Delft University of Technology
Master of Science Thesis in Embedded Systems

Keeping Track of Time on Energy Harvesting Systems

Jasper de Winkel

TU Delft Delft University of Technology

Keeping Track of Time on Energy Harvesting Systems

Master of Science Thesis in Embedded Systems

Embedded and Networked Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Jasper de Winkel
J.deWinkel@student.tudelft.nl
jasper.dewinkel@jdew.nl

14-10-2019

Author
Jasper de Winkel

Title
Keeping Track of Time on Energy Harvesting Systems

MSc Presentation Date
28-10-2019

Graduation Committee

Prof. Dr. Koen G. Langendoen (chairman)	Delft University of Technology
Dr. Przemysław Pawełczak (supervisor)	Delft University of Technology
Dr. Remco Litjens	Delft University of Technology, TNO

The work presented in this thesis has lead to a paper which has been submitted to a conference for publication, pending peer-review.

Abstract

Energy-harvesting devices have enabled Internet of Things applications that were impossible before. One core challenge of battery-less sensors that operate intermittently is reliable timekeeping. State-of-the-art low-power real-time clocks suffer from long start-up times (order of seconds) and have low timekeeping granularity (tens of milliseconds at best), often not matching timing requirements of devices that experience numerous power outages per second. Our key insight is that time can be inferred by measuring alternative physical phenomena, like the discharge of a simple *RC* circuit, and that timekeeping energy cost and accuracy can be modulated depending on the run-time requirements. We achieve these goals with a multi-tier timekeeping architecture, named Cascaded Hierarchical Remanence Timekeeper (CHRT), featuring an array of different *RC* circuits to be used for dynamic timekeeping requirements. The CHRT and its accompanying software interface are embedded into a fresh battery-less wireless sensing platform, called Botoks, capable of tracking time across power failures. Low start-up time (max 4 ms), high resolution (up to 1 ms) and run-time reconfigurability are the key features of our timekeeping platform. The timekeeper can be used in a wide range of applications but to demonstrate its usefulness, we developed two battery-less applications that require accurate notion of time: a bicycle analytics tool—where the CHRT is used to track time between revolutions of a bicycle wheel, and a wireless sensing—where the CHRT enables radio synchronization between two intermittently-powered sensors.

Preface

This MSc thesis has been carried out within the Embedded and Networked Systems group at the Delft University of Technology, under the supervision of Dr. Przemysław Pawełczak. The trend towards ever smaller devices combined with battery technology not scaling as fast and environmental reasons, lead to the complete removal of batteries from sensor nodes, known as battery-less sensors. These battery-less sensors bring forth the need for research in intermittent computing. One crucial aspect of most sensor applications is a notion of time. However, little work has been done on timekeeping of intermittently powered battery-less sensors. The presence of such a gap in research hindering development of actual applications, drove me to research this topic. With the result of the work from this thesis I hope to have enabled previously impossible battery-less applications that require a notion of time, inspiring future work on intermittent/battery-less devices.

Now at the end of my MSc journey, I look back with great fondness to the experience. During the introduction week of the master program I attended a presentation from Dr. Pawełczak about intermittent devices. This ultimately lead me to do my thesis with Dr. Pawełczak. Hence, I would like to thank him for this opportunity and guidance throughout my thesis. During my thesis extensive collaborations with the ever positive Dr. Josiah Hester (Northwestern University) only improved my work and drove me further, I would like to thank him for this extensive collaboration. For his continued involvement, interest and advice on the topic and my work I thank Carlo Della Donne. I also would like to thank Dr. Kasım Sinan Yıldırım, for his support and feedback. Finally, I would like to thank Prof. Dr. Koen G. Langendoen and Dr. Remco Litjens for reviewing my work as part of the graduation committee.

With this journey ending, a new and even more exiting journey begins.

Jasper de Winkel

Delft, The Netherlands
14th October 2019

Contents

Chapter 1

Introduction

The continued miniaturization of energy harvesting technology and the increase in computing efficiency has unlocked new application areas where untethered, nearly invisible devices can operate (sense, learn, infer, communicate) in perpetuity. To operate in perpetuity requires the removal of batteries from previously battery powered applications. Removing this main form of energy storage does impose a challenge to fully utilize the tiny amount of available harvested energy. Harvested energy is volatile in nature due to ever changing environmental conditions, meaning that devices relying on harvested energy as their main energy source almost certiantly experience power failures during operation. More importantly, these power failures can be frequent and difficult to predict. How to still make progress and maintain memory consistency on *intermittently* executing devices has been at the forefront of intermittent computing research [16, 33].

Software-based solutions have tried to mitigate the shortcomings of intermittent operation either by instrumenting programs with checkpoints [43, 2, 8], or by rewriting applications using task-based programming models [17, 54, 35]. Hardware and platform approaches have focused on reducing the cost of checkpointing [20], managing energy more efficiently to reduce power failures and increase event detection [14, 15, 8], and getting a rough estimation of time elapsed between power failures [42, 18].

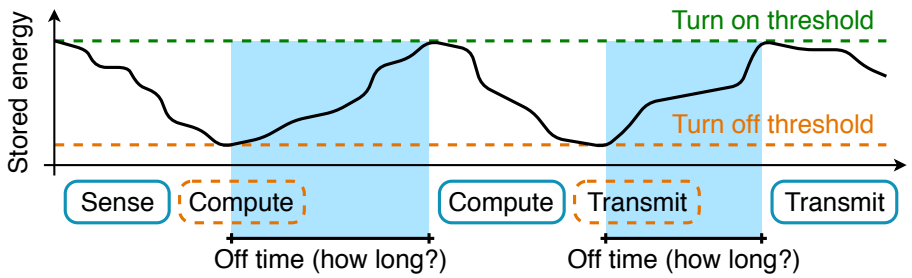


Figure 1.1: **Energy-harvesting battery-less sensors operate intermittently, with execution broken up by power failures of unpredictable duration. These off times are hard to measure with existing timekeepers such as real-time clocks.**

Although in the intermittent computing domain tremendous progress has been made, reliable infrastructure-less deployment of intermittently-powered sensors remains challenging due to the lack of one fundamental feature, *robust timekeeping*. The ability to keep track of time undergirds a multitude of computing and networking primitives such as synchronization and networking, data collection, real-time operation, and security.

1.1 Problem Statement

Having access to an accurate, continuous notion of time has always been taken for granted in embedded system development. Timeouts and timestamps are the backbone of embedded applications, particularly those targeting sensing, communication and actuation. The intermittent operation of ultra-low-energy battery-less devices makes it impossible to track time solely relying on on-chip digital timers, as they are not available during a power outage. Even dedicated ultra-low-power real-time clocks are not a good fit for intermittent operation, as they require very long start-up times[1] and a high initial energy deposit after each power outage.

Importantly, real-time clocks and other embedded systems timekeeping standards are statically provisioned with energy, usually conservatively to allow for the longest possible outage likely to be encountered. The problem with this approach is that the timekeeper's energy buffer (usually, its internal capacitor) is always *over-provisioned* to measure the longest outage, even if short outages are the common occurrence. This notion of over-provisioned timekeepers is summed up in a key insight:

> *Why would we provision a timekeeper to last ten hours when we only need to time an outage of a few seconds?*

As of now, it is impossible to dynamically set timekeeping granularity, energy cost, and start-up time of an embedded timekeeper. Exploiting reconfigurability in software for intermittently-powered runtimes would be aided by a flexible timekeeper. Having power-failure-resilient and adaptive timekeeping would enable new applications, including reliable intermittently-powered radio communication (since intermittently-powered embedded nodes require methods for *synchronizing* local clocks to align wake-up times), timestamped data collection and real-time scheduling.

1.2 Contributions

This work seeks to provide *hardware and software kernel support to enable continuous timekeeping* that is resilient against repeated power failures incurred by *battery-less devices operating intermittently* and adaptive to the dynamic constraints of intermittent computing. The *hardware layer* is inspired by the concept of remanence timekeepers [42, 18, 10] to keep track of time even when the device is depleted of energy. Remanence timekeepers work by discharging a small capacitor over a large resistor during execution and off-time, and measuring

[1] For instance, more than a second for Abracon AB18X5 [38] ultra-low-power real-time clock.

the voltage once power returns to estimate the time elapsed between two reboots. However, as the measured time increases, the resolution of the remanence timekeeper decreases (due to the exponential energy decay of an RC circuit), making it difficult to design a timekeeper that has *millisecond resolution* and *multi-second longevity*. Our observation is that cascading multiple of these cheap RC circuits of different sizes together, arranged hierarchically, can result in both high resolution and long timekeeping range, with low energy cost, low cold-boot time, and small area. The *software layer* abstracts the remanence timekeepers and makes them ready to use by any existing or future runtime for intermittently-powered sensor nodes. The contributions of this work are as follows:

1. *Timekeeping architecture resilient to power failures.* We present a timekeeping architecture, denoted as Cascaded Hierarchical Remanence Timekeeper (CHRT), that enables continuous tracking of time across repeated power failures. With the CHRT, multiple cascaded remanence timekeepers (each with different capacitors sizes) enable any software runtime supporting battery-less (and intermittent) operation to select desired timekeeping resolution and range.

2. *Runtime CHRT support.* A kernel software module accompanies our CHRT architecture to abstract its complexity and to expose a simple and effective API to the programmer. The hardware abstraction layer provides accurate millisecond-scale timekeeping information, and minimizes energy consumption of the CHRT based on run-time energy harvesting conditions.

3. *Battery-less timekeeping sensor.* With the aim to allow developers to experiment with the CHRT and its kernel module, we design and build a hardware platform named Botoks[2], featuring an energy harvester, a CHRT and an ultra-low-power active radio (see Figure 5.1) ready to be used to prototype complete timekeeping battery-less applications.

The timekeeping architecture embedded in Botoks is first characterized independently, and then evaluated inside two case-study applications, emphasizing the importance of timestamps and network synchronization.

[2]Phonetic transcription of BTKS: Battery-less Timekeeping Sensor.

Chapter 2

Related Work

Numerous papers in intermittent computing have promised to revolutionize the use of small sensing devices for diversity of application. We place CHRT (and Botoks) in the context of the state of the art.

2.1 Battery-less Systems and Sensing

Batteries [41] are an obstacle for long-term embedded sensing. In response to this challenge many battery-less sensing platforms have been proposed in the last decade, which we summarized in Table 2.1. Botoks does not require central coordination and an energy provision point, contrary to e.g. backscatter-based nodes [48] or visible light nodes [19]. While some works, rightfully, propose to reconsider the use of batteries in certain sensing applications [30], the huge environmental, maintenance and cost burden of battery-based sensing (that is *the battery itself*) still remains, Many unforeseen applications could be enabled by leaving the battery behind.

Platforms such as Capybara and Flicker [8, 15], both implement different methods of distributing the harvested energy throughout the system: either by a run-time configurable energy storage architecture that can adapt to the application, or by utilizing a localized energy storage approach, where each module has its own energy storage. These platforms both navigate the trade-off space

Platform	Intermitt.-safe	Infrastr.-independent	Time-aware
Hamilton [31]	✗	✓	✗
WISP [48]	✓	✗	✗
BLISP [21]	✓	✗	✗
Capybara [8]	✓	✓	✗
Flicker [15]	✓	✓	✗
Botoks	✓	✓	✓

Table 2.1: **Comparison of selected low-power sensing platforms of the past two decades. Botoks, and its timekeeper (CHRT), are resilient to intermittent operation, and do not depend on external infrastructure.**

between responsiveness and energy efficiency. Whilst both platforms integrate a wireless radio, methods for accurate timekeeping through power failures remain unexplored. Without an external synchronization signal, timekeeping is required to synchronize two intermittent nodes for wireless communication, hence [39] concludes the need for a embedded device such as Botoks that is powered by energy harvesting and facilitates ultra-low power communication.

2.2 Battery-less Timekeeping

The foundational work is [18] where two timekeeping techniques, *TARDIS* and *CusTARD*, were proposed to keep track of time across power failures in battery-less platforms. Both approaches exploit the decay of charge in physical components, SRAM and a capacitor respectively, by comparing a region in SRAM after waking up from power failure to a reference pattern, TARDIS measures the decay and hence time between power failures. However, this method yields poor accuracy and has a high memory overhead. CusTARD, that is also a foundation of Mayfly [17], has all the limitations of capacitive timekeepers listed in Section 4.1. A similar discussion on Mayfly's timekeeping limitations was presented in [11]. Expanding on capacitive timekeepers, [10] introduced a hierarchy of capacitive timekeepers, still suffering however from the same drawbacks as a single capacitive timekeeper as mentioned in Section 4.1.

2.3 Programming Models

Many programming models for intermittenly-powered sensors have been proposed in the past. These can be grouped into checkpoint-based systems—the recent ones being [37, 36], or task-based systems—the recent ones being [54, 35]. None of the checkpoint-based programming models (and very few task-based ones) support time-sensitive data processing. CHRT language constructs enables time-sensitive intermittent computation and sensing. CHRT can be integrated into *any* runtime: task- or checkpoint-based.

2.4 Tools for Battery-less System Development

Over the years several advancements have been made to simplify development of battery-less devices. One of the issues faced is repeatability of energy harvesting conditions during measurements. Small changes to energy harvesting measurements, such as the angle of the device to the energy source or environmental changes, can lead to large variations in the amount of energy harvested. To combat this challenge, emulators/recorders of energy harvesting sources have been developed to aid in the repeatability of measurements [13, 12]. Debugging of intermittent applications also is non-trivial since with every power failure microcontrollers (including their onboard debugger) are reset. A traditional approach would be to add additional communication from the device under test to verify if progress is made and to assert conditions of interest. This however influences the measurements not only in execution time but also in power consumption. Tools for debugging intermittent devices such as EDB [6]

assist by *freezing* the energy source and resuming after the debugging operations have completed.

2.5 Energy Harvesting Sensor Networks and Network Time Synchronization

Since the publication of [10, 18], which is foundational for the work presented here, no new relevant publications have been published on the topic of energy harvesting sensor networks and network time synchronization. Therefore, we kindly refer to [10, Section 5.1.2 and 5.2] for in-depth discussion on this topic.

Chapter 3

Motivation

Ultra-tiny Internet of Things (IoT) devices without batteries present intriguing possibilities for a more sustainable and lower-maintenance computational fabric, with application domains previously considered unfeasible becoming more attainable (like micro-satellites [8], deep-tissue micro-implants [34], and massive scale agriculture via augmented insects [29]).

3.1 Intermittent Operation

Battery-less systems are usually powered by ambient energy and operate intermittently [16, 33, 47], as shown in Figure 1.1. The core of the research in recent years was devoted to the design of efficient runtimes ensuring correctness of computation and memory consistency despite power failures [54, 36, 20, 35, 7, 43, 37]. For embedded devices engaging with real-time constraints and sensor data, efficient computation is only one of multiple necessary capabilities. Today, building and deploying untethered, battery-less IoT devices is challenging because *timekeeping resilient to power failures is imprecise, inaccurate, and statically provisioned.* In the absence of an external synchronization signal (e.g., coming from a controlled light source, or from an RF signal generator), accurate timekeeping support for intermittently-powered devices proves crucial.

		Start-up time		Current
RTC Model	**Resol.**	**Nominal**	**Worst**	**draw**
NXP PCF85263A [46]	10 ms	200 ms	2 s	320 nA
Abracon AB18X5 [38]	10 ms	900 ms	21.5 s	55 nA
ST M41T62 [49]	10 ms	<1000 ms	1 s	350 nA
Maxim DS139X [28]	10 ms	<1000 ms	N/A	500 nA

Table 3.1: **Comparison of selected ultra-low-power RTCs. Note that *all* commercial off-the-shelf RTCs have large start-up times and do not support one millisecond resolution.**

3.2 Timekeeping through Power Failures

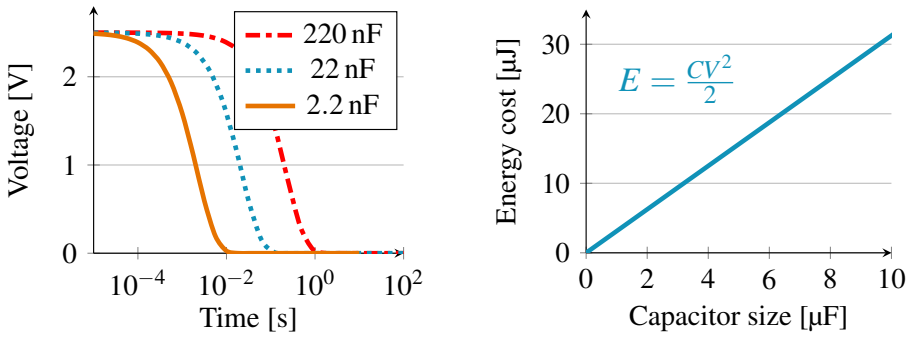
Existing low-power timekeeping solutions present challenges for intermittent operation: either they are not designed for fast restarting and short execution bursts—like real-time clocks—or they rely on signals coming from external infrastructure or the ambient, like light- or RF-based synchronization architectures [53, 19]. Real-time clocks (RTCs) rarely time sub-second intervals, and critically have excessively long cold-boot times (tens of seconds for some low power models if power fails, as in Table 3.1). This excessive boot time is tolerable for the battery-powered devices RTCs were made for, where power failures were rare and failure time generally short, while operation time after failure was orders of magnitude longer. RTC developers traded off cold-boot time for lower power operation and smaller footprint. This trade-off, however, is detrimental for intermittently-powered devices, where power-down failures are usually longer than operational time. These devices lose power even tens of times per second [43, 52], meaning that an RTC on an intermittently-powered device could still be in a cold-boot state when the energy runs out, wasting energy and losing timekeeping accuracy. In addition to cold-boot problems, on-reboot SPI/I2C configuration of these devices wastes valuable energy and time. Network-level sources of time, or sensed signals like visible light communication, power line noise [32], or RF carrier are not always viable, as these sources are not guaranteed to be present in a particular deployment and are susceptible to noise or interference.

3.3 Remanence Timekeepers

The most promising attempt at intermittency-safe, infrastructure-free local timekeeping was *remanence timekeepers*, introduced in Mayfly [17]. A remanence timekeeper is essentially a capacitor discharging through a resistor (RC circuit), whose voltage level is sampled on reboot to get an estimate of the time that has elapsed while the device was powered off. As the voltage decay can be easily modeled and can happen while the MCU is off, the timekeeping accuracy was good enough to enable real-time sensing. Hierarchical remanence timekeepers [10] add multiple individual remenance timekeepers together to enable multiple separate timing ranges.

3.3.1 Trade-Offs

To use a hierarchical remanence timekeeper, the developer is presented with a complex trade-off space between application timing requirements, resolution, and energy available. Tuning the size of capacitor and resistor of a remanence timekeeper is the primary means of controlling the trade-off between timekeeping resolution, range and energy consumption. For instance, larger capacitors take longer to discharge over the same-sized resistor, which means that longer outages can be timed (Figure 3.1a). However, charging a larger capacitor costs more energy at each reboot and more time to charge (Figure 3.1b) and reduces the resolution of measurements because the discharging profile is less steep as compared to smaller capacitors (i.e., the change in voltage is indistinguishable for a 12-bit analog-to-digital-converter). It becomes arduous to find a configuration



(a) **Off time measurement increases with capacitance.**

(b) **Larger timekeepers cost more energy.**

Figure 3.1: **Partial design space for Remanence Timekeepers.**

that can satisfy different needs of an application, balancing and anticipating all these trade-offs at design time.

3.3.2 Timekeeping Rigidity

Beyond the physical constraints, dynamic application behaviors with time-varying timekeeping requirements mean that a single or hierarchy of remanence timekeepers tuned to a *specific outage length* is not useful for outages of a varying length. For example, energy harvesting conditions can quickly change, as a result, a single remanence timekeeper that is sized to time outages measured in milliseconds with high precision cannot give any idea of the passing of time at the minute level. Time accuracy, resolution, and precision requirements are application-dependent and dynamic. Static remanence timekeepers are too rigid to be useful.

3.3.3 Cost, Size and Complexity

Despite the large trade-off space, remanence timekeepers offer attractive benefits due to simplicity of the circuit, allowing for ultra small size and cost. While most RTCs can be upwards of one US dollar even at scale, the discrete components of a remanence timekeeper can be purchased for less than a penny, and even less when built into a modern CMOS process.

Chapter 4

Design

In light of the shortcomings of single remanence timekeepers and RTCs, this work argues for a different approach where multiple remanence timekeepers of increasing capacitance are chained together in hardware. Depleted tiers automatically activate the next smallest tier. For short time intervals the smaller tiers provide higher resolution and consume less energy, while the larger energy-expensive tiers time longer intervals. From this key idea, we explore the complex design space of multi-tier remanence timekeeping with the CHRT, an intermittency-safe timekeeping architecture, and build Botoks, an energy-harvesting device with an on-board CHRT. The benefits of using multiple tiers are summarized below.

Consistent Time Resolution. Compared to single-tier designs, the cascaded multi-tier timekeeper does not suffer from loss of resolution as the discharge curve flattens, meaning that high resolution can still be leveraged after long outages.

Reduced Boot Time/Energy. Boot time for the CHRT is bounded only by the size of the capacitor, unlike for RTCs, where the capacitor must be filled, then the RTC must stabilize, then the RTC registers must be configured. With nearly-instant boot-up and energy only dedicated to timekeeping, the CHRT enables timekeeping through short outages.

Lower Energy Consumption. RTCs and single remanence timekeepers are provisioned with a specific amount of energy at design time, tuned to the best guess of the maximum power failure time. A CHRT can be provisioned at runtime so that the smallest tier that can time the likely length of the next outage is used. In this way the device avoids wasting energy on charging up unnecessarily large capacitors for timing short outages.

Overall Goals. Compared to existing low-power platforms, as in Table 2.1, Botoks features intermittency-safe, infrastructure-independent timekeeping, effectively enabling time-critical intermittent applications. Botoks embeds an ultra-low-power radio as well, ready to be used in combination with CHRT for infrastructure-less intermittent networking applications.

The primary goal of this work is infrastructure-free, high-resolution and high-accuracy timekeeping for intermittently-powered devices. To ensure applicability to a broad range of applications, we develop a software layer around the hard-

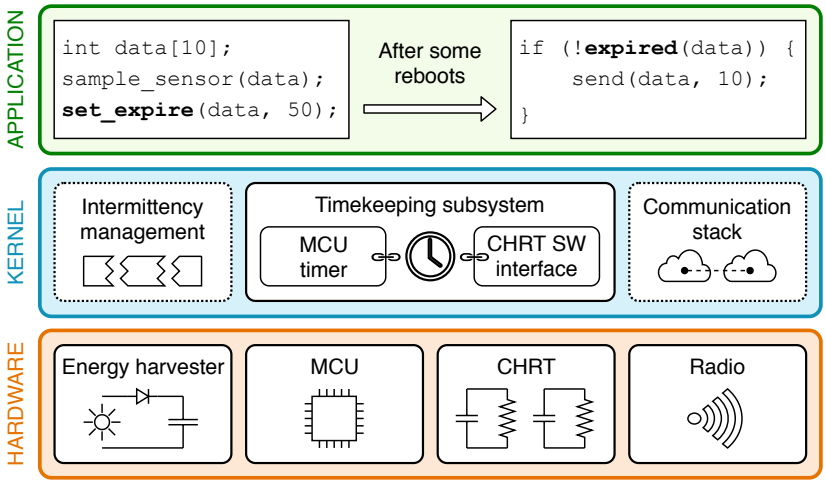


Figure 4.1: **Botoks high-level architecture. The runtime kernel keeps track of time using the CHRT. Applications are instrumented with CHRT functions to infer on the elapsed time despite power outages.**

ware architecture and platform. This allows developers to use the timekeeping functionality without having to delve into the underlying physics of capacitor discharge. This hardware abstraction software layer allows for integration into any intermittency-management runtime like InK [54] or Chinchilla [36]. In Section 4.1 and Section 5.2 we delve into the design aspects of the timekeeping hardware and its accompanying software support, respectively. Both hardware and software layer are integrated into Botoks, as portrayed in Figure 4.1 and described in Section 5, which will be used to evaluate our design (refer to Section 6).

In summary, our goals are:

(1) enable accurate and consistent timekeeping,

(2) allow runtime provisioning of CHRT tiers to enable energy savings, and

(3) provide software constructs for easy usage of the timekeeper in any runtime system.

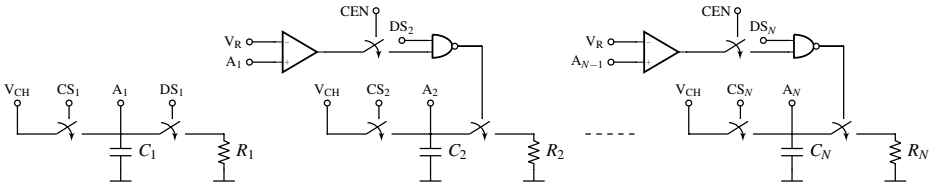


Figure 4.2: **Cascaded Hierarchical Remanence Timekeeper (CHRT). The left-most stage is activated first, while the subsequent stages are activated in a cascaded manner. V_{CH}: regulated output that charges the timekeepers; CS_i: charge signal of tier i; A_i: voltage sampling point of tier i; DS_i: discharge signal of tier i; CEN: cascaded mode enable signal; V_R: comparator reference voltage.**

4.1 CHRT: Hierarchical Timekeeping

The limitations of modern ultra-low-power timekeepers for battery-less and intermittently-powered systems listed in Section 3 motivate our new timekeeping architecture. The main idea is to combine multiple capacitive timekeepers of different sizes into a *Cascaded Hierarchical Remanence Timekeeper* (CHRT). The various *tiers* of the CHRT can be used to minimize cold-boot time and energy consumption, and maximize resolution and timekeeping range at run-time. The tiers are linked together in hardware, from smallest to largest, so that a depleted tier can automatically activate the next tier, therefore increasing the total timing range, whilst maintaining the best possible resolution. For short time intervals, the smaller tiers provide higher resolution and consume less energy. The larger tiers are used to time longer intervals, but have lower resolution and need more charging energy. To be able to use the CHRT, the cascaded tiers have to be pre-charged at each reboot. Our hardware abstraction layer, described in Section 5.2, can be configured to minimize energy consumption depending on the needs of an application and the expectations of energy availability of the environment by specifying how many tiers should be pre-charged at each reboot.

4.1.1 CHRT Circuit

A schematic of the CHRT circuit is shown in Figure 4.2 (only the first two tiers and the last tier are shown). The stable charging voltage V_{CH} is provided by a ultra low power regulator, not shown in the figure. The switches allow the MCU to recharge the capacitors and then to let them discharge through the resistor. The comparator is used to trigger the various stages of the cascade, i.e., a CHRT tier is activated when the voltage across the preceding tier drops below the reference of the comparator (V_R). The control signal CEN can be used to bypass the cascaded behavior and use any of the tiers as an independent timekeeper, providing more flexibility in function. Using the tiers independently, programmers can map tiers to particular functions based on the timing granularity required.

4.1.2 CHRT Range Heuristics

To determine the number of CHRT tiers and their size, timekeeping requirements need to be extracted. These range heuristics can be partially inferred from the

application itself. If data is of no use 10 seconds after it was gathered, then there is little reason to have a timekeeper that can capture periods longer than 10 seconds. Application code for embedded systems is usually full of timing requirements baked into the program. Often these are explicit: many task-based programming models have data timing requirements stored on the edges of the task graph [17]. We also imagine that empirical programming methods could use annotations or assertions to define the timekeeping requirements. Beyond application information, such requirements could be extracted from predictions about the environment. For particularly energy sparse-environments, larger tiers may be required to sustain through interruptions.

Once the time requirements have been extracted (for example, expiration time of data, synchronization granularity of communication), one can decide on the number of tiers and their size by examining the total range required, and the granularity. Since each tier can only cover a portion of the final timing range, we can use simple RC calculations to find these ranges.

Tier's Range. The maximum time interval that a timekeeping tier can measure is limited by the minimum required separation between the final time steps as the capacitor's discharge curve flattens. This minimum separation should be smaller than or equal to the difference in voltage $V_x - V_y$ at the final time step $\Delta t_y = \Delta t_x + \delta t$, assuming V_x and V_y are associated to Δt_x and Δt_y, and the capacitor is charged to V_0, equal to the reference voltage of the ADC. The authors of [10] defined this minimum required separation as

$$V_x - V_y \geq K \frac{V_0}{2^N}, \tag{4.1}$$

where at least K ADC steps with a N-bit ADC of separation is maintained at the final time step. Larger values of K increase robustness against noise but decrease the maximum time interval. To find the maximum time interval of a timekeeping tier at a resolution of δt, authors of [10] derived

$$\Delta t_y \leq RC \ \ln\left(\frac{2^N}{K}\left(\exp\left(\frac{\delta t}{RC}\right) - 1\right)\right) \triangleq \Delta t^{\max}, \tag{4.2}$$

by applying the simple RC discharge model $t = -RC\ln(V/V_0)$, to (4.1) where $\Delta t^{\max}$ is the maximum time a tier can measure, R is the discharge resistor and C being the capacitor charged to V_0 during operation.

Constants N and K are system specific, dependent on the chosen ADC and system noise. In general, R is always desired to be as large as possible to maximize $\Delta t^{\max}$ for the same C, leakage through switches of the CHRT and system noise are the main limiting factors to R. Hence we assume a constant R. For a required resolution δt, C can be increased until the desired maximum time interval requirement Δt is reached. Whilst capacitance C increases, the cut-off voltage at $\Delta t^{\max}$ also increases. Since all energy still contained in the capacitor after the cut-off point is wasted, an increase in capacitance might result in a slightly larger maximum time interval but at the consequence of large amounts of wasted energy.

To combat this, we introduce an additional constraint

$$100\exp\left(\frac{-2\Delta t^{\max}}{RC}\right) \leq H, \tag{4.3}$$

where H denotes the percentage of wasted power, limiting the amount of wasted energy.

Equation (4.2) with constraint (4.3) can be used to obtain the maximum timing interval of an arbitrary tier. A negative value means that the specific RC circuit cannot be used with a resolution δt.

Number of Tiers. Most applications have a variety of accuracy and timing interval requirements, for example, in a critical software section a data sample might expire within a few milliseconds, non critical data samples with 100 ms and the output might be timestamped with the accuracy of one second. This requires a range of different tiers to maintain the accuracy restrictions. Given a set of timekeeping requirements either extracted from program code or given by the developer, expressed in terms of resolution and maximum time-able interval, expression (4.2) with constraint (4.3) can be used multiple times to compute the number of CHRT tiers required and their size. Equally configured tiers can also be duplicated and cascaded, increasing the total time-able interval without reducing the resolution due to a larger capacitance. However, we note that for many applications the default configuration used in Botoks is likely to be suitable (refer to Section 5). This configuration allows for timing outages up to 100 s and has resolution up to a single millisecond.

Tier Selection Heuristic. From earlier sections we know that we can compute the maximum time interval of a single tier but to turn a resolution requirement δt and a maximum time interval requirement Δt into a configuration of tiers we require

$$f(\delta t, \Delta t) \rightarrow (C, K, \Delta\tau), \tag{4.4}$$

where the outputs of $f(\delta t, \Delta t)$ are the tier's capacitance C, the number of required equally sized tiers K and the tier's total timing interval $\Delta\tau$.

The outputs of 4.4 are determined by iteratively increasing C in (4.2) until either the required Δt is reached or constraint (4.3) limits the output, at which point we create multiple identical tiers by $\lceil \frac{\Delta t}{\Delta\tau} \rceil$, resulting in the number of equally sized tiers[1] K.

Now considering a list of N timing requirements, defined as $\boldsymbol{E} = \{\rho_1, \rho_2, \ldots, \rho_N\}$, where $\rho_i = (\delta t_i, \Delta t_i)$ sorted by smallest accuracy first, overlapping requirements are removed and σ is defined as the total timing interval of all tiers. Algorithm 1 can be used to compute a list of suggested tiers $\boldsymbol{S} = \{\gamma_1, \gamma_2, \ldots, \gamma_M\}$ that satisfies the list of requirements $\boldsymbol{E}$.

[1] A full implementation of (4.4) and Algorithm 1, is provided in the Botoks source material [1], published upon acceptance of the paper derived from this thesis.

Algorithm 1 Heuristic for CHRT tier selection based on user requirements

Input: $\boldsymbol{E}$: Array of requirements
Output: $\boldsymbol{S}$: CHRT configuration with capacitance C for each tier γ

$\boldsymbol{S} := []$
$M := 0$
$\sigma := 0$
for all $\rho_i \in \boldsymbol{E}$ **do**
  if $\Delta t_i > \sigma$ **then**
    $(C, K, \Delta\tau) := f(\rho_i)$
    if $\Delta\tau \leq 0$ **then**
      return Failure {Impossible requirements}
    end if
    for $j := 1$ **to** K **do**
      $\boldsymbol{S}[M] := C$
      $M := M + 1$
    end for
    $\sigma := \sigma + \Delta\tau$
  end if
end for

Chapter 5

Implementation

This section lists and briefly discusses the parameters we used for the implementation of CHRT hardware and software, and describes in more detail the components of the integrated Botoks demonstration platform.

5.1 CHRT Platform

The four-tier CHRT was implemented

(1) as a *system peripheral* integrated into Botoks, built with off-the shelf SMD components (see Figure 5.1),

(2) as a *stand-alone* development module, featuring the same components as on Botoks (see Figure 5.2), and

(3) as an *integrated* version.

5.1.1 CHRT Tier Settings

Following the practical guidelines given in Section 4.1.2, we implemented an instance of CHRT targeting a variety of general timekeeping requirements found in sensing and real time devices. To begin with, the value of the discharge resistor was fixed at 22 MΩ for all the tiers, as such resistors are cheap (less than a tenth of a USD cent per unit) and guarantee a good balance between long discharge time of the RC circuit, leakages and electrical noise. Larger resistors would allow for even wider timekeeping ranges, but the incurred noise would become detrimental to the accuracy and the precision of the CHRT. To support a wide range of applications we set the following list of requirements $\boldsymbol{E} = \{(0.001, 0.1), (0.01, 1), (0.1, 10), (1, 100)\}$ (refer to Section 4.1.2 for the explanation of this list). With Algorithm 1 we computed the number of required tiers and their configuration. Matching the ouput of the algorithm, we chose to cascade four tiers of increasing size: $C_1 = 2.2\,\text{nF}$, $C_2 = 22\,\text{nF}$, $C_3 = 220\,\text{nF}$ and $C_4 = 2200\,\text{nF}$. The best time resolution achievable with such configuration is 1 ms, when using the smallest tier, which is guaranteed for capturing time intervals up to 100 ms. The longest measurable interval is 100 s, obtained using the largest tier, for which a resolution of 1 s is guaranteed.

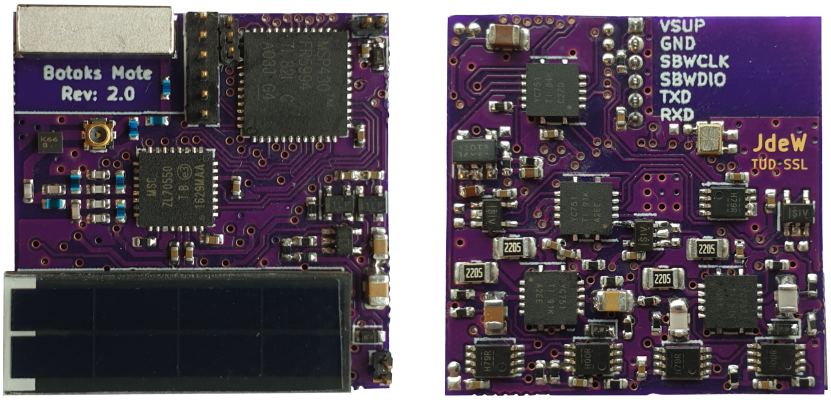


Figure 5.1: **Botoks platform. The front (left) contains radio, antenna, solar panel and MCU. On the back side (right) is the Cascaded Hierarchical Remanence Timekeeper. The board dimensions are 1"×1".**

5.1.2 CHRT with SMD Components

As proof of concept, we built the CHRT with off the shelf components on a custom PCB. Other than the four RC circuits, the CHRT includes other important components. TS3A4751 [26] switches are used to gate power to the capacitors and prevent back feeding to the MCU during power failure. Ultra-low-power D-type Flip Flops (SN74AUP2G79 [24]) protect the CHRT control signals after the MCU dies. To achieve cascaded discharging of the tiers, TLV3691 [25] comparators enable discharging of the next tier when the capacitor voltage of the current tier drops below the voltage at the maximum time interval of the tier, we selected 0.15V as a threshold for each tier. The chosen MCU integrates 1.2V, 2.0V and 2.5V reference voltage options for the ADC peripheral. Since the required energy to charge the capacitor scales quadratically with voltage ($\frac{CV^2}{2}$), the lowest reference voltage (1.2V) is selected. Then by charging the capacitors to 1.2V the full ADC range is utilized. However, lower reference voltages reduce the separation between ADC steps making the system more susceptible to noise.

5.1.3 CHRT Integrated Design

The proof-of-concept PCB is not what we envision will be eventually deployed in the battery-free IoT, an integrated circuit design scales better and is more cost effective in volume. We have also implemented the CHRT in a TSMC 0.18 µm mixed-signal process [51], and simulated the circuit with Cadence Virtuoso [3] in order to extract power consumption estimates of the integrated version of our architecture. The integrated design is significantly lower power and ultra tiny, enabling integration alongside battery-less enabled microcontrollers for low cost instead of expensive crystal centered RTCs. We report the power consumption numbers in Section 6.3.

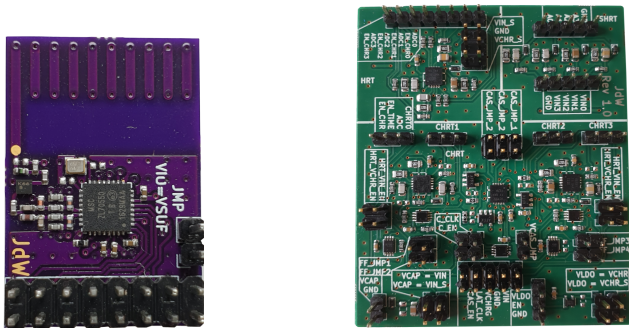


Figure 5.2: **Botoks hardware modules, including distinct development platforms for working with the CHRT (right), and ultra-low-power radio (left). These hardware options provide multiple ways to integrate timekeeping into emerging battery-less applications.**

5.1.4 Botoks Platform

Our battery-less sensor, Botoks, is meant to be used as a complete development board to evaluate and experiment with the CHRT, therefore it also includes an ultra-low-power MCU, an energy harvester and an ultra-low-power radio. Specifically, the device is centered on a Texas Instruments ultra-low-power FRAM-enabled MSP430FR5994 microcontroller [27], with 256 kByte FRAM and 8 kByte SRAM. By default, energy is harvested via the small solar cell on the back of the PCB, and is routed into the 100 µF main storage capacitor. A MIC841 [22] comparator with hysteresis lets the MCU turn on only when sufficient charge is available. The ultra-low-power active radio transceiver is built around the Microsemi ZL70550 [9] chip (as of time of writing, the lowest power 868 MHz radio transceiver available) and a monopole antenna. The complete fabricated device is shown in Figure 5.1. We expect this platform to enable other researchers and practicioners interested in battery-free devices.

5.1.5 Botoks Development Modules

Unlike the fully integrated Botoks platform, we also designed separate hardware modules for both the CHRT and ultra-low-power radio. These modules, depicted in Figure 5.2, are useful for experimenting with our hardware integrated into Botoks on a breadboard or other prototyping setup. The CHRT module integrates both cascaded and non-cascaded hierarchical timekeepers, in addition to debugging features such as jumpers to enable or disable individual tiers.

5.1.6 Software

The code of the CHRT software layer described in Section 5.2 is split into two sub-components, namely a platform-independent layer and a port layer, to give the option to port our software module to other embedded MCU architectures. Our port is written for the target MCU, that is, MSP430FR5994. In addition to the CHRT abstraction layer, we also implemented the necessary drivers to use the ZL70550 [9] radio transceiver. All the code is written in C and can be compiled with the MSP430 GCC compiler [23].

5.2 CHRT Software Layer

The CHRT architecture presented in Section 4.1 is complemented by a software layer, whose aim is twofold:

(1) make the best usage of the available tiers to maximize energy efficiency and timekeeping resolution and range, and

(2) abstract the complexity of the CHRT to provide a user-friendly API.

More specifically, the software layer exposes

(1) a *raw interface*, which can be used to directly request the CHRT hardware to charge the tiers and retrieve elapsed time on reboot, and

(2) a *high-level interface*, which uses the CHRT in combination with a digital timer to provide higher-level functionalities, like timestamp generation and data expiration.

Additionally, the API is responsible for a one-time factory circuit calibration that has to be performed before using the CHRT, as it is typical for RTCs and other integrated circuits.

5.2.1 CHRT Hardware Abstraction Layer

The *raw* CHRT interface is a hardware abstraction layer (HAL) of the underlying timekeeping hardware functionality, to be used for low-level control of the CHRT. It is mostly intended as a building block for more advanced timekeeping duties to be exposed by the runtime or kernel that has knowledge of the user tasks and operations, but can be used at the application level as well by the user. Upon reboot, the runtime calls a function to retrieve the time elapsed since the previous reboot, and then another function to recharge the tiers. The HAL does not have the goal to make the CHRT fully invisible to its user (for instance, the recharging procedure has to be explicitly called). The intermittency-management kernel (like InK [54] or Chinchilla [36]) can use the raw API to define its custom timekeeping functions, or just pass the high-level CHRT interface up to the application layer.

Elapsed Time Retrieval. Upon reboot, `chrt_get_time()` must be called to get the elapsed time of the power failure that was just recovered from. This function returns a 16-bit unsigned integer representing the elapsed time, and a scaling factor. The scaling factor times the elapsed time gives a time value in milliseconds, thus the scaling factor is larger than one, only in case of time intervals larger than $(2^{16} - 1)$ ms.

Dynamic Tier Recharge. After retrieving elapsed time, `chrt_charge()` must be invoked to recharge the CHRT tiers. This function provides a means to specify how many tiers are charged on each reboot, to reduce energy consumption and cold-boot time while still preserving the required timekeeping resolution and range. This function is useful for setting the dynamism of tier recharge to adapt the timekeeping energy expended based on application or environmental properties. Programmers and intermittent kernel designers can choose to be either *conservative* or *adaptive* with tier recharging.

Adaptive timekeeper provisioning is useful when energy environments and application behavior are somewhat predictable or continuous (for example, solar environments). The basic idea of adaptive tier recharge is to choose only the smallest tier that can still satisfy the timing requirements. Specifically, assume that the CHRT is composed of M tiers $\gamma_1, \gamma_2, \ldots, \gamma_M$, chosen as suggested in Section 4.1.2, and that $R_i = \left[t_i^{\min}, t_i^{\max}\right)$ is the optimal timekeeping range of tier γ_i, again, determined as given in Section 4.1.2. We call *target tier* γ_x the tier whose optimal timekeeping range R_x contains the elapsed time retrieved on reboot. Suppose that the CHRT is configured to only charge one tier, the target tier, on reboot. Then, if tier γ_x is charged on reboot j, and the time t retrieved on reboot $j+1$ is not in R_x, the target tier to be charged on reboot $j+1$ would be γ_{x+1} (if it exists) in case $t \geq t_x^{\max}$, or γ_{x-1} (if it exists) in case $t < t_x^{\min}$. The adaptive method saves energy since overcharging the timekeeper when the kernel is only timing a short outage is wasteful.

When the reboot frequency is variable the kernel may choose to be *conservative* in timekeeper provisioning (tier recharging), as retrieved times are more likely to be outside the timekeeping range of the current target tier. For better robustness against variable reboot frequency, the user can request the CHRT to charge more than just the current target tier. Specifically, the two function parameters P_{L} and P_{R} are used to tell the CHRT software layer to charge all tiers in $[\gamma_{x-P_{\mathrm{L}}}, \gamma_{x+P_{\mathrm{R}}}]$, given that γ_x is the current target tier. For instance, if $P_{\mathrm{L}} = P_{\mathrm{R}} = 1$, tiers γ_{x-1}, γ_x and γ_{x+1} would be recharged on reboot, and the discharge would start from tier γ_{x-1}, and continue with the larger tiers in a cascaded fashion. The parameters P_{L} and P_{R} control the trade-off between timekeeping robustness and energy consumption, as charging more tiers requires more energy. In particular, P_{R} has a higher impact on energy consumption, due to larger tiers needing more charging energy.

5.2.2 CHRT High-Level API

The CHRT HAL described previously exposes the most basic functions to control the CHRT. The *high-level* CHRT interface enhances raw functionalities to provide higher-level timekeeping tools to be used in real-world battery-less applications for intermittent devices. Fundamentally, this is implemented combining CHRT functionalities with an on-board MCU digital timer to maintain an always-available system time. The system time is incremented at each reboot using the raw `chrt_get_time()` function. When queried during on-time, the system time is combined with timing information retrieved from the digital timer running in the background. The system time is used to generate timestamps and to set expiration timers for data and functions. Figure 5.3 demonstrates the usage of this high-level API.

Timestamp Generation. The high-level API function `get_timestamp()` uses the aforementioned system time to generate a timestamp when the application requires it. In particular, the returned timestamp is a 32-bit unsigned integer representing system time in milliseconds. When `get_timestamp()` is invoked, the value of the MCU timer is added to the CHRT-powered system time to return a fine-grained timing value that can then decorate data.

Expiration Timers. The high-level timekeeping interface exposes two more functions to keep track of aging data, and discard it when it is deemed expired.

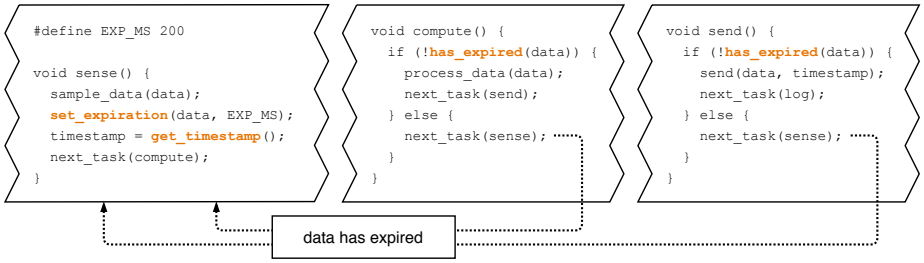


Figure 5.3: **Example usage of the high-level CHRT API. The program is a typical sensing routine, and the API is used to specify that the elapsed time between sensing and transmission of some data must not exceed 200 ms of absolute time (on and off), as well as to get a timestamp to be sent together with the data.**

The function `set_expiration()` can be used to set an expiration time for some data, or for a complete function/task. The user passes a `tag` (of type `void` pointer), representing the object (data or function pointer) to be assigned an expiration time, and an `exp_time` (of type `uint32_t`), to set an expiration time in milliseconds. Then, the API function `has_expired()` can be called, passing a `tag`, to check at any point if some object has expired.

Timekeeping Subsystem. The high-level API can be integrated by a kernel runtime to implement a full timekeeping subsystem for the application layer to use. The runtime must only implement a `timekeeper_init()` function to call during initialization, where system time is updated (using `chrt_get_time()`) and the CHRT is recharged (with `chrt_charge()`).

5.2.3 CHRT Software Calibration

Ideally, the RC circuit discharge model, $t = -RC \ln(V/V_0)$, could be used to estimate elapsed time, t. In actuality, capacitance C and resistance R never match their nominal values, and other parasitic capacitors and resistors are spread through the circuit. To resolve this issue, a software calibration routine, to be performed before CHRT deployment, was devised and implemented, aiming for a better precision and accuracy of the timekeeper.

The one-time calibration consists of two procedures: charge time calibration and accuracy calibration.

Charge Calibration. First, charge time is calibrated. This is achieved by repeatedly charging each tier with an increasing charge time until the ADC saturates. This ensures the full ADC range is used and each tier is properly charged to the reference voltage.

Accuracy Calibration. During accuracy calibration, all the tiers of the CHRT are repeatedly charged and discharged, and their discharging profile is sampled over time to obtain a realistic physical model for each tier. Unfortunately, reality does not perfectly match the the RC circuit discharge model. Therefore, the model is slightly altered by adding an offset G to compensate for slight

inaccuracies such as imperfect charge time calibration. This results in

$$V = V_0 \exp\left(\frac{-t}{RC}\right) + G \Rightarrow t = -RC \ln\left(\frac{V - G}{V_0}\right). \tag{5.1}$$

By curve fitting the previously obtained realistic physical model to the model of (5.1) using nonlinear least-squares, RC and G are obtained for each tier. Now the interpolated version of the RC circuit discharge model can be computed at run-time from (5.1) or by generating a lookup table and storing it in memory at production time. At run-time, the voltage across the target tier is used to look up or compute the corresponding elapsed time.

Chapter 6

Evaluation

To quantify the performance of our system, we first characterize the behavior of the CHRT in terms of accuracy, precision and power consumption. Then, we evaluate it in the context of two case study applications implemented for Botoks, to demonstrate the usefulness of a time-aware intermittently-powered device. In summary, we found that the CHRT is accurate and precise at high resolutions (1 ms) and for long ranges (hundreds of seconds), has two orders of magnitude lower startup time than a RTC, and has an ultra low power consumption, especially in the integrated design.

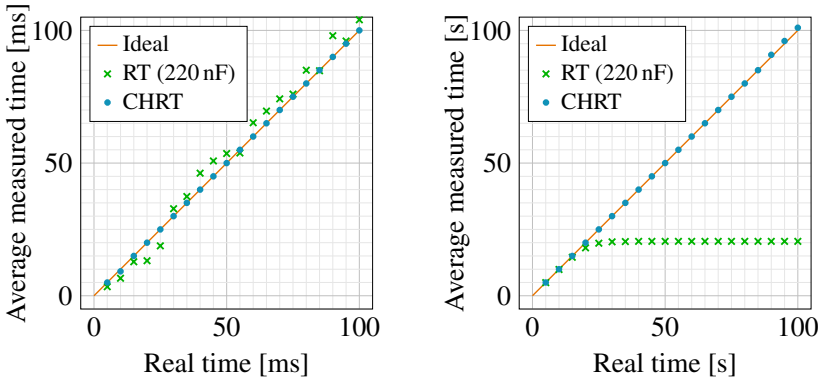
6.1 Experimental Setup

Botoks and its on-board CHRT were used throughout all the experiments. To evaluate timing performance the platform was connected to a continuously-powered microcontroller (a TI MSP430FR5994) that was used as a simulated intermittent power source, in order to control power-on and power-off times. For the case studies, *solar*, *radio frequency* and *magnetic energy* sources were used, as detailed in Section 6.3.1 and Section 6.3.2. To sample digital and analog traces, a Saleae Logic Pro 16 logic analyzer [45] was used. Finally, for power consumption measurements the X-NUCLEO-LPM01A expansion board [50] was used.

6.2 Evaluation Methodology

Our timekeeping architecture is benchmarked both on a fine-grained scale and on an application scale. The *fine-grained benchmarks* targets performance metrics such as timekeeping accuracy and precision, energy consumption and initialization time. The *case-study benchmarks (applications)* showcase the performance of the CHRT when used in real-world scenarios.

We build a bike speedometer using CHRT by capturing elapsed time between consecutive events (revolutions of the wheel). For this type of applications, timekeeping ability and accuracy is necessary to provide reliable readings of the bike speed. However, the constraints on accuracy are not as tight as other real-time systems. For this application we do not rely on the amount of incoming energy to determine the speed such as [47]. The second application is a message



(a) **Tier 0 (2.2 nF) average measured time.**

(b) **Tier 3 (2200 nF) average measured time.**

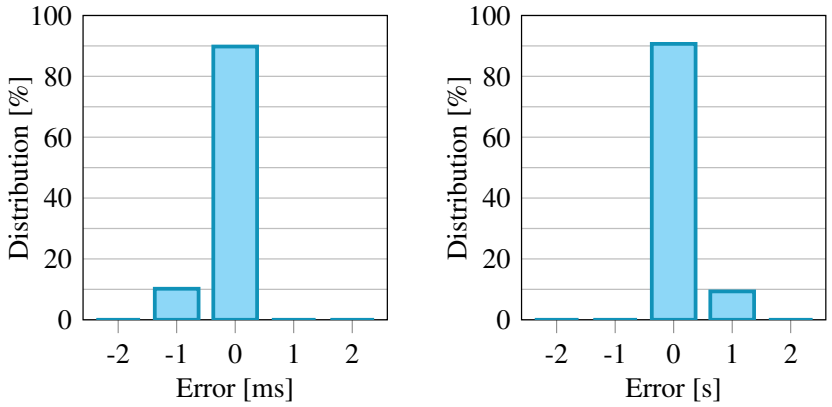
Figure 6.1: **CHRT average measured time for time measurements in the intervals 1 ms to 100 ms (tier 0, resolution of 1 ms) and 1 s to 100 s (tier 3, resolution of 1 s).**

passing protocol that aligns radio communication of two intermittently-powered devices. In this case, robust timekeeping is a stricter requirement, since active radio transmission and reception consume a lot of energy, thus it is crucial for networked nodes to know exactly when to turn on their radio to minimize packet loss and energy waste.

6.3 CHRT Microbenchmark

In order to characterize the isolated performance of the CHRT, a Botoks node was powered by a controlled source that was physically cutting power to the node under test at predefined frequencies. All four tiers of the on-board CHRT were tested extensively across various time ranges and at various resolutions. The smallest tier (2.2 nF) was tested in the range 1 ms to 100 ms, with steps equal to its resolution of 1 ms. Similarly, the other three tiers (22 nF, 220 nF and 2200 nF) where tested in the ranges 10 ms to 1000 ms, 100 ms to 10 000 ms and 1 s to 100 s, respectively, with steps equal to each tier's resolution (10 ms, 100 ms and 1 s, respectively).

CHRT Accuracy. Figure 6.1 plots the accuracy of the smallest tier and the largest tier of Botoks's on-board CHRT. Figures 6.1a and 6.1b show the average reported time of the two tiers, for their optimal timing ranges, where every data point is the average of 10 measurements. The accuracy of a single-tier remanence timekeeper (of 220 nF) is also plotted, to show the need for flexible cascaded multi-tier remanence timekeepers. If the designer selects only a single tier a compromise between resolution and timekeeping range is required. Single tiers have poor results for time ranges longer than what the RC circuit can sustain (the single-tier saturates to around 20 s when trying to capture time intervals larger than that), therefore motivating our multi-tier CHRT.



(a) **Tier 0 (2.2 nF) error.** (b) **Tier 3 (2200 nF) error.**

Figure 6.2: **CHRT error distribution for time measurements in the intervals 1 ms to 100 ms (tier 0, resolution of 1 ms) and 1 s to 100 s (tier 3, resolution of 1 s).**

Timekeeper	**Charge energy (nJ)**	**Start-up time (ms)**
CHRT Tier 0	1.584	0.311
CHRT Tier 1	15.84	0.505
CHRT Tier 2	158.4	0.913
CHRT Tier 3	1584	2.344
CHRT Total	1759.8	4.07

Table 6.1: **CHRT dynamic energy consumption, i.e. energy required to charge each tier, and charging times. The CHRT has orders of magnitude lower start-up time than any state-of-the-art RTC.**

Error Distribution. The same raw measurements were used to generate Figures 6.2a and 6.2b, in which the error distribution of the two tiers is shown. As reported in the plot, the two tiers have a maximum absolute error equal to their respective resolution, which is the case for the other two tiers as well. The error distribution is well centered on zero, and it is very narrow, demonstrating that the CHRT is accurate and precise at high resolutions (1 ms) and for long ranges (hundreds of seconds). The error distribution of a single-tier remanence timekeeper operating outside its optimal timekeeping range has a much higher standard deviation, which is the reason why it was not plotted at all.

Energy Consumption. The static current consumption of the four-tier CHRT amounts to 901 nA for the SMD version (the one embedded in Botoks), and 37.335 nA for the integrated version[1], when powering the circuit at 2.2 V. We

[1]Integrated design: NAND gate, G, consumes 15 pA, comparator, M, 7.43 nA, and the oscillator driving the comparator, S, 15 nA; all values were measured at 1 kHz oscillator frequency. Therefore, for N-tier CHRT the total static current draw is $(N-1)\times(G+M)+S$. Current draw of switches was below 15 nA and therefore removed from the calculation.

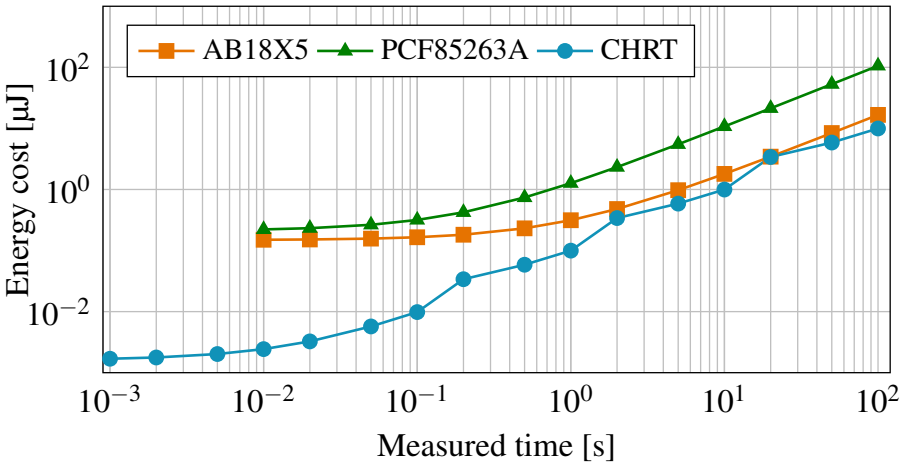


Figure 6.3: **Comparison between CHRT and two RTCs from Table 3.1 of the total energy required to time a period. The CHRT allows for millisecond timing whilst requiring less or similar energy compared to the lowest power RTC.**

also list the amount of energy required to charge the tiers as reported in Table 6.1. In Figure 6.3 the energy required to measure time is reported over the whole time measurement range. CHRT not only measures with much higher accuracy than state-of-the-art RTCs, but also the required energy to measure time is comparable or *lower*, even disregarding configuration time of RTCs.

Initialization Time. Initializing the CHRT boils down to recharging the depleted tiers. The start-up time for each of the tiers embedded in Botoks is reported in Table 6.1. Even when all the four tiers have to be recharged, the initialization time of the CHRT (≈4 ms) is orders of magnitude smaller than for any available RTC (refer to Table 3.1).

Chip Area. The estimated footprint of the integrated four-tier CHRT is less than 1 mm^2 (excluding packaging), proving its suitability for ultra-small embedded systems.

6.3.1 Application 1: Bicycle Analytics

For the bike speedometer application, Botoks's energy harvester was replaced with an off-the-shelf magnetic energy harvester, extracted from a bike light [44], to collect electro-magnetic energy induced by two magnets placed on the rear wheel of a bike. The CHRT is used to time each revolution of the wheel, as Botoks wakes up every time one of the magnets comes close to the harvester. The stored energy is used to charge and sample the CHRT, send a packet containing the calculated speed using Botoks's ULP radio, and power the LEDs on the bike light PCB for the remaining time.

Outcome. The run-time calculated speed, sent by Botoks over the radio and collected by a continuously-powered basestation, was compared to the ground truth, measured with a logic analyzer connected to Botoks's power pin. As

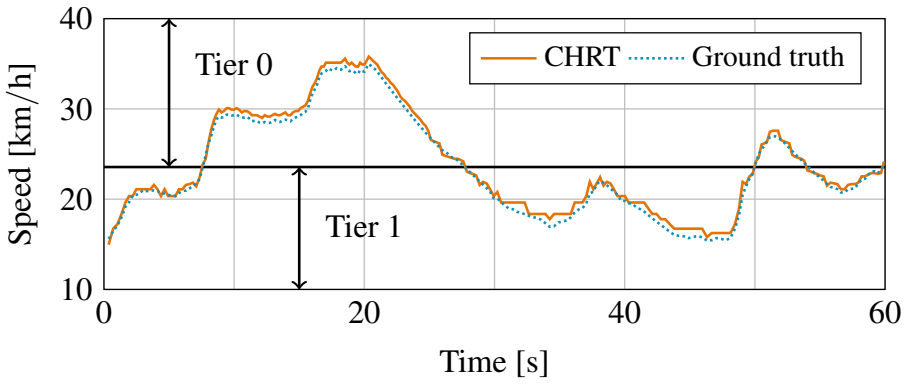


Figure 6.4: **Calculated speed of the bicycle, in km/h, over a period of 60 s. The result obtained with Botoks and its embedded CHRT is compared to the ground truth. As shown, the two smallest tiers of the CHRT are dynamically used through the experiment.**

shown in Figure 6.4, the speed estimated using the CHRT follows very closely the expected result. The figure also shows which of the CHRT tiers is used for different time ranges. Higher speeds, measured with the smallest tier, have better resolution, resulting in a smoother curve in the graph. While the figure only shows a single run lasting 60 s, the same experiment was run five times in total, with a total average RMS error of 0.5 km/h.

6.3.2 Application 2: Intermittent Communication

The second embedded application uses the CHRT to align transmission and reception schedules of two intermittently-powered wireless Botoks nodes communicating via active radios. We have established a point-to-point link with two nodes powered by solar energy in one experiment, and radio-frequency energy in another experiment. The solar energy was generated by two independent light bulbs placed in two closed boxes and collected by Botoks's solar panel. The radio-frequency energy was provided by a Powercast transmitter [5] and harvested by a Powercast receiver [4] connected to Botoks's power supply (in place of the solar cell).

In the application, the transmitting node would wake up to send one packet of data, using its buffered energy, eventually incurring a power failure, proceeding then to harvest energy and finally recharge and wake up again. Similarly, the receiving node would wake up and start listening until receiving a packet, or until a power outage. Each of the 20-Byte packets contained preamble, average on time measured with the CHRT, a dummy payload and 16-b CRC, and was sent at a data rate of 200 kb/s.

The baseline for the experiment is a scheme in which the receiver wakes up and turns on its radio as soon as possible, trying to catch a packet sent by the transmitter. This baseline is compared to the case in which a simple CHRT-powered synchronization protocol allows the receiver to align its listening activity to the transmitter. Specifically, the receiver uses the average transmission period contained in each packet to alter its listening schedule and align to the

	TX		RX			TX		RX	
	Lux	RPS	Lux	RPS		d	RPS	d	RPS
S1	11k	6.6	33k	12.5	**RF1**	0.6 m	12.5	0.6 m	17.3
S2	26k	9.5	26k	9.6	**RF2**	0.8 m	12.9	0.8 m	16.4
S3	26k	10.4	11k	5.9	**RF3**	1 m	11.6	1 m	12.6

Table 6.2: **Description of scenarios described in Figure 6.5. S1, S2, and S3, refer to using solar as energy source and RF1, RF2, and RF3 refer to radio-frequency energy harvesting. For solar energy harvesting the amount of Lux is measured at the solar panel; for radio-frequency energy harvesting the distance to the transmitter is listed. *RPS*: reboots per second.**

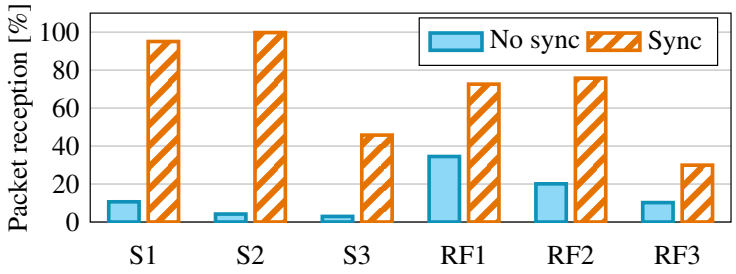


Figure 6.5: **Percentage of received packets for a point-to-point link between two Botoks devices, measured in six different energy harvesting conditions described in Table 6.2. The percentage of packets received using the CHRT-powered synchronization algorithm is compared to the baseline case—with no synchronization. Note that for S3 and RF3 the maximum packet reception available is 50 %, as the receiving node does not have enough energy to wake up as fast as the transmitting node.**

transmitter.

Packet reception rates were measured for the non-synchronized baseline and for the CHRT-powered synchronized protocol, for different energy harvesting conditions, as summarized in Table 6.2. Each experiment was run for 10 min.

Outcome. As it stands out in Figure 6.5 showing the percentage of received packets (the complementary of lost packets), our CHRT-based synchronization algorithm yields an improvement over the non-synchronized message passing scheme, resulting in a best-case increase in received packets of 3.77 times for RF and 23.75 times for solar. Inspecting the difference between energy sources, solar performed better than RF due to a more consistent wake-up period. The Powercast transmitter transmits a ID every 10ms causing an inconsistent wake-up period.

Chapter 7

Discussion and Future Work

We discuss future directions enabled by Botoks and CHRT.

7.1 General Application to Embedded Systems

While the CHRT concept is particularly useful for intermittent computing, where power failures and energy constraints force designers to rethink timekeeping; CHRT is generally applicable to all embedded systems that need timekeeping through power failures and are currently over-provisioned with an RTC. The CHRT offers a modular, lower power, short startup time and energy saving solution over RTCs.

7.2 Tool Support for CHRT

Beyond intermittently-powered networking, there is a dearth of tools for intermittent computing past EDB [6] and Ekho [13]. As complexity increases of the programming models, hardware, and runtime systems, the tradeoff space grows larger. A tool for exploring the tradeoff space of CHRT would be useful for VLSI design as well as prototyping with off-the-shelf components.

7.3 Complex Intermittently-powered Networks

Further exploration is needed on how to expand point-to-point Botoks link into a *full-fledged network* (if the industry estimates on the proliferation of IoT are to be believed [40]). To achieve this, coordinated compute, voting strategies, etc., must be made robust to intermittent power failures. That said, the techniques shown here could form the basis for some of these protocols. Improvements to our synchronization algorithm are required that integrate collisions and increase time accuracy.

7.4 Further Evaluation of CHRT

Naturally, we have not explored all possible corner cases of CHRT operation. A crucial measurement considers CHRT use in varying temperatures. With large variations in temperature, one approach could be to perform a one time temperature calibration. This would consist of a normal calibration cycle but at multiple temperatures. By storing the calibration results at the different temperatures these different calibration constants can be selected at run time by first measuring the temperature with the onboard temperature sensor and then applying the correct set of calibration constants to still achieve the desired accuracy over a wide temperature range.

Chapter 8

Conclusions

We have presented a new battery-less timekeeping mechanism enabling reliable batteryless sensing and computation, the *Cascaded Hierarchical Remanence Timekeeper* (CHRT), and presented two application instantiations useful for intermittent computing: time-critical sensing and intermittent communication. The CHRT is based on the idea of a hierarchy of remanence timekeepers which combine accuracy/resolution, have short cold-start, *and* allow timing long periods of power failure. Combining the CHRT architecture into one hardware and software platform we presented the design and implementation of Botoks—a new batteryless sensor. With Botoks, and its accompanying CHRT, intermittent computing can enter a new phase and explore new applications not possible so far.

Bibliography

[1] Botoks Website. https://github.com/TUDSSL/botoks, October 2019. Last accessed: Oct. 10, 2019.

[2] Naveed Bhatti and Luca Mottola. HarvOS: Efficient Code Instrumentation for Transiently-powered Embedded Devices. In *Proc. IPSN*, pages 209–219, Pittsburgh, PA, USA, 2017. ACM/IEEE.

[3] Cadence Design Systems, Inc. Cadence circuit design. https://www.cadence.com/content/cadence-www/global/en_US/home/tools/custom-ic-analog-rf-design/circuit-design.html, April 2016. Last accessed: Aug. 8, 2019.

[4] Powercast Co. P2110 Powerharvester Evaluation Board. www.powercastco.com/wp-content/uploads/2016/11/p2110-evb1.pdf. Last accessed: Aug. 2018.

[5] Powercast Co. TX91501 Powercaster Transmitter. www.powercastco.com/wp-content/uploads/2016/11/User-Manual-TX-915-01-Rev-A-4.pdf. Last accessed: Aug. 2018.

[6] Alexei Colin, Graham Harvey, Brandon Lucia, and Alanson Sample. An Energy-interference-free Hardware/Software Debugger for Intermittent Energy-harvesting Systems. In *Proc. ASPLOS*, pages 577–589, Atlanta, GA, USA, 2016. ACM.

[7] Alexei Colin and Brandon Lucia. Chain: Tasks and Channels for Reliable Intermittent Programs. In *Proc. OOPSLA*, pages 514–530, Amsterdam, The Netherlands, Oct. 30 – Nov. 4, 2016. ACM.

[8] Alexei Colin, Emily Ruppel, and Brandon Lucia. A Reconfigurable Energy Storage Architecture for Energy-harvesting Devices. In *Proc. ASPLOS*, pages 767–781, Williamsburg, VA, USA, 2018. ACM.

[9] Microsemi Corp. ZL70550 Ultra-Low-Power Sub-GHz RF Transceiver. www.microsemi.com/product-directory/sub-ghz-radio-transceivers/3928-zl70550, April 2019. Last accessed: Apr. 10, 2019.

[10] Carlo Delle Donne. Wake-up alignment for batteryless sensors with zero-energy timekeeping. MSc thesis, Delft University of Technology, Delft, The Netherlands, 2018. http://resolver.tudelft.nl/uuid:e871f33e-7ed2-452b-a962-1de2af9a2906.

[11] Carlo Delle Donne, Kasım Sinan Yıldırım, Amjad Yousef Majid, Josiah Hester, and Przemysław Pawełczak. Backing Out of Backscatter for Intermittent Wireless Networks. In *Proc. ENSsys*, pages 38–40, Shenzhen, China, 2018. ACM.

[12] Kai Geissdoerfer, Mikołaj Chwalisz, and Marco Zimmerling. Shepherd: A Portable Testbed for the Batteryless IoT. In *Proc. SenSys*, New York, USA, 2019. ACM.

[13] Josiah Hester, Timothy Scott, and Jacob Sorber. Ekho: Realistic and Repeatable Experimentation for Tiny Energy-Harvesting Sensors. In *Proc. SenSys*, pages 330–331, Memphis, TN, USA, 2014. ACM.

[14] Josiah Hester, Lanny Sitanayah, and Jacob Sorber. Tragedy of the Coulombs: Federating Energy Storage for Tiny, Intermittently-Powered Sensors. In *Proc. SenSys*, pages 5–16, Seoul, South Korea, 2015. ACM.

[15] Josiah Hester and Jacob Sorber. Flicker: Rapid Prototyping for the Batteryless Internet-of-Things. In *Proc. SenSys*, pages 19:1–19:13, Delft, The Netherlands, 2017. ACM.

[16] Josiah Hester and Jacob Sorber. The Future of Sensing is Batteryless, Intermittent, and Awesome. In *Proc. SenSys*, pages 21:1–21:6, Delft, The Netherlands, 2017. ACM.

[17] Josiah Hester, Kevin Storer, and Jacob Sorber. Timely Execution on Intermittently Powered Batteryless Sensors. In *Proc. SenSys*, pages 17:1–17:13, Delft, The Netherlands, 2017. ACM.

[18] Josiah Hester, Nicole Tobias, Amir Rahmati, Lanny Sitanayah, Daniel Holcomb, Kevin Fu, Wayne P. Burleson, and Jacob Sorber. Persistent Clocks for Batteryless Sensing Devices. *ACM Trans. Embed. Comput. Syst.*, 15(4):77:1–77:28, August 2016.

[19] Kasun Hewage, Ambuj Varshney, Abdalah Hilmia, and Thiemo Voigt. modbulb: A modular light bulb for visible light communication. In *Proc. VCLS*, pages 13–18, New York City, NY, USA, 2016. ACM.

[20] Matthew Hicks. Clank: Architectural Support for Intermittent Computation. In *Proc. ISCA*, pages 228–240, Toronto, ON, Canada, 2017. ACM.

[21] Ivar in 't Veen, Qingzhi Liu, Przemyslaw Pawełczak, Aaron Parks, and Joshua R. Smith. BLISP: Enhancing backscatter radio with active radio for computational RFIDs. In *Proc. RFID*, Orlando, FL, USA, 2016. IEEE.

[22] Microchip Technology Inc. MIC841 Comparator with 1.25% Reference and Adjustable Hysteresis. `http://ww1.microchip.com/downloads/en/DeviceDoc/20005758A.pdf`, 2017. Last accessed: Apr. 10, 2019.

[23] TI Inc. MSP430 GCC Compiler. `http://www.ti.com/tool/MSP430-GCC-OPENSOURCE`. Last accessed: Aug. 10, 2019.

[24] TI Inc. SN74AUP2G79 Low-Power Dual Positive Edge-Triggered D-Type Flip-Flop. `http://www.ti.com/lit/ds/symlink/sn74aup2g79.pdf`, July 2012. Last accessed: Apr. 10, 2019.

[25] TI Inc. MAX9060–MAX9064 Ultra-Small, nanoPower Single Comparators . http://www.ti.com/lit/ds/symlink/tlv3691.pdf, November 2015. Last accessed: Apr. 10, 2019.

[26] TI Inc. TS3A4751 0.9-Ω Low-voltage, single-supply,4-channel SPST analog switch. http://www.ti.com/lit/ds/symlink/ts3a4751.pdf, March 2015. Last accessed: Apr. 10, 2019.

[27] TI Inc. MSP430FR5994 16 MHz Ultra-Low-Power MCU. www.ti.com/lit/gpn/msp430fr5994, January 2018. Last accessed: Apr. 10, 2019.

[28] Maxim Integrated. Low-Voltage SPI/3-Wire RTCs with Trickle Charger. https://datasheets.maximintegrated.com/en/ds/DS1390-DS1394.pdf, October 2010. Last accessed: Apr. 9, 2019.

[29] Vikram Iyer, Rajalakshmi Nandakumar, Anran Wang, Sawyer B. Fuller, and Shyamnath Gollakota. Living IoT: A flying wireless platform on live insects. In *Proc. MobiCom*, pages 1–15, Los Cabos, Mexico, 2019. ACM.

[30] Neal Jackson, Joshua Adkins, and Prabal Dutta. Capacity over Capacitance for Reliable Energy Harvesting Sensors. In *Proc. IPSN*, Montreal, QC, Canada, April 16–18, 2019. ACM/IEEE.

[31] Hyung-Sin Kim, Michael P. Andersen, Kaifei Chen, Sam Kumar, William J. Zhao, Kevin Ma, and David E. Culler. System Architecture Directions for Post-SoC/32-bit Networked Sensors. In *Proc. SenSys*, pages 264–277, Shenzhen, China, 2018. ACM.

[32] Yang Li, Rui Tan, and David K. Y. Yau. Natural timestamping using powerline electromagnetic radiation. In *Proc. IPSN*, pages 55–66, Pittsburgh, PA, USA, 2017. ACM/IEEE.

[33] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. Intermittent Computing: Challenges and Opportunities. In *Proc. SNAPL*, pages 8:1–8:14, Alisomar, CA, USA, 2017.

[34] Yunfei Ma, Zhihong Luo, Christoph Steiger, Giovanni Traverso, and Fadel Adib. Enabling Deep-Tissue Networking for Miniature Medical Devices. In *Proc. SIGCOMM*, pages 417–431, Budapest, Hungary, 2018. ACM.

[35] Kiwan Maeng, Alexei Colin, and Brandon Lucia. Alpaca: Intermittent Execution without Checkpoints. In *Proc. OOPSLA*, pages 96:1–96:30, Vancouver, BC, Canada, 2017. ACM.

[36] Kiwan Maeng and Brandon Lucia. Adaptive Dynamic Checkpointing for Safe Efficient Intermittent Computing. In *Proc. OSDI*, Carlsbad, CA, USA, October 8–10, 2018. USENIX.

[37] Kiwan Maeng and Brandon Lucia. Supporting Peripherals in Intermittent Systems with Just-In-Time Checkpoints. In *Proc. PLDI*, Phoenix, AZ, USA, June 22–26, 2019. ACM.

[38] Ambiq Micro. AM18xx Family Ultra-Low Power RTCs Data Sheet. https://www.ambiqmicro.com/static/rtc/files/AM18X5_Data_Sheet_DS0003V1p3.pdf, February 2019. Last accessed: Apr. 9, 2019.

[39] Alessandro Montanari, Mohammed Alloulah, and Fahim Kawsar. Degradable inference for energy autonomous vision applications. In *Proc. UbiComp*, pages 592–597, London, United Kingdom, 2019. ACM.

[40] Amy Nordrum. Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated. `http://spectrum.ieee.org/tech-talk/telecom/internet/popular_internet_of_things_forecast_of_50_billion_devices_by_2020_is_outdated`, August 2016. Last accessed: Mar. 30, 2018.

[41] M. R. Palacín and A. de Guibert. Why do batteries fail? *Science*, 351(6273), 2016.

[42] Amir Rahmat, Mastooreh Salajegheh, Dan Holcomb, Jacob Sorber, Wayne P. Burleson, and Kevin Fu. TARDIS: Time and Remanence Decay in SRAM to Implement Secure Protocols on Embedded Devices without Clocks. In *Proc. Security*, pages 1–16, Bellevue, WA, USA, 2012. USENIX.

[43] Benjamin Ransford, Jacob Sorber, and Kevin Fu. Mementos: System Support for Long-running Computation on RFID-scale Devices. In *Proc. ASPLOS*, pages 159–170, Newport Beach, CA, USA, 2011. ACM.

[44] Reelight. SL150 Hub Lights. `https://www.reelight.com/products/hub-lights`, August 2019. Last accessed: Aug. 16, 2019.

[45] Saleae. Saleae Logic Pro 16 Analyzer. `http://downloads.saleae.com/specs/Logic+Pro+16+Data+Sheet.pdf`, 2019. Last accessed: Apr. 10, 2019.

[46] NXP Semiconductors. PCF85263 Tiny Real-Time Clock Data Sheet. `https://www.nxp.com/docs/en/data-sheet/PCF85263A.pdf`, November 2015. Last accessed: Apr. 10, 2019.

[47] Uvis Senkans, Domenico Balsamo, Theodoros D. Verykios, and Geoff V. Merrett. Applications of energy-driven computing: A transiently-powered wireless cycle computer. In *Proc. ENSsys*, pages 1–7, Delft, Netherlands, 2017. ACM.

[48] Joshua R. Smith, Alanson P. Sample, Pauline S. Powledge, Sumit Roy, and Alexander Mamishev. A Wirelessly-Powered Platform for Sensing and Computation. In *Proc. UbiComp*, pages 495–506, Orange County, CA, USA, 2006. ACM.

[49] STMicroelectronics. ST M41T62 Low-power serial real-time clock Data Sheet. `https://www.st.com/resource/en/datasheet/m41t62.pdf`, January 2019. Last accessed: Apr. 10, 2019.

[50] STMicroelectronics. X-NUCLEO-LPM01A Nucleo Expansion Board for Power Consumption Measurement. `https://www.st.com/en/evaluation-tools/x-nucleo-lpm01a.html`, April 2019. Last accessed: Apr. 10, 2019.

[51] Taiwan Semiconductor Manufacturing Company, Ltd. 0.18-micron technology. `https://www.tsmc.com/english/dedicatedFoundry/technology/0.18um.htm`, August 2019. Last accessed: Aug. 8, 2019.

[52] Jethro Tan, Przemysław Pawełczak, Aaron Parks, and Joshua R. Smith. Wisent: Robust downstream communication and storage for computational RFIDs. In *Proc. INFOCOM*, pages 1–9, San Francisco, CA, USA, 2016. IEEE.

[53] Kasım Sinan Yıldırım, Henko Aantjes, Przemysław Pawełczak, and Amjad Yousef Majid. On the Synchronization of Computational RFIDs. *IEEE Trans. Mobile Comput.*, 18(9):2147–2159, September 2018.

[54] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemysław Pawełczak, and Josiah Hester. InK: Reactive kernel for tiny batteryless sensors. In *Proc. SenSys*, pages 41–53, Shenzhen, China, 2018. ACM.