Periodic tasks are considered in this work. Tasks fall into two categories: sensing and computation tasks. Sensing tasks are those that collect data from sensors. All the communications to the sensor as well as sensor data transmissions

are considered as sensing tasks. Computation tasks are CPU-only tasks that do not involve direct interaction with sensor peripherals but process data obtained from the sensors. The dependency between sensing and computation tasks can be easily resolved by using release offsets or deadlines if they have the same period. In more complex scenarios, e.g., sensing and computation tasks have different periods or each sensor is used by multiple computation tasks with different periods, sensing tasks can write data in memory at their own rates and computation tasks can just proceed with the latest data stored in memory, without having to introduce strong dependency or synchronization constraints. This notion of independent (or loosely dependent) operation of sensing and computation tasks has been widely used in many practical applications, such as read-execute-write and publisher-subscriber models [23]–[25].

In this paper, we characterize a task τ_i in a taskset Γ by $\tau_i = (C_i, T_i, D_i)$, where C_i, T_i, and D_i are the worst-case execution time, period, and deadline of the task i, respectively. We consider both fixed-priority scheduling and Earliest-Deadline First (EDF) scheduling in this work. Without loss of generality, we assume that a task with a smaller index has a higher priority under fixed-priority scheduling, i.e., τ_1 is the highest-priority task, and has a smaller relative deadline under EDF, i.e., τ_1 has the smallest relative deadline. All tasks are considered non-preemptive, following the requirements of prior work for the data consistency and forward progress of computational workloads [2]–[5], [13] and respecting the nature of sensing operations.

It should be noted that, if a program has multiple sensing and computation segments, it can be divided into several non-preemptive tasks with the same period. The correct execution order of these divided tasks can be achieved by setting the priorities and deadlines properly. For example, when a task τ_i is divided into multiple subtasks $\tau_i^1, \tau_i^2, ..., \tau_i^m$, where m is the number of subtasks, the deadline of each subtask τ_i^j can be calculated as:

$$D_i^j = D_i - \sum_{k=j+1}^{m} C_i^k \tag{4}$$

where C_i^k is the execution time of a subtask τ_i^k. The execution of a task τ_i is considered complete when all of its subtasks finish their executions. It is worth mentioning that we only consider hard real-time systems where all the tasks (or subtasks) in the taskset should meet their deadline and any missed deadline causes the system to lose its functionality.

IV. Challenges

To elaborate on the challenges of IPDs in sensing applications, we conduct a case study using WISP, a well-known RFID-harvesting device [8]. Based on [7], the power received by WISP can be calculated as:

$$P_r = \frac{G_s G_r \eta}{L_p} \left(\frac{\lambda}{4\pi(d+\beta)} \right)^2 P_t \tag{5}$$

where G_r is the reception antenna gain, G_s is the transmission antenna gain, η is the rectifier efficiency, L_p is the polarization loss, λ is the wavelength of the RF signal, d is the distance from tag to reader, P_t is the transmission power, and β is the adjustment parameter to adjust Friis' free space equation for short distance.

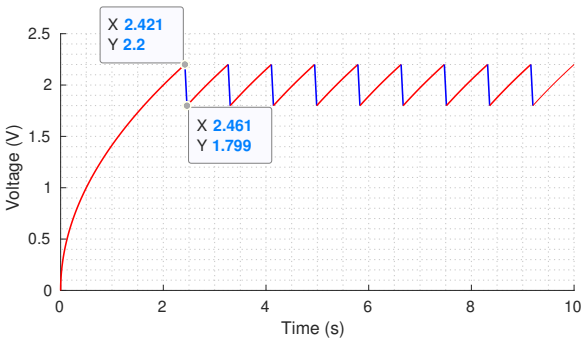


Fig. 2: Charging and discharging cycles of an IPD

The parameters for the above equation are as follows. We use an RFMAX S9028PCL polarized directional antenna which has the transmission gain of $G_s = 8\ dBi$. WISP has a linear dipole antenna; therefore, the reception gain is $G_r = 2\ dBi$. WISP works on 915MHz frequency so the wavelength is about $\lambda = 0.327$. For the other parameters, we use the values of WISP reported in [7]: $\beta = 0.2316$, $\eta = 0.125$, and $L_p = 2$. Since all the parameters except d are constant, we can rewrite (5) to:

$$P_r = \alpha \left(\frac{1}{d+\beta} \right)^2 P_t \tag{6}$$

where α and β are constant. As it can be inferred from (6), the power received by the device is fixed when it is located at a fixed distance to the reader. For the distance of 60 cm and the power transmission of 1 W, the power reception would be about 1 mW.

Let us consider an energy harvesting circuit following the RC model discussed in Section III, with $R_p = 1\ G\Omega$ and $\mathbb{C} = 100\ \mu F$. Based on this harvester and the WISP parameters obtained above, we now analyze the charging and discharging characteristics of an IPD. Fig. 2 shows an example of the voltage level of the device when it is always getting charged by a stationary RFID reader. In this figure, red lines are charging cycles (i.e., the device is turned off) and blue lines are discharging cycles (i.e., the device is on and can execute tasks). The power-on and power-off thresholds, both of which are determined by hardware, are 2.2V and 1.8V, respectively.

We discuss four major challenges observed from this case study. The first is to execute sensing tasks that require long atomic operation without any power disruption. As shown in Fig. 2, the device turns on when the voltage reaches the *power-on threshold*, executes tasks, and turns off when the voltage drops below the *power-off threshold*. The maximum execution time allowed for a task is only about 40 ms. Therefore, any task that needs more than 40 ms of continuous execution can never complete its job. For example, most gas sensors are designed to work at a specific temperature and have an internal micro-heater to maintain that temperature. Since the heater takes time to reach the desired temperature, any intermittent

power loss would lead to a failure in sensing operation and sensor data could not be obtained at all.

The second challenge is to support periodic tasks, e.g., sensing every fixed time interval. However, IPDs start executing tasks immediately when they receive enough energy to turn on the device, e.g., discharging cycles (blue lines) in Fig. 2. The problem becomes more complicated if the RFID reader is not always available, e.g., mobile readers and line-of-sight obstructions. In many sensing applications, capturing samples at a specific time is required to guarantee the data freshness and the validity of processing results.

The third challenge is timekeeping. The device goes off after each discharging cycle and it loses the notion of time. Therefore, any application that needs the notion of time over long periods would not be able to run on IPDs. Using an RTC with a separate battery is not a good solution as doing so runs into the same problems as battery-powered devices. Without having reliable and accurate timekeeping, it is very difficult (if possible at all) to schedule periodic sensing tasks on IPDs and check the freshness of obtained data.

The last challenge lies in the nature of intermittent energy sources. Although we empirically characterize the availability of an energy source using the periodic energy supply model, the actual energy provision may occasionally deviate from expected. For example, in IPDs powered by solar energy, an object may interfere and block the sunlight in one period of charging and cause the system to lose the power and go off. Therefore, a scheduling method on IPDs should tolerate a bounded degree of energy supply misbehavior.

In this work, our goal is to develop a new scheduling framework and analysis to address the aforementioned challenges.

V. Proposed Framework

This section presents our proposed scheduling and analysis framework. We first introduce our runtime scheduler design, and then analyze the energy demand and schedulability of a given taskset under our scheduler. Finally, we derive the minimum charging rate required for schedulability, the maximum time of tolerance to occasional energy supply misses, and the recovery time from a device power loss.

A. Runtime Scheduler Design

Fig. 3 depicts an overview of how our scheduling framework operates at runtime. In our scheduling framework, tasks that are released and ready to run are stored in a ready queue. When no other task is currently running, the runtime scheduler picks a task τ_i that has the highest priority among all tasks in the ready queue. Note that task priority is statically assigned under fixed-priority scheduling and is determined dynamically under EDF. Once the task τ_i to execute is found, the scheduler checks the current energy level of the device, using the `update_charge()` function (derived from (10) in Section V-B). Each task τ_i has a pre-computed charging time Q_i (given in Section V-C) which is to satisfy the energy demand of the task. As the device may already have some accumulated energy, the scheduler computes the waiting time required to gain enough energy (i.e., actual charging time) before the start

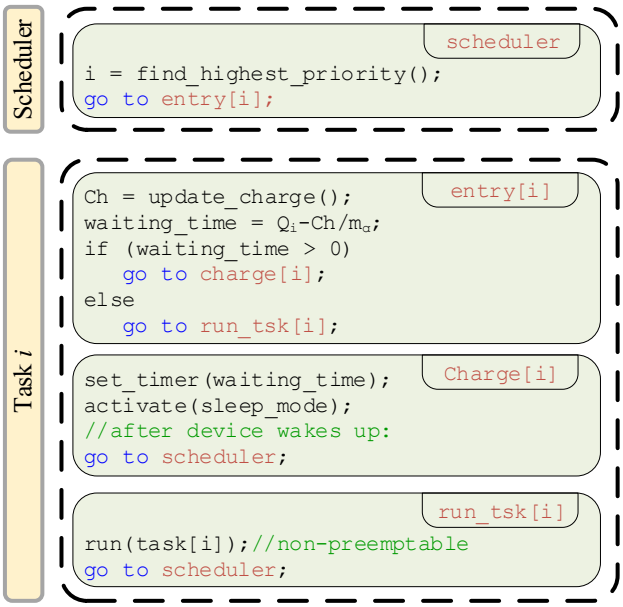


Fig. 3: Scheduler and task execution block diagram

of task execution. If the energy demand of the task is satisfied by the device and no waiting is needed, the task τ_i starts its execution (`run_tsk[i]` in the figure). Otherwise, the scheduler sets a wake-up timer and puts the device in the sleep mode for the required waiting time (`charge[i]`). Whenever a new task is released or the device wakes up from the timer, the scheduler performs the aforementioned procedures again, i.e., finding the highest-priority task, checking the energy demand, computing the waiting time, and putting the device in sleep mode if necessary. In this way, our scheduler ensures that each task runs with enough energy to complete its execution.

There are multiple ways to implement the waiting time and sleep mode. It can be implemented by using the MCU's low-power mode with timer capabilities, e.g., LPM3 in TI MSP430, or adding an external ultra low-power programmable RTC, e.g., 14nA with Ambiq AM0815 RTC [26], which can be powered by harvested energy and wake up the MCU by an interrupt. In the latter case, the MCU can be put into a deeper low power mode, e.g., LPM4 in MSP430, since the MCU's clock sources and timers can be turned off. In either case, the device still draws some power although the amount is much smaller than the active processing power. This will be taken into account by our analysis.

B. Energy Demand and Supply Analysis

We categorize the sources of discharging into three types: *decaying*, *processing*, and *waiting*. First, *decaying* occurs when the device is not receiving any energy from the energy source and the MCU is turned off. In this case, since the circuit is not ideally open circuit, by considering the parallel equivalent resistor in the capacitor, the device gradually loses some energy. Secondly, *processing* occurs during the time when the MCU is turned on and is executing tasks. Lastly, *waiting* happens when the MCU is turned on but is put into sleep (low-power) mode. When the device is being charged, the

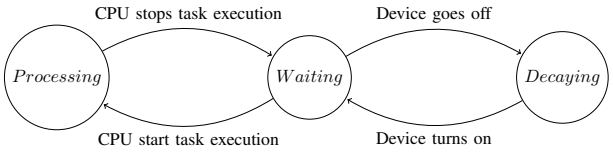


Fig. 4: Three discharging modes transition state machine

difference between charging and discharging would affect the actual voltage of the device. It is worth noting that the *waiting*-induced discharging is not considered in prior work [10], but we explicitly model it since the device cannot indefinitely wait in sleep mode in reality. Fig. 4 shows the transition of different discharging modes.

For simplicity, we assume all the discharging rates to be linear. Due to the fact that the CPU frequency of the MCU remains fixed, the task execution time C_i is independent of the supply voltage. Hence, we use m_Y, m_P, and m_W to denote the discharging rates of *decaying*, *processing*, and *waiting*, respectively. During the *waiting* time, the system consumes some energy but the energy reception from the energy source is assumed to be higher than the energy consumption in *waiting* time. In other words, the capacitor voltage can increase in *waiting* time if the energy source is available. This is true for most commercial MCUs since the power consumption in low-power mode is orders of magnitude lower than that in active mode, e.g., 0.4μA vs. 100μA in MSP430 [27].

The charging rate of the device can also be approximated to be linear by a slope of m_c, where m_c is calculated based on (3) by substituting V_0 with V_{on}, which is the *power-on threshold* of the device, and V with V_{max}, which is the maximum voltage that the capacitor can hold based on its specifications. Therefore, the charging slope is given by:

$$m_c = \frac{V_{max} - V_{on}}{\frac{\mathbb{C}R_p}{2} Ln\left(\frac{PR_p - V_{on}^2}{PR_p - V_{max}^2}\right)} \tag{7}$$

Thus, the worst-case voltage accumulation during each charging period, T_c, can be calculated as:

$$\Delta V = \frac{m_c C_c - m_d\left(T_c - C_c\right)}{T_c} \times \Delta t \tag{8}$$

where C_c and T_c are the charging time and charging period of the energy source, respectively, $m_d = \max\{m_W, m_Y\}$ is the worst-case discharging rate, and m_W and m_Y are the discharging slope of voltage drop during *waiting* and *decaying* time, respectively. We consider an accumulation rate to be

$$m_a = \frac{m_c C_c - m_d\left(T_c - C_c\right)}{T_c} \tag{9}$$

Therefore, the minimum voltage of the capacitor at time t with n periodic tasks can be calculated as:

$$V_{cap}\left(t\right) = m_a t - \sum_{i=1}^{n}\left(\left\lfloor\frac{t}{T_i}\right\rfloor + s_i\right) C_i m_{Pi} + V_0 \tag{10}$$

where C_i, T_i, and m_{Pi} are the worst case execution time, the period, and the *processing* discharging rate of a task τ_i. The runtime variable $s_i \in \{0, 1\}$ indicates if τ_i's last released job for its period at time t has been executed or not. Hence, s_i is set to 0 at the beginning of each period and to 1 when the task finishes execution for that period. In (10), V_0 is given by

$$V_0 = V_{on} - (T_c - C_c) \times m_d \tag{11}$$

where V_{on} is the power-on threshold. Although we consider charging to periodic, the charging can happen at any time during its period. In the worst-case scenario, which we call *back-to-back discharging*, during an interval of $2 \cdot T_c$, charging may happen at the beginning of the first T_c period while the next charging may happen at the end of the second T_c period. The $(T_c - C_c) \times m_d$ term in (11) is to consider the effect of this back-to-back discharging when calculating the voltage of the device from (10). Equations (10) and (11) are used in our runtime scheduler shown in Fig. 3 to estimate the current voltage of the capacitor (i.e., device energy level) at time t.

C. Schedulability Analysis

We first define the necessary condition for a taskset Γ containing n tasks when system parameters are known.

Lemma 1: The taskset is not schedulable if the following condition is not met (necessary condition):

$$m_a \geq \sum_{i=1}^{n} \frac{C_i}{T_i} \times m_{Pi} \tag{12}$$

Proof: Consider the hyperperiod T_h of the taskset, which is the least common multiple of all task periods. The energy reception from the energy source in one hyperperiod can be calculated as $m_a \times T_h$. On the other hand, the energy consumption of the taskset during one hyperperiod can be calculated as $\sum_{i=1}^{n} \frac{T_h}{T_i} C_i m_{Pi}$. Due to the fact that the energy reception during one hyperperiod should be always bigger or equal to the energy consumption of the taskset, the equation can be easily obtained. ■

In order to capture the energy demand of each task τ_i, we define the *charging time*, Q_i, as follows:

$$Q_i = \frac{(m_{Pi} - m_a) \times C_i}{m_a} \tag{13}$$

Since charging can happen while the task is running, we consider $m_{Pi} - m_a$ as the energy consumption rate during task execution. This charging time Q_i is the maximum time the task has to wait before execution. At runtime, the actual waiting time can be shorter than Q_i as the system may already have a non-zero amount of accumulated energy, as discussed in Section V-A. It should be noted that Q_i can be negative, which means the discharging rate of the task can be lower than the charging when $m_a > m_{Pi}$. For example, if the RFID charger is very close to an RF-powered device, the device can have a positive net energy gain even when it is executing some tasks, but at a slower rate than when the device is in sleep mode.

For schedulability analysis, the charging time of a task τ_i can be considered as a *preemptable* execution segment of Q_i that precedes the non-preemptable execution segment of C_i. The reason the charging segment is preemptable is that it can be interrupted at any time by higher-priority tasks without

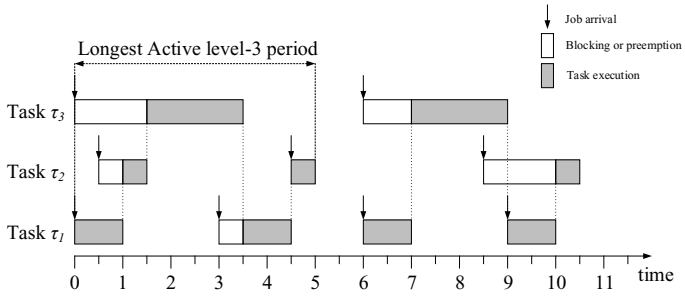


Fig. 5: Active level-i example with 3 tasks.

causing data consistency or forward progress issues. Based on this modeling of charging and execution segments, we derive schedulability analysis in our framework with both fixed-priority and EDF policies.

1) Fixed-Priority Scheduling: For a system under non-preemptive fixed-priority scheduling with no energy restriction, it is known that the schedulability of a task τ_i can be determined by using the notion of the level-i active period [28], which is a time interval that contains only the active jobs of a task τ_i and its higher priority tasks. Fig. 5 shows an example of task execution for 3 tasks of $\tau_1 = (1, 3, 3)$, $\tau_2 = (0.5, 4, 4)$, and $\tau_3 = (2, 6, 6)$ where tasks are ordered based on priority, i.e., τ_1 has the highest priority. As shown in the figure, the active level-3 of the example taskset can be measured as 5 seconds.

Theorem 1 (from [28]): The longest level-i active period of a non-preemptive task τ_i can be computed using the following recurrent relation:

$$L_i^{k+1} = B_i + \sum_{h:\pi_h \geq \pi_i} \left\lceil \frac{L_i^k}{T_h} \right\rceil C_h \tag{14}$$

where B_i is the blocking time due to lower priority tasks and can be calculated as

$$B_i = \max_{l:\pi_l < \pi_i} \{C_l\} \tag{15}$$

To determine the worst-case response time of the task τ_i, it is sufficient to consider the response times of jobs during the longest level-i active period. The first term in (14) captures the delay that occurs when the execution of a lower-priority task prevents a higher-priority task from execution due to the nature of non-preemptive scheduling. For example, at time 3 in Fig. 5, τ_1 is blocked by τ_3 for 0.5 time units. The second term of the equation captures the delay caused by the execution of higher-priority tasks. For example, at time 1, τ_2 executes ahead of τ_3 due to its higher priority. It is worth mentioning that, due to non-preemptive scheduling, scheduling decisions are made only when the previous task completes execution. More details of the theorem can be found in [28].

We extend the conventional non-preemptive fixed-priority schedulability test [28] in order to take into account the energy requirements of intermittently-powered tasks. However, the charging time Q_i cannot be directly added as an additional active workload to the level-i active period because Q_i may be negative for certain tasks. Having a negative execution time, i.e., $Q_i < 0$, violates the assumptions of the schedulability analysis and leads to a failure in finding critical instants in the level-i active period. Hence, to address this problem, we use $Q_i^+ = \max(Q_i, 0)$ in our analysis. This is safe since it means that tasks with $Q_i < 0$ do not need any charging prior to their execution, but introduces pessimism in the analysis as possible surplus energy cannot be captured exactly. Based on this discussion, we give the following lemma.

Lemma 2: The level-i active period of a task τ_i with charging time Q_i can be computed recurrently by:

$$L_i^s = B_i + \sum_{h:\pi_h \geq \pi_i} \left\lceil \frac{L_i^{s-1}}{T_h} \right\rceil \left(C_h + Q_h^+\right) \tag{16}$$

The iteration in (16) starts with $L_i^0 = B_i + C_i$ and stops when $L_i^s = L_i^{s-1}$ or $L_i^s \geq T_H$ where T_H is the hyperperiod of the taskset.

Proof: The proof is straightforward and similar to that in [28] for Theorem 1. The only difference with the regular non-preemptive task analysis mentioned in Theorem 1 is the term $\sum_{h:\pi_h \geq \pi_i} \left\lceil \frac{L_i^{s-1}}{T_h} \right\rceil Q_h^+$, which is to consider the additional delay caused by charging time required for the task itself and its higher priority tasks. This term gives an effect of introducing an artificial task τ_h' to the taskset, which has the same priority as its original counterpart τ_h but runs for the execution time of Q_h^+. ■

Theorem 2: A taskset Γ containing n tasks with energy constraints and arbitrary release offsets is feasible in our scheduling framework if

$$\forall i \leq n, R_i \leq D_i \tag{17}$$

where R_i is the worst-case response time of τ_i given by

$$R_i = \max_{k \leq K_i} \{F_{i,k} - (k-1)T_i\} \tag{18}$$

where K_i is the number of jobs of τ_i in the longest level-i active period, given by

$$K_i = \left\lceil \frac{L_i}{T_i} \right\rceil \tag{19}$$

and $F_{i,k}$ is the finishing time of the k^{th} job of the task τ_i in the level-i active period, given by

$$F_{i,k} = S_{i,k} + C_i \tag{20}$$

In the above equation, $S_{i,k}$ is the starting time of the k^{th} job of τ_i and can be computed recurrently as follows:

$$S_{i,k}^s = B_i + (k-1)\, C_i + \sum_{h:\pi_h > \pi_i} \left(\left\lfloor \frac{S_{i,k}^{s-1}}{T_h} \right\rfloor + 1 \right) C_h + \nu_{i,k}^s \tag{21}$$

where B_i can be obtained from (15). $\nu_{i,k}^s$ is the charging time required for the k^{th} job of τ_i which is calculated at each iteration by

$$\nu_{i,k}^s = k \times Q_i^+ \sum_{h:\pi_h > \pi_i} \left(\left\lfloor \frac{S_{i,k}^{s-1}}{T_h} \right\rfloor + 1 \right) Q_h^+ \tag{22}$$

Proof: The proof is similar to the starting time calculation for non-preemptive tasks in the conventional analysis [28], but considers additional terms to capture the energy requirements

of tasks. Under non-preemptive scheduling, the worst-case response time of a task does not necessarily occur in its first job due to the scheduling anomaly, which can be caused when the higher-priority jobs arrived during the execution of a lower-priority task are pushed to a later time and cause larger delay to its successor jobs. Thus, for a task τ_i, the response time of all K_i jobs during the level-i active period should be calculated and the largest one should be selected as the worst-case response time, as done in (18) and (19). The starting time calculated in (21) is similar to that in [28], except for $\nu^s_{i,k}$ which is the delay caused by charging of the jobs of the task under analysis and its higher-priority tasks. ■

Theorem 2 is a necessary and sufficient condition for the schedulability of a taskset if $\forall Q_i \geq 0$. Otherwise, i.e., $\exists Q_i < 0$, the theorem is a sufficient condition. In addition, it is worth noting that, unlike [10], our analysis holds for arbitrary release offsets so it can be used when the release offsets of tasks are unknown or change after recovery from a power loss.

2) Earliest Deadline First: Unlike the fixed-priority scheduling case, it is not straightforward to compute the worst-case response time of a task under EDF. Hence, we will first transform our scheduling problem to a preemptive scheduling problem, and then analyze schedulability by using the classical utilization bound.

Lemma 3: The scheduling problem of a non-preemptive taskset Γ in our framework can be transformed to that of a conventional preemptive taskset with a shared resource in the following steps.

- Create a mutually-exclusive resource r protected by a lock. Hence, only one task can access r at a time.
- For each task $\tau_i \in \Gamma$, create a new task τ'_i that contains a normal preemptive execution segment of Q^+_i followed by a critical section segment of C_i.
- Make the critical section segments of all tasks access the same shared resource r. Use a real-time synchronization protocol like Stack Recourse Policy (SRP) [29] to ensure that each task can be blocked by at most one critical section segment.
- Replace each task $\tau_i \in \Gamma$ with its counterpart τ'_i.

Proof: The proof is intuitive considering that (i) the charging time of the original task τ_i matches the normal execution segment of the transformed task τ'_i as both are scheduled preemptively from the analysis point of view, and (ii) the execution time of τ_i matches the critical section of τ'_i as they both can block higher-priority tasks. It is worth noting that this transformation does not introduce any pessimism since it essentially changes only the names of execution segments (charging time to normal segments and execution time to critical sections) and the actual schedule of tasks remains unchanged. ■

Based on the above lemma, we can check the schedulability of intermittently-powered tasks under EDF as follows.

Theorem 3: A taskset of n tasks ordered by relative deadlines, i.e., $\forall i, j \leq n, i \leq j \rightarrow D_i \leq D_j$, is schedulable in our scheduling framework with the EDF policy if

$$\forall k = 1, ..., n, \quad \sum_{i=1}^{k} \left(\frac{C_i + Q^+_i}{D_i} \right) + \frac{B_k}{D_k} \leq 1 \tag{23}$$

where B_k is the blocking time of a task τ_k given by

$$B_k = \max_{j:D_j > D_k} C_j \tag{24}$$

Proof: According to the proof in [29], a taskset of n tasks under EDF with the Stack Resource Policy (SRP) is schedulable if

$$\forall k = 1, ..., n, \quad \sum_{i=1}^{k} \left(\frac{C_i}{D_i} \right) + \frac{B_k}{D_k} \leq 1 \tag{25}$$

By considering Lemma 3, the charging time of each task τ_i can be modeled as a critical section segment of its preemptable counterpart. Hence, the utilization of τ_i is changed to $\frac{C_i + Q^+_i}{D_i}$ by adding the charging time. The blocking time remains the same as in SRP since higher-priority tasks can be blocked by at most one non-preemptive execution of lower-priority tasks. Therefore, (25) can be easily extended to (23). ■

3) Sporadic tasks: Sporadic tasks are very common in real-time systems. We show that the proposed scheduling framework and analysis hold for sporadic tasks as well.

Theorem 4: By capturing the minimum inter-arrival time of a sporadic task τ_i as T_i, the analysis presented for periodic tasks can be used to check the schedulability of sporadic tasks.

Proof: For any task with $Q > 0$, the proof is straightforward since any late arrival of a task does not have any adverse effect on the starting time of other tasks. However, since Q can be negative, one may raise the following question: *Can any late arrival of a higher-priority task with a negative Q introduce an additional delay to the starting time of a lower-priority task which has a positive Q?* The negative Q of a higher-priority task may reduce the required charging time of other tasks and thus let them be ready to execute earlier than expected. However, given the definition of Q in (13) and $m_P > 0$, it always holds that $Q_h > -C_h$ for any task τ_h, meaning that the amount of cumulative interference from the higher-priority task (charging time and execution time) can never be negative and it is safely captured by the Q^+_h term in our analysis. Hence, the proof is complete. ■

D. Uncertainties in Energy Supply

When IPDs are deployed in a real environment, energy sources may not strictly follow the model and may not be always available at expected time. Although our periodic energy supply model is more flexible to capture the intermittent availability of energy than the exact arrival patterns used in prior work [10], the problem still exists and makes the real-time scheduling of IPDs more challenging. This subsection presents how an IPD controlled by our framework can deal with uncertainties in energy supply. We first derive an upper bound on the energy charging rate for schedulability, and then analyze the tolerance to occasional energy supply misses and the recovery time from a power loss.

1) Bounding Charging Rate: The schedulability tests presented in Section V-C analyze if a taskset is schedulable when a specific charging rate is given. On the other hand, here we find a sufficient condition on the charging rate for a given taskset. Specifically, our goal is to obtain the least upper bound

(minimum) on the charging rate, M_{min}, which implies that a taskset is guaranteed to remain schedulable under the proposed scheduling framework as long as the charging rate of M_{min} or higher is provided at runtime. To do so, we first give the following lemma.

Lemma 4: The lower bound on the charging rate, M_l, is given by

$$M_l = \sum_{i=1}^{n} \frac{C_i}{T_i} \times m_{Pi} \tag{26}$$

In other words, the taskset is not schedulable if the charging rate m_a is lower than M_l (necessary condition).

Proof: This holds due to (12). ∎

We also find a loose upper bound on the charging rate, M_u.

Lemma 5 (from [29], [30]): A taskset of n tasks where all tasks have relative deadlines equal to their periods is schedulable if

$$\sum_{i=1}^{n} \left(\frac{C_i}{T_i} \right) + \max_{i \leq n} \left\{ \frac{B_i}{T_i} \right\} \leq U(n) \tag{27}$$

where $U(n) = n(2^{1/n} - 1)$ for Rate Monotonic (RM) and $U(n) = 1$ for EDF scheduling.

By extending Lemma 5 with charging tasks, we have

$$\sum_{i=1}^{n} \frac{C_i + Q_i^+}{T_i} + \max_{i \leq n} \left\{ \frac{B_i}{T_i} \right\} \leq U(n) \tag{28}$$

Let us define two subsets of a taskset: $\Gamma_q = \{\tau_q \mid m_{Pq} > m_a^k\}$ and $\Gamma_r = \{\tau_r \mid m_{Pr} \leq m_a^k\}$. Then, by substituting Q_i with (13), we can rewrite the equation (28) as

$$\sum_{i:\tau_i \in \Gamma_q} \frac{m_{Pi} C_i}{m_a T_i} + \sum_{i:\tau_i \in \Gamma_r} \frac{C_i}{T_i} + \max_{i \leq n} \left\{ \frac{B_i}{T_i} \right\} \leq U(n) \tag{29}$$

Then, we can rewrite m_a as

$$m_a \geq \frac{\sum_{i:\tau_i \in \Gamma_q} \frac{m_{Pi} C_i}{T_i}}{U(n) - \left(\sum_{i:\tau_i \in \Gamma_r} \frac{C_i}{T_i} + \max_{i \leq n} \left\{ \frac{B_i}{T_i} \right\} \right)} \tag{30}$$

Based on Lemma 3 and Lemma 5, the loose upper bound on the charging rate, M_u, can be found iteratively using Algorithm 1. The taskset Γ is always schedulable under the proposed framework if the charging rate provided to the device is higher than or equal to M_u.

Although M_u is a safe condition to test schedulability, it is not a tight bound. The least upper bound M_{min} can be found by a binary search between the two bounds, M_u and M_l, and by checking the schedulability test using (16) to (23) for each iteration of the search.

2) Tolerance to Occasional Energy Supply Misses: Based on the least upper bound charging rate, M_{min}, we find the maximum degree of tolerance to occasional energy supply misses if $m_a > M_{min}$.

Lemma 6: Consider the periodic energy supply with the charging time of C_c and the period of T_c. After N_c consecutive period of successful charging, the IPD can tolerate up to N_m periods of charging misses where

$$\frac{N_m}{N_c} \leq \frac{m_a - M_{min}}{m_d + M_{min}} \tag{31}$$

Algorithm 1 Least upper bound on charging rate

```
k = 1
m_a^k = M_l                                   ▷ M_l is obtained by Eq. (26)
m_a^{k+1} = 0
while m_a^{k+1} < m_a^k do
    Γ_q = {τ_i | m_Pi > m_a^k}
    Γ_r = {τ_i | m_Pi ≤ m_a^k}
    if U(n) − Σ_{i:τ_i∈Γ_r}(C_i/T_i) − max_{i≤N}{Bi/Ti} > 0 then
        Compute m_a^{k+1} by Eq. (30)
        M_u = m_a^{k+1}
    else
        M_u = max_{i≤n}{m_Pi}
        break
    end if
    k = k + 1
end while
return M_u
```

Proof: If N_m periods of charging misses happen after N_c consecutive periods of charging, the cumulative amount of charging during the $N_m + N_c$ periods can be calculated as $N_c T_c m_a - N_m T_c m_d$, which should be higher than the minimum charging requirement of the tasks during that time which is $(N_m + N_c)T_c \times M_{min}$. Therefore

$$N_c T_c m_a - N_m T_c m_d \geq (N_m + N_c) T_c \times M_{min} \tag{32}$$

By simplifying the above equation, (31) can be derived. ∎

3) Recovery Time from Device Power Loss: When the energy source is not available for a long time, the device powers off as its voltage level drops below the power-off threshold. To recover from such a power loss, the device may need a long time to accumulate enough energy to start scheduling tasks again. Knowing this recovery time is important because, even after the energy source comes back, the device may not be able to turn on for multiple charging periods.

Lemma 7: Consider t_m, which is the time interval from when the device powers off to when the periodic energy supply comes back. The recover time t_r from when the energy supply is back until the device powers on is given by

$$t_r = \frac{\min\{(t_m + T_c - C_c) m_Y, V_{off}\} + V_{on} - V_{off}}{m_c C_c - m_Y (T_c - C_c)} \times T_c \tag{33}$$

where m_Y and m_c are the decaying and charging rates, respectively, and V_{off} is the power-off threshold of the device.

Proof: Even after the device turns off, it continues to discharge at the rate of m_Y until the voltage decays from V_{off} to zero. The maximum duration of absence of the energy source can be extended to $t_m + (T_c - C_c)$ due to the back-to-back discharging explained in Sec V-B. Therefore, the voltage drop from V_{off} is $\min\{(t_m + T_c - C_c)m_Y, V_{off}\}$, and the voltage needed to turn on the device is $\min\{(t_m + T_c - C_c)m_Y, V_{off}\} + (V_{on} - V_{off})$. After the energy source becomes available, the device starts charging at the rate of $\frac{m_c C_c - m_Y (T_c - C_c)}{T_c}$. Thus, t_r can be found from (33) and the lemma is proved. ∎

VI. Implementation

This section describes the hardware and software implementations of our proposed framework. For experiments on real

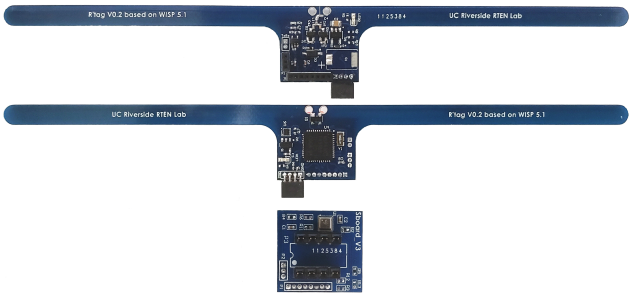


Fig. 6: R'tag and Sensor board

hardware, we developed an RFID-based energy harvesting tag device, called *R'tag*. It follows the design of WISP [8] and has the same MSP430 MCU, RF circuits, and antennas. In addition, we integrated the tag with extra external I/Os that can be used for ADC reading for analog and digital measurements. Furthermore, our device is made available to add a large supercapacitor that could be used to store enough energy to run the device for extended time.

For the sensing purpose, we designed a pluggable sensor board that can be mounted on R'tag and measure high resistance values generated by chemiresistive sensors which are widely used in various sensing applications. It is also equipped with Bosch BME680, an integrated environmental sensor that can measure temperature, pressure, humidity, and total volatile organic compounds in the air. Both R'tag and the sensor board are shown in Fig. 6.

We used an Impinj Speedway Revolution R420 UHF RFID reader that can generate up to 30dBm power to charge the tag. The Ethernet interface is used to connect the RFID reader to the PC to control the power generation of the RFID reader and adjust the RFID setting to generate the desired energy rate. We also used an RFMAX S9028PCL polarized directional antenna which has the transmission gain of $G_s = 8dBi$.

The proposed scheduler is implemented in C language on our designed prototype. The data privatization buffer for task execution [3] is adopted to ensure data consistency and forward progress of the program in case of unexpected power loss. Based on our design, the program running on the device consists of multiple tasks that are non-preemptive. Each task has a separate predefined energy consumption rate which is assigned based on the worst-case energy consumption rate measured in multiple runs. The implemented scheduler keeps track of the energy and manages when to run the tasks so that the schedulability of the taskset as well as timekeeping is guaranteed. Based on our design, task execution is atomic and non-preemptable. Thus, a power loss in the middle of a task's execution causes the task to restart its execution in the next power-on cycle. The results of each completed task are stored in the non-volatile memory, i.e. FRAM of MSP430 MCU, using the double-buffering method [3].

VII. Evaluation

We first conduct schedulability experiments to evaluate the performance of our scheduling framework over prior work. We then present real system experiments using our implementation to demonstrate its effectiveness.

A. Analytical and Simulation Experiments

As discussed in Section II, the only and latest prior work that provides a real-time scheduler for IPDs is the *Celebi* [10] but it assumes preemptive tasks. Hence, in order to compare our scheduling method with Celebi in various scenarios, we implemented the Celebi scheduler such that it allows preemption at the boundary of jobs to deal with non-preemptive tasks. Due to the low processing power of the CPU in most IPDs as well as the limitation of power source and storage capacitor, computation tasks that can be executed on IPDs are considered to be small, e.g., usually less than a second. Furthermore, when the number of tasks increases, the overall feasible utilization of the system may decrease dramatically. In our evaluation scenarios, since the utilization of the taskset is chosen randomly and can be as high as 90%, the number of tasks cannot be very high. Due to the same reason, in previous work such as [12] and [10], only up to 3 and 10 tasks are considered, respectively.

In all experiments, the charging rate is fixed to 3, and deadlines are set equal to task periods ($D_i = T_i$). The experiments for simulation and analysis are conducted in MATLAB on a workstation equipped with an Intel 4.2GHz Core i7 CPU with 16GB of RAM.

We first evaluate the effect of taskset utilization on schedulability. In this experiment, the taskset utilization is chosen from 0.1 to 0.9 in 0.1 steps. For each taskset utilization, 1000 tasksets are generated and the average schedulability ratio of the total 1000 tasksets is reported. The number of tasks, the period, and the discharging rate of each task are chosen randomly from 2 to 20, from 1s to 60s, and from 1 to 10, respectively (all in integer). To generate task execution time, task utilization is first obtained using the UUniFast method [31], then multiplied by the task period, and finally rounded to the nearest positive integer, i.e., $C_i = \max(\lfloor T_i \cdot U_i \rceil, 1)$. The schedulability of each taskset is checked for both our method and Celebi with RM and EDF. Unlike our work, Celebi does not provide an analytical method to test schedulability; instead, it generates a schedule for one hyperperiod and checks if there is a deadline miss or not. To prevent excessive test time when the hyperperiod is long, we limit the taskset generation of Celebi to one hyperperiod or 10000s, whichever is smaller. Also, for convenience, the initial release offset of each task is set to zero. This could be advantageous for Celebi as it does not test all possibilities, but was an inevitable choice to conduct experiments in a reasonable time. For example, a taskset with 10 tasks generated by the above parameters can have a hyperperiod as large as 5.4×10^{16}! In case of our method, we use the schedulability analysis given in Section V-C.

Fig. 7 shows the schedulability ratio as the taskset utilization increases. Our proposed method significantly outperforms Celebi, with as much as 60% higher schedulability ratio. The primary reason why the proposed method yields such an improvement is that it faithfully models the energy-harvesting process of real IPDs [2], [8], [9], where energy harvesting and

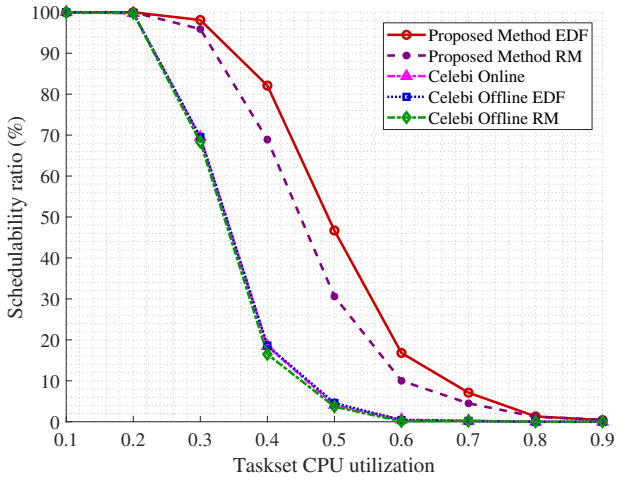


Fig. 7: Scheduler performance for different CPU utilization

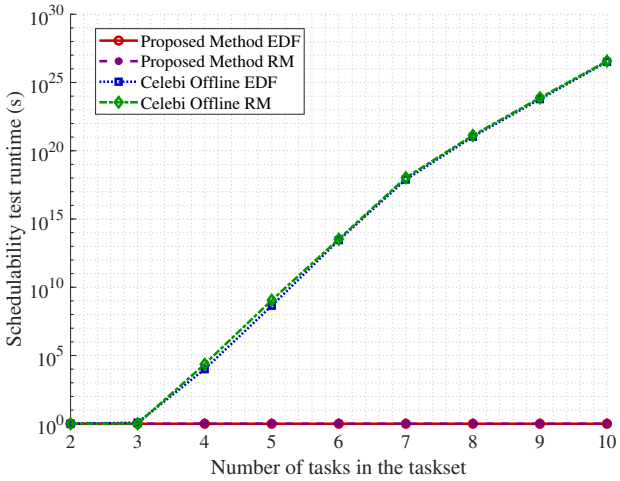


Fig. 9: Computational cost to check taskset schedulability

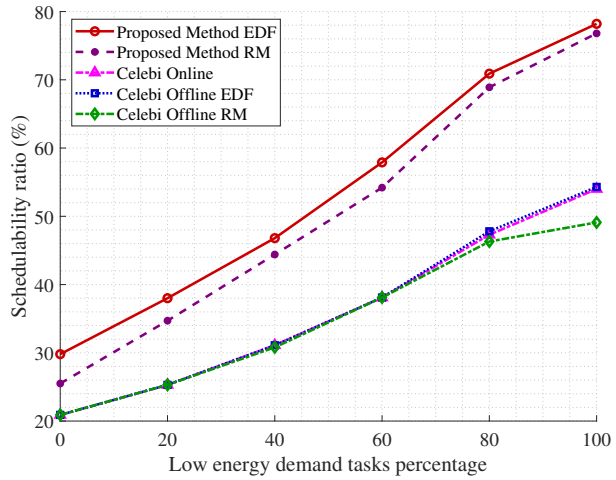


Fig. 8: Scheduler performance for different discharging rate percentages

task execution are not mutually exclusive. Hence, the proposed method allows more time to be available for task execution and ensures not to lose opportunities for charging.

Secondly, we investigate the impact of a discharging rate on schedulability. In this experiment, tasks are divided into two categories: high and low energy demands. The discharging rate is randomly chosen from 8 to 10 for high energy demand tasks, and from 1 to 3 for low energy demand tasks. Fig. 8 depicts the schedulability ratios as the percentage of low energy demand tasks per taskset increases. All other parameters remain the same as the previous experiment, but the total number of tasks per taskset and the taskset utilization are fixed to 5 and 50%, respectively. As shown in the figure, the proposed method gives much higher schedulability than Celebi under both RM and EDF, especially when the percentage of low demand tasks is high. EDF tends to perform better than RM because, as proved in [32], EDF is the optimal scheduler even for non-preemptive task scheduling as long as work-conserving schedulers are considered.

Thirdly, we evaluate the computational cost to test schedulability under our proposed method and Celebi. For our method, the running time of the schedulability analysis given in Section V-C is measured and rounded up to the nearest integer value in seconds. In case of Celebi, generating a schedule for the entire hyperperiod is not practically doable when the number of tasks is large. Therefore, we take the following approach for Celebi: (i) if the hyperperiod of the taskset is below 20000s, a schedule is generated for the entire hyperperiod and the runtime is measured, and (ii) otherwise, we estimate the runtime of Celebi based on the 3-degree polynomial curve fitting, which is derived from the measurements of time duration of 1000s, 2000s, 5000s, and 10000s for a given number of tasks. Testing with a random sample of tasksets, we found that the error of the curve fitting was negligibly small. Fig. 9 shows the results. The schedulability test runtime of our method is extremely small (less than a second in all cases), but that of Celebi increases exponentially with the number of tasks. The reason why Celebi suffers significantly is that the time to generate a schedule depends not only on the length of the hyperperiod but also on the total number of jobs released during the hyperperiod. From our estimation, a taskset with 5 or more tasks would take more than a decade on average in the tested machine to find the schedulability under Celebi. The large size of the generated schedule is also a problem since IPDs can store only tens of KBs in RAM, e.g., up to 64KB in MSP430 [27].

Lastly, we compare the results of our proposed analysis with those from simulation. For simulation, the initial release offset of each task is set to zero by default. If there is a task failed by the analysis, then we find a lower-priority task causing the largest blocking time to the failed task and make that lower-priority task starts 0.1s earlier in simulation (so that the failed task gets the blocking time). Taskset utilization is chosen randomly from 0.1 to 0.9. The other parameters remain the same as in the previous experiments. Fig. 10 shows the results. The deviation of the analysis from simulation is due to the pessimism of the analysis, especially when tasks have discharging rates lower than the charging rate of the device. In addition, the release offsets used for simulation might have

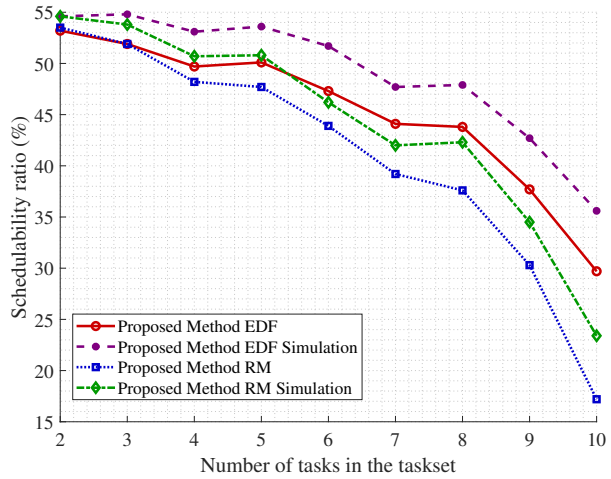


Fig. 10: Simulation and analysis comparison

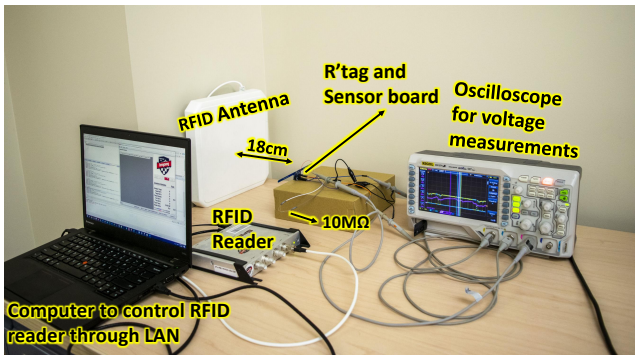


Fig. 11: Experimental setup

not led to the worst possible case. Nonetheless, the degree of pessimism is about 5% for both RM and EDF, which is small compared to the improvement our method has made over Celebi (up to 60% as discussed before). Therefore, we conclude that our proposed method achieves high efficiency and practicality for real-time task scheduling on IPDs.

B. Real System Experiments

Fig. 11 shows the experimental setup used for our experiments. The Impinj RFID reader is connected to the computer via an Ethernet cable for data logging and reader control. The R'tag device is located some distance away from the reader's antenna and a 100 μF capacitor is used for the energy buffer of the device. Our sensor board is attached to the R'tag and a 10 $M\Omega$ resistor is connected to the sensor board as a pseudo chemiresistive sensor. To check task scheduling behavior and device energy level, we instrumented our scheduler code to produce digital output and used an oscilloscope to measure them along with the capacitor voltage. Specifically, we used one I/O pin to show task execution intervals, one pin for charging intervals, and one pin for capacitor voltage monitoring. Our experimental setup is to demonstrate the effectiveness of the analysis and to verify if a real-world scenario follows the analysis. The setup is similar to previous work [20], [22] in that it uses a fixed distance to the energy source.

TABLE I: Task parameters

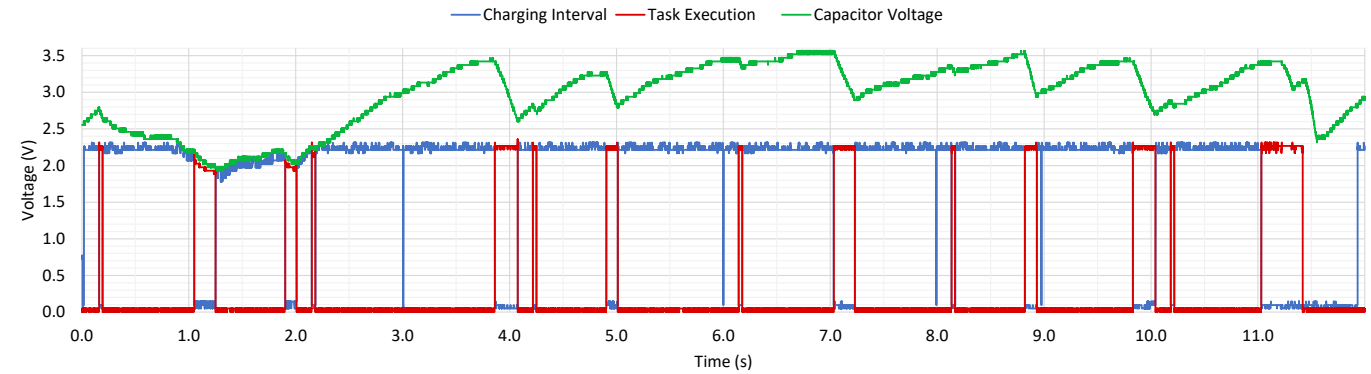
	$C\ (ms)$	$T\ (s)$	$D\ (s)$	π	$m_P\ (mV/s)$
τ_1	32	2	2	4	4400
τ_2	198	3	3	3	4320
τ_3	112	6	2	2	5500
τ_4	387	12	12	1	4000

The taskset we used consists of a mix of sensing and computation tasks. Two computation tasks, τ_1 and τ_2, perform pure CPU processing. Two sensing tasks, τ_3 and τ_4, perform high resistance measurement (τ_3) and interface with the BME680 sensor for temperature, pressure, humidity, and gas data acquisition (τ_4). The sensing and computation tasks follow the read-execute-write semantics [24], so there is no need for synchronization or dependency checking at runtime, as discussed in Section III-C. Task priorities are statically assigned by RM and their parameters are shown in Table I. Based on Lemma 4 and Algorithm 1, we have $M_l = 658\ mV/s$ and $M_u = 1168\ mV/s$ for this taskset, and with a binary search, we find $M_{min} = 587\ mV/s$. Due to some energy waste of I/Os used for experimental purposes, we set $m_a = 600\ mV/s$, which satisfies schedulability since $m_a > M_{min}$. Then, using the equations from (6) to (9), we find that the desired distance from the reader to R'tag is 18 cm when charging constantly.

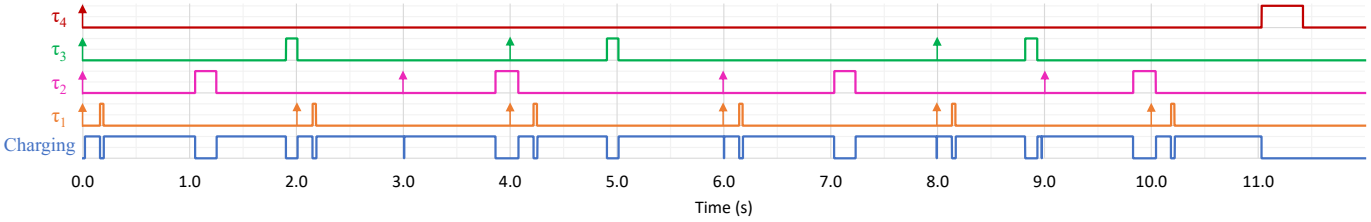
Fig. 12a depicts the task execution and charging intervals and the capacitor voltage of the device during one hyperperiod. Although the tasks are recognizable by their execution times, for the ease of understanding, we illustrate per-task execution in Fig. 12b with arrows indicating job arrival times. As can be seen, when a new job arrives during the charging interval, charging stops promptly (the charging pin goes down) and the scheduler selects the highest-priority task to execute. If the energy demand is not enough, the scheduler lets charging continue and configures the wake-up timer for later execution (e.g., at time 3, 6, and 8). In addition, since task execution is non-preemptive, and lower-priority tasks may block higher-priority tasks (e.g., at time 4, τ_2 blocks τ_1) and this blocking time can be captured by our analysis. In this experiment, the average and the maximum difference between the estimated voltage by (10) and the actual voltage measured from an oscilloscope are 0.21V and 1.17V, respectively. The difference exists due to the fact that the voltage estimation equation considers the worst-case combinations of charging rate, voltage reduction rate and execution time of tasks, but they rarely occur altogether in real-world scenarios. Despite some pessimistic estimations, the overall results show that our scheduling framework works on a real platform as designed and the runtime behavior is predictable by analysis.

VIII. Discussions

Since most IPDs developed in the literature only consider a single capacitor per device, our focus in this paper is on providing a real-time scheduling framework for such devices. However, our work can be extended to IPDs utilizing more than one capacitor bank, e.g., Capybara [2]. To do so, one can divide tasks into multiple groups based on their energy requirements and let each group be served using different



(a) Oscilloscope measurements: blue, red, and green lines are charging, task execution, and capacitor voltage, respectively



(b) Per-task execution and charging intervals reconstructed from the oscilloscope measurements

Fig. 12: Task execution behavior of R'tag under the proposed scheduler

capacitor banks. Thus, each group would have a different charging rate that should be considered when calculating Q_i. Our analysis can then be applied to the entire taskset without any modification. The problem of assigning tasks to different capacitors is analogous to the multiprocessor task allocation problem, which can be solved by using bin-packing heuristics, e.g., tasks can be assigned to capacitors based on their utilization.

In our implementation, a supercapacitor was used but it is not mandatory for our work. Even a small capacitor can be used as long as it is capable of holding enough charges to schedule a given taskset (V_{max} and capacitor charging time affect m_a and consequently affect the schedulability of the taskset). Of course, for an energy source with a long charging period, or for tasks with very long atomic execution and high energy demand, the use of a supercapacitor would help since the maximum voltage of the capacitor is limited and the required energy can only be served by either having a capacitor with higher maximum voltage or increasing the size of the capacitor. However, a larger capacitance does not necessarily improve the schedulability of the system since the time to reach a desired voltage ($t_{charging}$ in (3)) is affected by the capacitor size $\mathbb{C}$.

The runtime scheduler of our framework estimates a lower bound on the current voltage of the capacitor. Although the accuracy of the voltage estimation does not affect taskset schedulability, improving accuracy could help achieve a better performance at runtime than the result predicted by analysis. For example, if a programmable supply voltage supervisor (SVS) is available, the wake up voltage threshold can be set to the desired value and the signal generated from SVS can be used to wake up the device. Therefore, there is no need to estimate the voltage by equations and to set a wake-up timer. However, since a programmable SVS is not widely available, the value of our voltage estimation still holds.

Our periodic energy supply model is motivated by [11] where the RFID charger can move and charge devices periodically. Although our model may not cover all types of energy-harvesting sources, it can be applied to unpredictable energy sources, like wind and vibrations, if their stochastic models are available and the charging rate can be linearized, even though it might be pessimistic. Furthermore, for an energy source that has a set of distinct charging characteristics, one can use our model to build a multi-mode IPD system. Consider solar energy as an example. The amount of harvested solar energy varies in the morning, afternoon, and night of the day. In this case, different charging modes can be considered, and for each mode, a different energy accumulation rate can be calculated and the schedulability of the taskset can be analyzed for each of the modes separately by our analysis.

Lastly, although we have primarily considered the system where all tasks have to meet their deadlines, our work can be applied to imprecise computing-based IPDs such as Zygarde [19]. Consider a set of imprecise-computing tasks, each of which is composed of mandatory and optional subtasks. As their names imply, the mandatory subtasks have to execute by deadlines, but the optional tasks may execute only when possible since their execution does not affect the functional correctness of the system. Then, our analysis can be applied to the mandatory subtasks to check their real-time schedulability. The optional subtasks can be ignored from analysis since their execution can be discarded if the energy is scarce.

IX. Conclusion

In this paper, we proposed a new task scheduling framework for periodic real-time task execution on intermittently-powered

devices (IPDs). Our runtime scheduler design ensures to accumulate enough energy for the safe execution of tasks with indivisible atomic operations. It also enables timekeeping in the presence of intermittent power losses. For analyzable guarantees, our framework captures intermittent energy harvesting as a periodic supply model, provides schedulability tests for both fixed-priority and EDF scheduling, and derives an upper bound on the minimum charging rate, tolerance to occasional energy supply misses, and the recovery time. In our experiments compared to the state of the art, the proposed framework achieved a significant improvement in schedulability (e.g., up to 60% higher schedulability ratio) and verified the schedulability of a given taskset at a much lower computational cost (e.g., taking less than a second while prior work cannot finish in a day). The real system experiments using our hardware and software implementations also demonstrated the practical effectiveness of our work. For future work, we plan to extend our scheduler to deal with different types of external timekeeping methods and to apply it to more complex sensing applications.

REFERENCES

[1] M. Karimi and H. Kim, "Energy scheduling for task execution on intermittently-powered devices," in *Embedded Operating Systems Workshop (EWiLi)*, 2019.

[2] A. Colin, E. Ruppel, and B. Lucia, "A Reconfigurable Energy Storage Architecture for Energy-harvesting Devices," in *ACM SIGPLAN Notices*, vol. 53, 2018.

[3] K. S. Yildirim, A. Y. Majid, D. Patoukas, K. Schaper, P. Pawelczak, and J. Hester, "InK: Reactive Kernel for Tiny Batteryless Sensors," in *SenSys*, 2018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3274783.3274837

[4] K. Maeng and B. Lucia, "Adaptive dynamic checkpointing for safe efficient intermittent computing," in *OSDI*, 2018. [Online]. Available: http://dl.acm.org/citation.cfm?id=3291168.3291178

[5] A. Colin and B. Lucia, "Chain: tasks and channels for reliable intermittent programs," in *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2016, pp. 514–530.

[6] *Low Power Gas, Pressure, Temperature and Humidity Sensor*, Bosch, 06 2020, rev. 1.4.

[7] S. He *et al.*, "Energy provisioning in wireless rechargeable sensor networks," *IEEE Transactions on Mobile Computing*, vol. 12, no. 10, pp. 1931–1942, Oct 2013.

[8] A. P. Sample *et al.*, "Design of an RFID-based battery-free programmable sensing platform," *IEEE Trans. on Inst. and Measurement*, vol. 57, no. 11, pp. 2608–2615, 2008.

[9] H. Jayakumar *et al.*, "QUICKRECALL: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers," in *IEEE International Conference on VLSI Design*, 2014.

[10] B. Islam and S. Nirjon, "Scheduling computational and energy harvesting tasks in deadline-aware intermittent systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020.

[11] Z. Dong, C. Liu, L. Fu, P. Cheng, L. He, Y. Gu, W. Gao, C. Yuen, and T. He, "Energy synchronized task assignment in rechargeable sensor networks," in *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2016, pp. 1–9.

[12] D. Lee, H. Jung, and H. Yang, "Real-time schedulability analysis and enhancement of transiently powered processors with nvms," *IEEE Transactions on Computers*, pp. 1–1, 2020.

[13] J. Hester, K. Storer, and J. Sorber, "Timely Execution on Intermittently Powered Batteryless Sensors," in *SenSys*, 2018.

[14] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on rfid-scale devices," in *ASPLOS*, 2011. [Online]. Available: http://doi.acm.org/10.1145/1950365.1950386

[15] A. Gomez, L. Sigrist, M. Magno, L. Benini, and L. Thiele, "Dynamic energy burst scaling for transiently powered systems," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, ser. DATE '16. San Jose, CA, USA: EDA Consortium, 2016, p. 349–354.

[16] J. de Winkel, C. Delle Donne, K. S. Yildirim, P. Pawełczak, and J. Hester, "Reliable timekeeping for intermittent computing," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 53–67. [Online]. Available: https://doi.org/10.1145/3373376.3378464

[17] J. Hester, N. Tobias, A. Rahmati, L. Sitanayah, D. Holcomb, K. Fu, W. P. Burleson, and J. Sorber, "Persistent clocks for batteryless sensing devices," *ACM Trans. Embed. Comput. Syst.*, vol. 15, no. 4, Aug. 2016. [Online]. Available: https://doi.org/10.1145/2903140

[18] M. Buettner, B. Greenstein, and D. Wetherall, "Dewdrop: An energy-aware runtime for computational rfid," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. USA: USENIX Association, 2011, p. 197–210.

[19] B. Islam and S. Nirjon, "Zygarde: Time-sensitive on-device deep inference and adaptation on intermittently-powered systems," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 4, no. 3, Sep. 2020. [Online]. Available: https://doi.org/10.1145/3411808

[20] T. Wei and X. Zhang, "Gyro in the air: Tracking 3d orientation of batteryless internet-of-things," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 55–68. [Online]. Available: https://doi.org/10.1145/2973750.2973761

[21] J. Hester and J. Sorber, "Flicker: Rapid prototyping for the batteryless internet-of-things," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3131672.3131674

[22] Z. Dong *et al.*, "Enabling Predictable Wireless Data Collection in Severe Energy Harvesting Environments," in *IEEE Real-Time Systems Symposium*, 2017.

[23] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[24] M. Becker *et al.*, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2016.

[25] H. Choi, M. Karimi, and H. Kim, "Chain-based fixed-priority scheduling of loosely-dependent tasks," in *IEEE International Conference on Computer Design (ICCD)*, 2020.

[26] Ambiq Micro, "Ultra-low power rtcs," https://ambiq.com/artasie-am0815/, 2020 (accessed Oct. 2020).

[27] Texas Insturments, "Msp430fr596x, msp430fr594x mixed-signal microcontrollers," https://www.ti.com/lit/ds/symlink/msp430fr5969.pdf, Aug. 2018.

[28] R. J. Bril, J. J. Lukkien, and W. F. J. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited," in *19th Euromicro Conference on Real-Time Systems (ECRTS'07)*, 2007, pp. 269–279.

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

[30] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: an approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, 1990.

[31] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[32] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of period and sporadic tasks," in *[1991] Proceedings Twelfth Real-Time Systems Symposium*, 1991, pp. 129–139.